

Registration No.

735



BLAS (Basic Linear Algebra Subprograms) in the C# Language

DISTRIBUTION STATEMENT A.

Approved for public release. Distribution is unlimited. Other requests for this document shall be referred to US ARMY RDECOM/TARDEC, Dynamics and Structures Team, ATTN: RDTA-SIE/MS157, Warren, MI 48397-5000

December 2017

U.S. Army Tank Automotive Research,
Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

BLAS (Basic Linear Algebra Subprograms) in the C# Language

W. Bylsma

Analytics/Dynamics

U.S. Army Research, Development and Engineering Command (RDECOM)

U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)

Detroit Arsenal

ATTN: RDTA-SIE/MS157

6501 East 11 Mile Road

Warren, Michigan 48397-5000

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

DISTRIBUTION STATEMENT A: Approved for public release. Distribution is unlimited. Other requests for this document shall be referred to US ARMY RDECOM/TARDEC, Dynamics and Structures Team, ATTN: RDTA-SIE/MS157, Warren, MI 48397-5000.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 12-01-2017		2. REPORT TYPE TECHNICAL		3. DATES COVERED (From - To) 2015-04-02 to 2017-12-22	
4. TITLE AND SUBTITLE BLAS (Basic Linear Algebra Subprograms) in the C# Language				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) WESLEY BYLSMA				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ANALYTICS/DYNAMICS-US ARMY RDECOM/TARDEC DYNAMICS AND STRUCTURES TEAM ATTN: RDTA-SIE/MS157 6501 E 11 MILE RD WARREN, MI 48397-5000				8. PERFORMING ORGANIZATION REPORT NUMBER 735	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement A: Approved for public release. Distribution is unlimited. Other requests for this document shall be referred to US ARMY RDECOM/TARDEC, DYNAMICS AND STRUCTURES TEAM, ATTN: RDTA-SIE/MS157, WARREN, MI 48397-5000.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Document the proof of principle study undertaken to rewrite BLAS in the C# language. Comparison of machine floating point precision, error-exit tests, and computational tests between the FORTRAN 90 and C# implementations for Level 1, 2, and 3 BLAS subroutines were completed successfully. Additional comparison tests included DDOT (Level 1), DGER (Level 2), and DSYMM (Level 3) that matched the output from the Intel Math Kernel Library BLAS Code. No optimization was attempted. This working C# language implementation of BLAS defines a starting point to further optimize BLAS in an object-oriented class based .NET language and explore further expansion into the areas of other operating systems (.NET Core is designed to work on Windows, Linux, and macOS operating systems) and cloud computing environments.					
15. SUBJECT TERMS BLAS, numerical linear algebra, C#					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Unclassified	18. NUMBER OF PAGES 675	19a. NAME OF RESPONSIBLE PERSON Wesley Bylsma
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code) 586-282-4104

Contents

1.0 INTRODUCTION	1
2.0 IMPLEMENTATION	2
2.1 FORTRAN CONVERSION	2
2.2 C# LAPACK/BLAS SUBROUTINE CONVERSION	2
2.3 ARRAY ISSUES	3
2.4 TEST CALLING PROGRAMS	3
2.4.1 DBLAT1	3
2.4.1.1 FORTRAN 90.....	3
2.4.1.2 C#	4
2.4.2 DBLAT2	5
2.4.2.1 SUMMARY DATA	5
2.4.2.2 FORTRAN 90.....	6
2.4.2.3 C#	6
2.4.3 DBLAT3	7
2.4.3.1 SUMMARY DATA	7
2.4.3.2 FORTRAN 90.....	7
2.4.3.3 C#	7
2.4.4 ZBLAT1.....	8
2.4.4.1 FORTRAN 90.....	8
2.4.4.2 C#	8
2.4.5 ZBLAT2.....	9
2.4.5.1 SUMMARY DATA	9
2.4.5.2 FORTRAN 90.....	9
2.4.5.3 C#	9
2.4.6 ZBLAT3.....	10
2.4.6.1 SUMMARY DATA	10
2.4.6.2 FORTRAN 90.....	10
2.4.6.3 C#	10
3.0 OUTPUT COMPARISON	10
3.1 DBLAT1	11
3.2 DBLAT2	12
3.3 DBLAT3	13
3.4 ZBLAT1.....	14
3.5 ZBLAT2.....	14
3.6 ZBLAT3.....	16
4.0 LICENSING	17
5.0 CONSLUSION.....	18
APPENDIX A.1 – cs_BLAS.cs	19
class version	19
PRINT	19
TSTIEE.....	19
ILAENV.....	19
IEEECK.....	25
class misc_d	26

DLAMCH.....	26
DLAMC1.....	27
DLAMC2.....	29
DLAMC3.....	32
DLAMC4.....	32
DLAMC5.....	32
DSECND.....	34
DSECNDTST	34
MYSUB.....	35
DROTG.....	35
DNRM2.....	35
DROTMG.....	35
DBLAT1.....	35
GET_DARR	35
GET_DARR_FULL	35
GET_DARRF.....	35
GET_DARR2D.....	36
GET_DARR2DF.....	36
HEADER.....	36
CHECK0.....	36
CHECK1.....	38
CHECK2.....	38
CHECK3.....	41
STEST.....	43
TESTDSDOT	44
SDIFF.....	44
ITEST1.....	44
DDOT.....	45
DAXPY.....	45
DROTG.....	46
DROT.....	46
DCOPY.....	47
DSWAP.....	47
DNRM2.....	48
DASUM.....	48
DSCAL.....	49
IDAMAX.....	49
DROTMG.....	49
DROTM.....	51
DSDOT.....	52
DBLAT2.....	53
GET_2DFROM1D.....	58
PRINT_2DFROM1D.....	58
COPY_1D.....	59
GET_1DFROM2D.....	59
GET_1DFROM2D_STARTI	59
GET_1DFROM2D_STARTI_FULL.....	59
DCHK1.....	59
DCHK2.....	63
DCHK3.....	67
DCHK4.....	71
DCHK5.....	74

DCHK6	78
DCHKE	83
DMAKE	86
DMVCH	88
LDE	89
LDERES	90
DBEG	90
DDIFF	91
CHKXER	91
XERBLA	91
DGBMV	91
DGEMV	93
DGER	95
DSBMV	96
DSPMV	98
DSPR	100
DSPR2	101
DSYMV	102
DTRSV	104
DTRMV	106
DSYR	107
DSYR2	109
DTBMV	110
DTBSV	112
DTPMV	115
DTPSV	117
DBLAT3	119
DBLAT3_DCHK1	124
DBLAT3_DCHK2	126
DBLAT3_DCHK3	129
DBLAT3_DCHK4	132
DBLAT3_DCHK5	135
DBLAT3_DCHKE	138
DBLAT3_DMAKE	143
DMMCH	144
DGEMM	145
DSYMM	147
DTRMM	148
DTRSM	151
DSYRK	154
DSYR2K	156
class misc_z	158
ZBLAT1	158
GET_DARR	158
GET_ZARR	158
GET_ZARR2D	159
GET_ZARR3D	159
GET_ZARR_FULL	159
HEADER	159
CHECK1	159
CHECK2	162

STEST	166
SDIFF	166
CTEST	166
ITEST1	167
ZDOTC	168
ZDOTU	168
ZAXPY	169
DROTG	169
DROT	170
ZCOPY	171
ZSWAP	172
DZNRM2	173
DCABS1	173
DZASUM	174
ZSCAL	175
ZDSCAL	175
IZAMAX	176
DROTMG	176
DROTM	178
DSDOT	179
ZBLAT2	179
XERBLA	214
ZGEMV	219
ZGBMV	222
ZHEMV	225
ZHBMV	227
ZHPMV	229
ZTRMV	231
ZTBMV	234
ZTPMV	238
ZTRSV	241
ZTBSV	244
ZTPSV	248
ZGERC	251
ZGERU	252
ZHER	253
ZHPR	255
ZHPR2	256
ZHER2	258
ZBLAT3	260
ZGEMM	295
ZHEMM	299
ZSYMM	301
ZTRMM	303
ZTRSM	307
ZHERK	311
ZSYRK	314
ZHER2K	316
ZSYR2K	319
class misc_s	322
SLAMCH	323

SECOND.....	330
SECONDTST	330
MYSUB	331
APPENDIX A.2 – DBLAT1.f90	332
LSAME	332
DLAMCH.....	333
DLAMC1	334
DLAMC2.....	336
DLAMC2.....	338
DLAMC4.....	339
DLAMC5.....	339
DSECND.....	341
DSECNDTST	341
SLAMCH.....	342
SLAMC1	344
SLAMC2.....	345
SLAMC3.....	347
SLAMC4.....	348
SLAMC5.....	349
SECOND.....	350
SECONDTST	350
MYSUB_S.....	351
TSTIEE.....	352
ILAENV.....	352
IEEECK.....	357
DBLAT1	358
HEADER.....	359
CHECK0	359
CHECK1	361
CHECK2	361
STEST	366
TESTDSDOT	367
STEST1	367
SDIFF	367
ITEST1	368
DDOT	368
DAXPY	369
DROTG.....	370
DROT	371
DCOPY	372
DSWAP	373
DNRM2	374
DASUM.....	375
DSCAL	376
IDAMAX.....	377
DROTMG.....	378
DROTM.....	380
DSDOT	382
APPENDIX A.3 – DBLAT2.f90	384
LSAME	384

DBLAT2	386
DCHK1	389
DCHK2	392
DCHK3	395
DCHK4	398
DCHK5	401
DCHK6	404
DCHKE	407
DMAKE	410
DMVCH	411
LDE	413
LDERES	413
DBEG	414
DDIFF	414
CHKXER	414
XERBLA	414
DGEMV	416
DGBMV	419
DSBMV	423
DSPMV	426
DTRMV	429
DTBMV	432
DTPMV	435
DTBSV	438
DTPSV	442
DGER	445
DSYR	447
DSPR	449
DSYR2	451
DSPR2	454
DSYMV	457
DTRSV	459
APPENDIX A.4 - DBLAT3.f90	462
LSAME	462
DBLAT3	464
DCHK1	466
DCHK2	469
DCHK3	471
DCHK4	475
DCHK5	477
DCHKE	480
DMAKE	485
DMMCH	486
LDE	487
LDERES	487
DBEG	487
DDIFF	488
CHKXER	488
XERBLA	488
DGEMM	490
DSYMM	494

DTRMM.....	497
DTRSM.....	501
DSYRK.....	504
DSYR2K.....	508
APPENDIX A.5 – ZBLAT1.f90	511
LSAME.....	511
ZBLAT1.....	512
HEADER.....	513
CHECK1.....	513
CHECK2.....	515
STEST.....	517
STEST1.....	517
SDIFF.....	518
CTEST.....	518
ITEST1.....	518
ZDOTC.....	518
ZDOTU.....	519
ZAXPY.....	520
ZCOPY.....	521
ZSWAP.....	522
DZNRM2.....	523
DCABS1.....	524
DZASUM.....	524
ZSCAL.....	525
ZDSCAL.....	526
IZAMAX.....	527
LSAME.....	529
ZBLAT2.....	531
ZCHK1.....	534
ZCHK2.....	537
ZCHK3.....	540
ZCHK4.....	543
ZCHK5.....	545
ZCHK6.....	548
ZCHKE.....	551
ZMAKE.....	554
ZMVCH.....	555
LZE.....	556
LZERES.....	557
ZBEG.....	557
DDIFF.....	558
CHKXER.....	558
XERBLA.....	558
ZGEMV.....	560
ZGBMV.....	563
ZHEMV.....	566
ZHBMV.....	570
ZHPMV.....	573
ZTRMV.....	576
ZTBMV.....	579
ZTPMV.....	583

ZTRSV	586
ZTBSV	590
ZTPSV	593
ZGERC	596
ZGERU	598
ZHER	600
ZHPR	603
ZHER2	605
ZHPR2	608
LSAME	611
ZBLAT3	612
ZCHK1	615
ZCHK2	618
ZCHK3	620
ZCHK4	623
ZCHK5	626
ZCHKE	629
ZMAKE	637
ZMMCH	639
LZE	640
LZERES	640
ZBEG	641
DDIFF	642
CHKXER	642
XERBLA	642
ZGEMM	644
ZHEMM	648
ZSYMM	652
ZTRMM	655
ZTRSM	659
ZHERK	663
ZSYRK	666
ZHER2K	670
ZSYR2K	674

BLAS (Basic Linear Algebra Subprograms) in the C# Language

TARDEC Technical Report No. 735
December 2017

1.0 INTRODUCTION

Equations are a central part of solving engineering problems. In particular, equations of motion in dynamics or equations of stiffness in structural analysis are used to find solutions for these problem domains. Numerical methods for solving these equations include the Finite Element Method (FEM). Many of these equations are linear and therefore Linear Algebra can be used to great advantage. The usefulness of Linear Algebra is clearly demonstrated by its ability to solve least squares problems [4]. In Linear Algebra, scalar, vector, and matrix operations are used to perform computations and find solutions to equations.

There has been a great amount of work done in the area of Linear Algebra resulting in BLAS [1]. From [1a] a concise description follows

Basic Linear Algebra Subprograms (BLAS) is a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. They are the de facto standard low-level routines for linear algebra libraries; the routines have bindings for both C and Fortran. Although the BLAS specification is general, BLAS implementations are often optimized for speed on a particular machine, so using them can bring substantial performance benefits. BLAS implementations will take advantage of special floating point hardware such as vector registers or SIMD instructions.

It originated as a Fortran library in 1979[1] and its interface was standardized by the BLAS Technical (BLAST) Forum, whose latest BLAS report can be found on the netlib website.[2] This Fortran library is known as the reference implementation (sometimes confusingly referred to as the BLAS library) and is not optimized for speed but is in the public domain.[3][4]

Most libraries that offer linear algebra routines conform to the BLAS interface, allowing library users to develop programs that are agnostic of the BLAS library being used. Examples of BLAS libraries include: AMD Core Math Library (ACML), ATLAS, Intel Math Kernel Library (MKL), and OpenBLAS. ACML is no longer supported by its producer.[5] ATLAS is a portable library that automatically optimizes itself for an arbitrary architecture. MKL is a freeware[6] and proprietary[7] vendor library optimized for x86 and x86-64 with a

performance emphasis on Intel processors.[8] OpenBLAS is an open-source library that is hand-optimized for many of the popular architectures. The LINPACK benchmarks rely heavily on the BLAS routine GEMM for its performance measurements.

Many numerical software applications use BLAS-compatible libraries to do linear algebra computations, including Armadillo, LAPACK, LINPACK, GNU Octave, Mathematica,[9] MATLAB,[10] NumPy,[11] R, and Julia.

and from [1]

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

BLAS is an amazing resource that can be used as a foundation to solve problems as outlined above. Its adaption for use in cloud computing or solving in-house problems is restricted because it is written in FORTRAN. While BLAS implementations are available in other languages they usually are binary only or in native code. One example is DotNumerics in [5], but even this implementation provides only “double” routines, not “complex” also, and does not include all the testing routines (DBLAT1, ZBLAT1, etc.). More recent languages, such as C#, enjoy support in cloud computing environments, are object-oriented class based, and have extensive development environments (.NET, Visual Studio). While these newer languages may not perform as fast as native code (C++, etc.) their platform flexibility is quite desirable; .NET Core is designed to work on Windows, Linux, and macOS operating systems.

With the above thoughts in mind, a proof of principle study was undertaken to rewrite BLAS in the C# language.

2.0 IMPLEMENTATION

Using LAPACK 3.5.0 [5], the reference BLAS FORTRAN was modified to work as FORTRAN 90 for comparing results in Microsoft Visual Studio (Intel Parallel Studio XE 2011 with Microsoft Visual Studio 2010 (.NET Framework 4.7)). The FORTRAN code was then converted to C#. The system specification on which the results were obtained are

Windows 7 Enterprise (w/ Service Pack 1)
Intel Xeon CPU E5-2699 v3 @ 2.30GHz 2.30 GHz (2 processors)
192 GB RAM
64-bit Operating System

2.1 FORTRAN CONVERSION

The conversion to FORTRAN 90 included

- replacing “*” comment characters with “!”
- adding the “&” continuation character at the end of split lines
- removing EXTERNAL and DEC\$ declarations
- changing keyword PROGRAM (DBLAT1, etc.) to SUBROUTINE
- including END FUNCTION or END SUBROUTINE at the end of each
- placing functions/subroutines inside the CONTAINS statement and calling them from the main program

2.2 C# LAPACK/BLAS SUBROUTINE CONVERSION

The BLAS subroutines converted were based on LAPACK 3.5.0. Several of the LAPACK numerical tests were included, but the focus was the BLAS double and complex Level 1, Level 2, and Level 3 test

subroutines---including any other code necessary for their output to match the FORTRAN 90 equivalent. A list of the main starting subroutines is shown below.

LAPACK

- DLAMCH – determines double precision machine parameters
- SLAMCH – determines float (single) precision machine parameters
- DSECNDTST – timing tests for DAXPY
- SECONDTST – timing tests for SAXPY
- TSTIEE (ILAENV, IEECK) – numerical tests (LAPACK)

BLAS

- DBLAT1 – test double precision Level 1 BLAS
- DBLAT2 – test double precision Level 2 BLAS
- DBLAT3 – test double precision Level 3 BLAS
- ZBLAT1 – test double complex Level 1 BLAS
- ZBLAT2 – test double complex Level 2 BLAS
- ZBLAT3 – test double complex Level 3 BLAS

The source code for each FORTRAN 90 and C# subroutine is included in the Appendix. This can be used to compare implementation differences in each language. Comments used for debugging have been left to help in this effort, but they can be deleted if so desired. No optimization has been attempted. The focus was to get a working C# version that matches the FORTRAN 90 results.

For details of the individual subroutines reference should be made to [1] and [2].

2.3 ARRAY ISSUES

Arrays in FORTRAN 90 are handled differently than in C#. In general, FORTRAN uses column major storage while C# uses row major. FORTRAN 90 also allows array “reshaping” where a one dimensional array can be passed to a two dimensional array and it will automatically be resized. In C#, this is accomplished with several “helper” functions that convert each array to its proper form, before passing it as a parameter, if necessary.

2.4 TEST CALLING PROGRAMS

Test calling programs for each BLAS test listed in section 2.2 is documented below.

2.4.1 DBLAT1

Test for double precision Level 1 BLAS.

2.4.1.1 FORTRAN 90

```

program testdblat1
implicit none
DOUBLE PRECISION :: X(9), Y(5)
INTEGER :: I
print *, 'Epsilon' = ',DLAMCH('E')
print *, 'Safe minimum' = ',DLAMCH('S')
print *, 'Base' = ',DLAMCH('B')
print *, 'Precision' = ',DLAMCH('P')
print *, 'Number of digits in mantissa' = ',DLAMCH('N')
print *, 'Rounding mode' = ',DLAMCH('R')
print *, 'Minimum exponent' = ',DLAMCH('M')
print *, 'Underflow threshold' = ',DLAMCH('U')
print *, 'Largest exponent' = ',DLAMCH('L')
print *, 'Overflow threshold' = ',DLAMCH('O')
print *, 'Reciprocal of safe minimum' = ',1/DLAMCH('S')
print *, '====='
print *, 'Epsilon' = ',SLAMCH('E')
print *, 'Safe minimum' = ',SLAMCH('S')
print *, 'Base' = ',SLAMCH('B')
print *, 'Precision' = ',SLAMCH('P')
print *, 'Number of digits in mantissa' = ',SLAMCH('N')
print *, 'Rounding mode' = ',SLAMCH('R')
print *, 'Minimum exponent' = ',SLAMCH('M')
print *, 'Underflow threshold' = ',SLAMCH('U')
print *, 'Largest exponent' = ',SLAMCH('L')
print *, 'Overflow threshold' = ',SLAMCH('O')

```

```

        print *, ' Reciprocal of safe minimum   = ',1/SLAMCH('S')
        print *, '====='
        CALL DSECNDTST()
        print *, '====='
        CALL SECONDTST()
        print *, '====='
        CALL TSTIEE()
        print *, 'TSTIEE Done.'
        CALL DBLAT1
        print *, 'DBLAT1 Done.'
!https://software.intel.com/en-us/node/522288
        DO 10 I=1,5
            X((I-1)*ABS(2)+1) = 2.0D0
            Y((I-1)*ABS(1)+1) = 1.0D0
        10 CONTINUE
        print *, DDOT(5,X,2,Y,1)
CONTAINS
    (see Appendix A.2)
end program testdblat1
!
!       Specifies the value to be returned by DLAMCH:
!
!       = 'E' or 'e',   DLAMCH := eps
!       = 'S' or 's',   DLAMCH := sfmin
!       = 'B' or 'b',   DLAMCH := base
!       = 'P' or 'p',   DLAMCH := eps*base
!       = 'N' or 'n',   DLAMCH := t
!       = 'R' or 'r',   DLAMCH := rnd
!       = 'M' or 'm',   DLAMCH := emin
!       = 'U' or 'u',   DLAMCH := rmin
!       = 'L' or 'l',   DLAMCH := emax
!       = 'O' or 'o',   DLAMCH := rmax
!
!       where
!
!       eps   = relative machine precision
!       sfmin = safe minimum, such that 1/sfmin does not overflow
!       base  = base of the machine
!       prec  = eps*base
!       t     = number of (base) digits in the mantissa
!       rnd   = 1.0 when rounding occurs in addition, 0.0 otherwise
!       emin  = minimum exponent before (gradual) underflow
!       rmin  = underflow threshold - base**(emin-1)
!       emax  = largest exponent before overflow
!       rmax  = overflow threshold - (base*emax)*(1-eps)

```

Execute in Windows PowerShell

```
.\testdblat1.exe
```

2.4.1.2 C#

```

namespace testcsblas1
{
    class dblat1
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();
            Console.WriteLine("Epsilon") = {0:E15}", cs_BLAS.misc_d.DLAMCH("E"));
            Console.WriteLine("Safe minimum") = {0:E15}", cs_BLAS.misc_d.DLAMCH("S"));
            Console.WriteLine("Base") = {0:E15}", cs_BLAS.misc_d.DLAMCH("B"));
            Console.WriteLine("Precision") = {0:E15}", cs_BLAS.misc_d.DLAMCH("P"));
            Console.WriteLine("Number of digits in mantissa") = {0:E15}", cs_BLAS.misc_d.DLAMCH("N"));
            Console.WriteLine("Rounding mode") = {0:E15}", cs_BLAS.misc_d.DLAMCH("R"));
            Console.WriteLine("Minimum exponent") = {0:E15}", cs_BLAS.misc_d.DLAMCH("M"));
            Console.WriteLine("Underflow threshold") = {0:E15}", cs_BLAS.misc_d.DLAMCH("U"));
            Console.WriteLine("Largest exponent") = {0:E15}", cs_BLAS.misc_d.DLAMCH("L"));
            Console.WriteLine("Overflow threshold") = {0:E15}", cs_BLAS.misc_d.DLAMCH("O"));
            Console.WriteLine("Reciprocal of safe minimum") = {0:E15}", 1/cs_BLAS.misc_d.DLAMCH("S"));
            Console.WriteLine("=====");
            Console.WriteLine("Epsilon") = {0:E7}", cs_BLAS.misc_s.SLAMCH("E"));
            Console.WriteLine("Safe minimum") = {0:E7}", cs_BLAS.misc_s.SLAMCH("S"));
            Console.WriteLine("Base") = {0:E7}", cs_BLAS.misc_s.SLAMCH("B"));
            Console.WriteLine("Precision") = {0:E7}", cs_BLAS.misc_s.SLAMCH("P"));
            Console.WriteLine("Number of digits in mantissa") = {0:E7}", cs_BLAS.misc_s.SLAMCH("N"));
            Console.WriteLine("Rounding mode") = {0:E7}", cs_BLAS.misc_s.SLAMCH("R"));
            Console.WriteLine("Minimum exponent") = {0:E7}", cs_BLAS.misc_s.SLAMCH("M"));
            Console.WriteLine("Underflow threshold") = {0:E7}", cs_BLAS.misc_s.SLAMCH("U"));
            Console.WriteLine("Largest exponent") = {0:E7}", cs_BLAS.misc_s.SLAMCH("L"));
            Console.WriteLine("Overflow threshold") = {0:E7}", cs_BLAS.misc_s.SLAMCH("O"));
            Console.WriteLine("Reciprocal of safe minimum") = {0:E7}", 1 / cs_BLAS.misc_s.SLAMCH("S"));
            Console.WriteLine("=====");
            cs_BLAS.misc_d.DSECNDTST();
            Console.WriteLine("=====");
            cs_BLAS.misc_s.SECONDTST();
            Console.WriteLine("=====");
            cs_BLAS.version.TSTIEE();
            Console.WriteLine("TSTIEE Done.");

            cs_BLAS.misc_d.DLAMCH("E");
            cs_BLAS.misc_s.SLAMCH("E");
        }
    }
}

```

```

cs_BLAS.misc_d.DBLAT1(); // requires misc_d.eps [DLAMCH("E")], misc_s.eps[SLAMCH("E")]
Console.WriteLine("DBLAT1 Done.");
//https://software.intel.com/en-us/node/522288
double[] x = new Double[9];
double[] y = new Double[5];
int i;
for (i=1;i<=5;i=i+1)
{
    x[ (i-1)*Math.Abs(2)+1] -1] = 2.0d;
    y[ (i-1)*Math.Abs(1)+1] -1] = 1.0d;
}
Console.WriteLine("{0,20:F13}",cs_BLAS.misc_d.DDOT(5,x,2,y,1));
/*
    Specifies the value to be returned by DLAMCH:
    *      = 'E' or 'e',    DLAMCH := eps
    *      = 'S' or 's',    DLAMCH := sfmin    // SFMIN = TINY(ZERO); SMALL = ONE / HUGE(ZERO)
    *                                          IF( SMALL.GE.SFMIN ) THEN
    *                                          //Use SMALL plus a bit, to avoid the possibility of rounding
    *                                          //causing overflow when computing 1/sfmin.
    *                                          SFMIN = SMALL*( ONE+EPS )
    *                                          END IF
    *      = 'B' or 'b',    DLAMCH := base    // RMACH = RADIX(ZERO)
    *      = 'P' or 'p',    DLAMCH := eps*base // RMACH = EPS * RADIX(ZERO)
    *      = 'N' or 'n',    DLAMCH := t      // RMACH = DIGITS(ZERO)
    *      = 'R' or 'r',    DLAMCH := rnd     // RMACH = RND
    *      = 'M' or 'm',    DLAMCH := emin    // RMACH = MINEXPONENT(ZERO)
    *      = 'U' or 'u',    DLAMCH := rmin    // RMACH = tiny(zero)
    *      = 'L' or 'l',    DLAMCH := emax    // RMACH = MAXEXPONENT(ZERO)
    *      = 'O' or 'o',    DLAMCH := rmax    // RMACH = HUGE(ZERO)
    *
    *                                          // DIGITS(X)      The number of significant digits
    *                                          // EPSILON(X)     The least positive number that added to 1
    *                                          //              returns a number that is greater than 1
    *                                          // HUGE(X)       The largest positive number
    *                                          // MAXPONENT(X)   The largest exponent
    *                                          // MINEXPONENT   The smallest exponent
    *                                          // PRECISION(X)  The decimal precision
    *                                          // RADIX(X)     The base in the model
    *                                          // RANGE(X)     The decimal exponent
    *                                          // TINY(X)      The smallest positive number
    *
    *      where
    *
    *      eps = relative machine precision
    *      sfmin = safe minimum, such that 1/sfmin does not overflow
    *      base = base of the machine
    *      prec = eps*base
    *      t = number of (base) digits in the mantissa
    *      rnd = 1.0 when rounding occurs in addition, 0.0 otherwise
    *      emin = minimum exponent before (gradual) underflow
    *      rmin = underflow threshold - base**(emin-1)
    *      emax = largest exponent before overflow
    *      rmax = overflow threshold - (base**emax)*(1-eps)
    */
}
}

```

Execute in Windows PowerShell

```
.\testcsblas1.exe
```

2.4.2 DBLAT2

Test for double precision Level 2 BLAS.

2.4.2.1 SUMMARY DATA

```

'dblat2.out'      NAME OF SUMMARY OUTPUT FILE
6                UNIT NUMBER OF SUMMARY FILE
'DBLAT2.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
-1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F              LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F              LOGICAL FLAG, T TO STOP ON FAILURES.
T              LOGICAL FLAG, T TO TEST ERROR EXITS.
16.0           THRESHOLD VALUE OF TEST RATIO
6              NUMBER OF VALUES OF N
0 1 2 3 5 9     VALUES OF N
4              NUMBER OF VALUES OF K
0 1 2 4         VALUES OF K
4              NUMBER OF VALUES OF INCX AND INCY
1 2 -1 -2      VALUES OF INCX AND INCY
3              NUMBER OF VALUES OF ALPHA
0.0 1.0 0.7     VALUES OF ALPHA
3              NUMBER OF VALUES OF BETA
0.0 1.0 0.9     VALUES OF BETAC
DGBMV T PUT F FOR NO TEST. SAME COLUMNS.
DSBMV T PUT F FOR NO TEST. SAME COLUMNS.
DSBMV T PUT F FOR NO TEST. SAME COLUMNS.
DSBMV T PUT F FOR NO TEST. SAME COLUMNS.
DSBMV T PUT F FOR NO TEST. SAME COLUMNS.
DTRMV T PUT F FOR NO TEST. SAME COLUMNS.

```

```
DTBMV T PUT F FOR NO TEST. SAME COLUMNS.
DTPMV T PUT F FOR NO TEST. SAME COLUMNS.
DTRSV T PUT F FOR NO TEST. SAME COLUMNS.
DTBSV T PUT F FOR NO TEST. SAME COLUMNS.
DTPSV T PUT F FOR NO TEST. SAME COLUMNS.
DGER T PUT F FOR NO TEST. SAME COLUMNS.
DSYR T PUT F FOR NO TEST. SAME COLUMNS.
DSPR T PUT F FOR NO TEST. SAME COLUMNS.
DSYR2 T PUT F FOR NO TEST. SAME COLUMNS.
DSPR2 T PUT F FOR NO TEST. SAME COLUMNS.
```

2.4.2.2 FORTRAN 90

```
program testdblat2
implicit none
DOUBLE PRECISION AA(9), X(10), Y(10)
INTEGER I,J,LDA
CALL DBLAT2
print *, 'DBLAT2 Done.'
!https://software.intel.com/en-us/node/522288
DO 10 I=1,10
  X(I) = 1.0D0
  Y(I) = 1.0D0
10 CONTINUE
LDA = 3 ! Row major
DO 11 I= 1, 2
  DO 12 J=1,3
    AA(((I-1)+(J-1)*LDA)+1) = (J-1)+1
12 CONTINUE
11 CONTINUE
CALL DGER(2,3,0.5D0,X,2,Y,1,AA,LDA)
DO 13 I= 1, 2
  DO 14 J=1,3
    print *, ( (I-1)+(J-1)*LDA)+1,AA(((I-1)+(J-1)*LDA)+1)
14 CONTINUE
13 CONTINUE
CONTAINS
  (see Appendix A.3)
end program testdblat2
```

Execute in Windows PowerShell

```
get-content dblat2_summry.txt | & ".\testdblat2.exe"
```

2.4.2.3 C#

```
namespace testcsblas2
{
    class dblat2
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();

            cs_BLAS.misc_d.DLAMCH("E");
            cs_BLAS.misc_s.SLAMCH("E");

            cs_BLAS.misc_d.DBLAT2(); //requires misc_d.eps [DLAMCH("E")]
            Console.WriteLine("DBLAT2 Done.");
            //https://software.intel.com/en-us/node/522288
            double[] aa = new Double[9];
            double[] x = new Double[10];
            double[] y = new Double[10];
            double[,] tmp2d_aa;
            int i,j,lda;
            for (i = 1; i <= 10; i = i + 1)
            {
                x[i - 1] = 1.0d;
                y[i - 1] = 1.0d;
            }
            lda = 3;
            for (i = 1; i <= 2; i = i + 1)
            {
                for (j = 1; j <= 3; j = j + 1)
                {
                    aa[((i - 1) + (j - 1) * lda) + 1 - 1] = (j - 1) + 1;
                }
            }
            tmp2d_aa = cs_BLAS.misc_d.get_2dfrom1d(3 * 3, 0, lda, aa);
            //DGER(m, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
            cs_BLAS.misc_d.DGER(2, 3, 0.5d, x, 2, y, 1, ref tmp2d_aa, lda);
            aa = cs_BLAS.misc_d.get_1dfrom2d(3 * 3, 0, lda, tmp2d_aa);
            for (i = 1; i <= 2; i = i + 1)
            {
                for (j = 1; j <= 3; j = j + 1)
                {
                    Console.WriteLine("{0,20:F13} {1,20:F13}", ((i - 1) + (j - 1) * lda) + 1, aa[((i - 1) + (j - 1) * lda) + 1 - 1]);
                }
            }
        }
    }
}
```

```

    }
}
}

```

Execute in Windows PowerShell

```
get-content dblat2_summry.txt | & ".\testcsblas2.exe"
```

2.4.3 DBLAT3

Test for double precision Level 3 BLAS.

2.4.3.1 SUMMARY DATA

```

'dblat3.out'      NAME OF SUMMARY OUTPUT FILE
6                UNIT NUMBER OF SUMMARY FILE
'DBLAT3.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
-1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F                LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F                LOGICAL FLAG, T TO STOP ON FAILURES.
T                LOGICAL FLAG, T TO TEST ERROR EXITS.
16.0            THRESHOLD VALUE OF TEST RATIO
6                NUMBER OF VALUES OF N
0 1 2 3 5 9      VALUES OF N
3                NUMBER OF VALUES OF ALPHA
0.0 1.0 0.7      VALUES OF ALPHA
3                NUMBER OF VALUES OF BETA
0.0 1.0 1.3      VALUES OF BETA
DGEMM T PUT F FOR NO TEST. SAME COLUMNS.
DSYMM T PUT F FOR NO TEST. SAME COLUMNS.
DTRMM T PUT F FOR NO TEST. SAME COLUMNS.
DTRSM T PUT F FOR NO TEST. SAME COLUMNS.
DSYRK T PUT F FOR NO TEST. SAME COLUMNS.
DSYR2K T PUT F FOR NO TEST. SAME COLUMNS.

```

2.4.3.2 FORTRAN 90

```

program testdblat3
implicit none
DOUBLE PRECISION AA(12), BB(8), CC(8)
INTEGER LDA, LDB, LDC, I, J
CALL DBLAT3
print *, 'DBLAT3 Done.'
!https://software.intel.com/en-us/node/522288
LDA = 4
LDB = 4
LDC = 4
DO 11 I= 1, 3
DO 12 J=1,3
AA(((I-1)+(J-1)*LDA)+1) = 1.0
12 CONTINUE
11 CONTINUE
DO 13 I= 1, 3
DO 14 J=1,2
CC( ((I-1) + (J-1)*LDC)+1 ) = 1.0
BB( ((I-1) + (J-1)*LDB)+1 ) = 2.0
14 CONTINUE
13 CONTINUE
CALL DSYMM('L', 'U', 3, 2, 0.5D0, AA, LDA, BB, LDB, 2.0D0, CC, LDC)
DO 15 I= 1, 3
DO 16 J=1,2
print *, ( (I-1)+(J-1)*LDC)+1,CC(((I-1)+(J-1)*LDC)+1)
16 CONTINUE
15 CONTINUE
CONTAINS
(see Appendix A.4)
end program testdblat3

```

Execute in Windows PowerShell

```
get-content dblat3_summry.txt | & ".\testdblat3.exe"
```

2.4.3.3 C#

```

namespace testcsblas3
{
    class dblat3
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();

```

```

cs_BLAS.misc_d.DLAMCH("E");
cs_BLAS.misc_s.SLAMCH("E");

cs_BLAS.misc_d.DBLAT3(); //requires misc_d.eps [DLAMCH("E")]
Console.WriteLine("DBLAT3 Done.");
//https://software.intel.com/en-us/node/522288
double[] aa = new Double[12];
double[] bb= new Double[8];
double[] cc = new Double[8];
double[,] tmp2d_cc;
int i,j,lda,ldb,ldc;
lda = 4;
ldb = 4;
ldc = 4;
for (i = 1; i <= 3; i = i + 1)
{
    for (j = 1; j <= 3; j = j + 1)
    {
        aa[((i - 1) + (j - 1) * lda) + 1 - 1] = 1.0d;
    }
}
for (i = 1; i <= 3; i = i + 1)
{
    for (j = 1; j <= 2; j = j + 1)
    {
        cc[((i - 1) + (j - 1) * ldc) + 1 - 1] = 1.0d;
        bb[((i - 1) + (j - 1) * ldb) + 1 - 1] = 2.0d;
    }
}
tmp2d_cc = cs_BLAS.misc_d.get_2dfrom1d(8, 0, ldc, cc);
//DSYMM(side, uplo, m, n, alpha, get_2dfrom1d(nmax * nmax, 0, lda, aa), lda, get_2dfrom1d(nmax * nmax, 0, ldb, bb),
ldb, beta, ref tmp2d_cc, ldc);
cs_BLAS.misc_d.DSYMM("L", "U", 3, 2, 0.5d, cs_BLAS.misc_d.get_2dfrom1d(12, 0, lda, aa), lda,
cs_BLAS.misc_d.get_2dfrom1d(8, 0, ldb, bb), ldb, 2.0d, ref tmp2d_cc, ldc);
cc = cs_BLAS.misc_d.get_ldfrom2d(8, 0, ldc, tmp2d_cc);
for (i = 1; i <= 3; i = i + 1)
{
    for (j = 1; j <= 2; j = j + 1)
    {
        Console.WriteLine("{0,20:F13} {1,20:F13}", ((i - 1) + (j - 1) * ldc) + 1, cc[((i - 1) + (j - 1) * ldc) + 1
- 1]);
    }
}
}
}
}

```

Execute in Windows PowerShell

```
get-content dblat3_summry.txt | & ".\testcsblas3.exe"
```

2.4.4 ZBLAT1

Test for double complex Level 1 BLAS.

2.4.4.1 FORTRAN 90

```

program testzblat1
implicit none
CALL ZBLAT1
print *, 'ZBLAT1 Done.'
CONTAINS
    (see Appendix A.5)
end program testzblat1

```

Execute in Windows PowerShell

```
.\testzblat1.exe
```

2.4.4.2 C#

```

namespace testcsblas1z
{
    class zblat1
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();

            cs_BLAS.misc_d.DLAMCH("E");
            cs_BLAS.misc_s.SLAMCH("E");

            cs_BLAS.misc_z.ZBLAT1();
            Console.WriteLine("ZBLAT1 Done.");
        }
    }
}

```

```

    }
}

```

Execute in Windows PowerShell

```
.\testcsblas1z.exe
```

2.4.5 ZBLAT2

Test for double complex Level 2 BLAS.

2.4.5.1 SUMMARY DATA

```

'zblat2.out'      NAME OF SUMMARY OUTPUT FILE
6                UNIT NUMBER OF SUMMARY FILE
'CBLA2T.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
-1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F                LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F                LOGICAL FLAG, T TO STOP ON FAILURES.
T                LOGICAL FLAG, T TO TEST ERROR EXITS.
16.0            THRESHOLD VALUE OF TEST RATIO
6                NUMBER OF VALUES OF N
0 1 2 3 5 9      VALUES OF N
4                NUMBER OF VALUES OF K
0 1 2 4          VALUES OF K
4                NUMBER OF VALUES OF INCX AND INCY
1 2 -1 -2        VALUES OF INCX AND INCY
3                NUMBER OF VALUES OF ALPHA
(0.0,0.0) (1.0,0.0) (0.7,-0.9)  VALUES OF ALPHA
3                NUMBER OF VALUES OF BETA
(0.0,0.0) (1.0,0.0) (1.3,-1.1)  VALUES OF BETA
ZGEMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZGBMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZHEMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZHBMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZHPMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTRMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTBMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTPMV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTRSV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTBSV  T PUT F FOR NO TEST. SAME COLUMNS.
ZTPSV  T PUT F FOR NO TEST. SAME COLUMNS.
ZGERC  T PUT F FOR NO TEST. SAME COLUMNS.
ZGERU  T PUT F FOR NO TEST. SAME COLUMNS.
ZHER   T PUT F FOR NO TEST. SAME COLUMNS.
ZHPR   T PUT F FOR NO TEST. SAME COLUMNS.
ZHER2  T PUT F FOR NO TEST. SAME COLUMNS.
ZHPR2  T PUT F FOR NO TEST. SAME COLUMNS.

```

2.4.5.2 FORTRAN 90

```

      program testzblat2
      implicit none
      CALL ZBLAT2
      print *, 'ZBLAT2 Done.'
CONTAINS
      (see Appendix A.6)
end program testzblat2

```

Execute in Windows PowerShell

```
get-content zblat2_summry.txt | & ".\testzblat2.exe"
```

2.4.5.3 C#

```

namespace testcsblas2z
{
    class zblat2
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();

            cs_BLAS.misc_d.DLAMCH("E");
            cs_BLAS.misc_s.SLAMCH("E");

            cs_BLAS.misc_z.ZBLAT2(); // requires misc_d.eps [DLAMCH("E")]
            Console.WriteLine("ZBLAT2 Done.");
        }
    }
}

```

Execute in Windows PowerShell

```
get-content zblat2_summry.txt | & ".\testcsblas2z.exe"
```

2.4.6 ZBLAT3

Test for double complex Level 3 BLAS.

2.4.6.1 SUMMARY DATA

```
'zblat3.out'      NAME OF SUMMARY OUTPUT FILE
6                UNIT NUMBER OF SUMMARY FILE
'ZBLAT3.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
-1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F               LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F               LOGICAL FLAG, T TO STOP ON FAILURES.
T               LOGICAL FLAG, T TO TEST ERROR EXITS.
16.0            THRESHOLD VALUE OF TEST RATIO
6               NUMBER OF VALUES OF N
0 1 2 3 5 9      VALUES OF N
3               NUMBER OF VALUES OF ALPHA
(0.0,0.0) (1.0,0.0) (0.7,-0.9)  VALUES OF ALPHA
3               NUMBER OF VALUES OF BETA
(0.0,0.0) (1.0,0.0) (1.3,-1.1)  VALUES OF BETA
ZGEMM T PUT F FOR NO TEST. SAME COLUMNS.
ZHEMM T PUT F FOR NO TEST. SAME COLUMNS.
ZSYMM T PUT F FOR NO TEST. SAME COLUMNS.
ZTRMM T PUT F FOR NO TEST. SAME COLUMNS.
ZTRSM T PUT F FOR NO TEST. SAME COLUMNS.
ZHERK T PUT F FOR NO TEST. SAME COLUMNS.
ZSYRK T PUT F FOR NO TEST. SAME COLUMNS.
ZHER2K T PUT F FOR NO TEST. SAME COLUMNS.
ZSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
```

2.4.6.2 FORTRAN 90

```
program testzblat2
implicit none
CALL ZBLAT3
print *, 'ZBLAT3 Done.'
CONTAINS
(see Appendix A.7)
end program testzblat3
```

Execute in Windows PowerShell

```
get-content zblat3_summry.txt | & ".\testzblat3.exe"
```

2.4.6.3 C#

```
namespace testcsblas2z
{
    class zblat2
    {
        static void Main(string[] args)
        {
            cs_BLAS.version.print();

            cs_BLAS.misc_d.DLAMCH("E");
            cs_BLAS.misc_s.SLAMCH("E");

            cs_BLAS.misc_z.ZBLAT3(); // requires misc_d.eps [DLAMCH("E")]
            Console.WriteLine("ZBLAT3 Done.");
        }
    }
}
```

Execute in Windows PowerShell

```
get-content zblat3_summry.txt | & ".\testcsblas3z.exe"
```

3.0 OUTPUT COMPARISON

Output for each BLAS test listed in section 2.2 is shown below. The standard output is denoted as “stdout”. For Level 2 and Level 3 tests some output is written to a file denoted as “*.out”. The “[insert]” text is used to indicate an additional line added to align the output for better comparison.

Additional output for DDOT (Level1), DGER (Level 2), and DSYMM (Level 3) can be compared to output from the Intel Math Kernel Library [7, <https://software.intel.com/en-us/node/522288>].

3.1 DBLAT1

stdout

FORTRAN 90		C#	
[insert]		LINALG 3.5.0	
Epsilon	= 1.110223024625157E-016	Epsilon	= 1.110223024625157E-016
Safe minimum	= 2.225073858507201E-308	Safe minimum	= 2.225073858507201E-308
Base	= 2.000000000000000	Base	= 2.000000000000000E+000
Precision	= 2.220446049250313E-016	Precision	= 2.220446049250313E-016
Number of digits in mantissa	= 53.00000000000000	Number of digits in mantissa	= 5.300000000000000E+001
Rounding mode	= 1.000000000000000	Rounding mode	= 1.000000000000000E+000
Minimum exponent	= -1021.000000000000	Minimum exponent	= -1.021000000000000E+003
Underflow threshold	= 2.225073858507201E-308	Underflow threshold	= 2.225073858507201E-308
Largest exponent	= 1024.000000000000	Largest exponent	= 1.024000000000000E+003
Overflow threshold	= 1.797693134862316E+308	Overflow threshold	= 1.797693134862316E+308
Reciprocal of safe minimum	= 4.494232837155790E+307	Reciprocal of safe minimum	= 4.494232837155790E+307
=====		=====	
Epsilon	= 5.9604645E-08	Epsilon	= 5.9604645E-008
Safe minimum	= 1.1754944E-38	Safe minimum	= 1.1754944E-038
Base	= 2.000000	Base	= 2.0000000E+000
Precision	= 1.1920929E-07	Precision	= 1.1920929E-007
Number of digits in mantissa	= 24.00000	Number of digits in mantissa	= 2.4000000E+001
Rounding mode	= 1.000000	Rounding mode	= 1.0000000E+000
Minimum exponent	= -125.0000	Minimum exponent	= -1.2500000E+002
Underflow threshold	= 1.1754944E-38	Underflow threshold	= 1.1754944E-038
Largest exponent	= 128.0000	Largest exponent	= 1.2800000E+002
Overflow threshold	= 3.4028235E+38	Overflow threshold	= 3.4028235E+038
Reciprocal of safe minimum	= 8.5070592E+37	Reciprocal of safe minimum	= 8.5070592E+037
=====		=====	
Time for 0.100E+09 DAXPY ops	= 0.207 seconds	Time for 1E+08 DAXPY ops	= 0.255 seconds
DAXPY performance rate	= 483. mflops	DAXPY performance rate	= 392 mflops
Including DSECND, time	= 0.201 seconds	Including DSECND, time	= 0.198 seconds
=====		=====	
Time for 0.100E+09 SAXPY ops	= 0.200 seconds	Time for 1E+08 SAXPY ops	= 0.182 seconds
SAXPY performance rate	= 500. mflops	SAXPY performance rate	= 549 mflops
Including SECOND, time	= 0.209 seconds	Including SECOND, time	= 0.189 seconds
Average time for SECOND	= 0.180E-03 milliseconds	Average time for SECOND	= 0.00014 milliseconds
Equivalent floating point ops	= 90.0 ops	Equivalent floating point ops	= 76.9 ops
=====		=====	
We are about to check whether infinity arithmetic can be trusted. If this test hangs, set ILAENV = 0 for ISPEC = 10 in LAPACK/SRC/ilaenv.f		We are about to check whether infinity arithmetic can be trusted. If this test hangs, set ILAENV = 0 for ISPEC = 10 in LAPACK/SRC/ilaenv.f	
Infinity arithmetic performed as per the ieee spec. However, this is not an exhaustive test and does not guarantee that infinity arithmetic meets the ieee spec.		Infinity arithmetic performed as per the ieee spec. However, this is not an exhaustive test and does not guarantee that infinity arithmetic meets the ieee spec.	
We are about to check whether NaN arithmetic can be trusted. If this test hangs, set ILAENV = 0 for ISPEC = 11 in LAPACK/SRC/ilaenv.f		We are about to check whether NaN arithmetic can be trusted. If this test hangs, set ILAENV = 0 for ISPEC = 11 in LAPACK/SRC/ilaenv.f	
NaN arithmetic did not perform per the ieee spec		NaN arithmetic performed as per the ieee spec. However, this is not an exhaustive test and does not guarantee that NaN arithmetic meets the ieee spec.	
[insert]		[insert]	
[insert]		[insert]	
TSTIEE Done.		TSTIEE Done.	
Real BLAS Test Program Results		Real BLAS Test Program Results	
Test of subprogram number 1	DDOT ----- PASS -----	Test of subprogram number 1	DDOT ----- PASS -----
Test of subprogram number 2	DAXPY ----- PASS -----	Test of subprogram number 2	DAXPY ----- PASS -----
Test of subprogram number 3	DROTG ----- PASS -----	Test of subprogram number 3	DROTG ----- PASS -----
Test of subprogram number 4	DROT ----- PASS -----	Test of subprogram number 4	DROT ----- PASS -----
Test of subprogram number 5	DCOPY ----- PASS -----	Test of subprogram number 5	DCOPY ----- PASS -----
Test of subprogram number 6	DSWAP ----- PASS -----	Test of subprogram number 6	DSWAP ----- PASS -----
Test of subprogram number 7	DNRM2 ----- PASS -----	Test of subprogram number 7	DNRM2 ----- PASS -----
Test of subprogram number 8	DASUM ----- PASS -----	Test of subprogram number 8	DASUM ----- PASS -----
Test of subprogram number 9	DSCAL ----- PASS -----	Test of subprogram number 9	DSCAL ----- PASS -----
Test of subprogram number 10	IDAMAX	Test of subprogram number 10	IDAMAX

----- PASS ----- Test of subprogram number 11 DROTMG ----- PASS ----- Test of subprogram number 12 DROTM ----- PASS ----- Test of subprogram number 13 DSDOT ----- PASS ----- DBLAT1 Done. 10.00000000000000	----- PASS ----- Test of subprogram number 11 DROTMG ----- PASS ----- Test of subprogram number 12 DROTM ----- PASS ----- Test of subprogram number 13 DSDOT ----- PASS ----- DBLAT1 Done. 10.00000000000000
--	--

3.2 DBLAT2

stdout

FORTRAN 90	C#
[insert]	LINALG 3.5.0
[insert]	Read:'dbl2.out' NAME OF SUMMARY OUTPUT FILE
[insert]	Read:6 UNIT NUMBER OF SUMMARY FILE
[insert]	Read:'DBLAT2.SNAP' NAME OF SNAPSHOT OUTPUT FILE
[insert]	Read:-1 UNIT NUMBER OF SNAPSHOT FILE (NOT USED
[insert]	IF .LT. 0)
[insert]	Read:F LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER
[insert]	EACH RECORD.
[insert]	Read:F LOGICAL FLAG, T TO STOP ON FAILURES.
[insert]	Read:T LOGICAL FLAG, T TO TEST ERROR EXITS.
[insert]	Read:16.0 THRESHOLD VALUE OF TEST RATIO
[insert]	Read:6 NUMBER OF VALUES OF N
[insert]	Read:0 1 2 3 5 9 VALUES OF N
[insert]	Read:4 NUMBER OF VALUES OF K
[insert]	Read:0 1 2 4 VALUES OF K
[insert]	Read:4 NUMBER OF VALUES OF INCX AND INCY
[insert]	Read:1 2 -1 -2 VALUES OF INCX AND INCY
[insert]	Read:3 NUMBER OF VALUES OF ALPHA
[insert]	Read:0.0 1.0 0.7 VALUES OF ALPHA
[insert]	Read:3 NUMBER OF VALUES OF BETA
[insert]	Read:0.0 1.0 0.9 VALUES OF BETA
[insert]	Read:DGEMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DGBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSPMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTRMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTPMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTRSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTBSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTPSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DGER T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYR T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSPR T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYR2 T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSPR2 T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	end.
DBLAT2 Done.	DBLAT2 Done.
1 1.500000000000000	1.000000000000000 1.500000000000000
4 2.500000000000000	4.000000000000000 2.500000000000000
7 3.500000000000000	7.000000000000000 3.500000000000000
2 1.500000000000000	2.000000000000000 1.500000000000000
5 2.500000000000000	5.000000000000000 2.500000000000000
8 3.500000000000000	8.000000000000000 3.500000000000000

dbl2.out

FORTRAN 90	C#
TESTS OF THE DOUBLE PRECISION LEVEL 2 BLAS	TESTS OF THE DOUBLE PRECISION LEVEL 2 BLAS
THE FOLLOWING PARAMETER VALUES WILL BE USED:	THE FOLLOWING PARAMETER VALUES WILL BE USED:
FOR N 0 1 2 3 5 9	FOR N 0 1 2 3 5 9
FOR K 0 1 2 4	FOR K 0 1 2 4
FOR INCX AND INCY 1 2 -1 -2	FOR INCX AND INCY 1 2 -1 -2
FOR ALPHA 0.0 1.0 0.7	FOR ALPHA 0.0 1.0 0.7
FOR BETA 0.0 1.0 0.9	FOR BETA 0.0 1.0 0.9
ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00	ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00
RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2D-16	RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2E-016
DGEMV PASSED THE TESTS OF ERROR-EXITS	DGEMV PASSED THE TESTS OF ERROR-EXITS
DGEMV PASSED THE COMPUTATIONAL TESTS (3460 CALLS)	DGEMV PASSED THE COMPUTATIONAL TESTS (3460 CALLS)
DGBMV PASSED THE TESTS OF ERROR-EXITS	DGBMV PASSED THE TESTS OF ERROR-EXITS
DGBMV PASSED THE COMPUTATIONAL TESTS (13828 CALLS)	DGBMV PASSED THE COMPUTATIONAL TESTS (13828 CALLS)
DSYMV PASSED THE TESTS OF ERROR-EXITS	DSYMV PASSED THE TESTS OF ERROR-EXITS
DSYMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)	DSYMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)

DSBMV PASSED THE TESTS OF ERROR-EXITS	DSBMV PASSED THE TESTS OF ERROR-EXITS
DSBMV PASSED THE COMPUTATIONAL TESTS (5761 CALLS)	DSBMV PASSED THE COMPUTATIONAL TESTS (5761 CALLS)
DSPMV PASSED THE TESTS OF ERROR-EXITS	DSPMV PASSED THE TESTS OF ERROR-EXITS
DSPMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)	DSPMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)
DTRMV PASSED THE TESTS OF ERROR-EXITS	DTRMV PASSED THE TESTS OF ERROR-EXITS
DTRMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	DTRMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
DTBMV PASSED THE TESTS OF ERROR-EXITS	DTBMV PASSED THE TESTS OF ERROR-EXITS
DTBMV PASSED THE COMPUTATIONAL TESTS (961 CALLS)	DTBMV PASSED THE COMPUTATIONAL TESTS (961 CALLS)
DTPMV PASSED THE TESTS OF ERROR-EXITS	DTPMV PASSED THE TESTS OF ERROR-EXITS
DTPMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	DTPMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
DTRSV PASSED THE TESTS OF ERROR-EXITS	DTRSV PASSED THE TESTS OF ERROR-EXITS
DTRSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	DTRSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
DTBSV PASSED THE TESTS OF ERROR-EXITS	DTBSV PASSED THE TESTS OF ERROR-EXITS
DTBSV PASSED THE COMPUTATIONAL TESTS (961 CALLS)	DTBSV PASSED THE COMPUTATIONAL TESTS (961 CALLS)
DTPSV PASSED THE TESTS OF ERROR-EXITS	DTPSV PASSED THE TESTS OF ERROR-EXITS
DTPSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	DTPSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
DGER PASSED THE TESTS OF ERROR-EXITS	DGER PASSED THE TESTS OF ERROR-EXITS
DGER PASSED THE COMPUTATIONAL TESTS (388 CALLS)	DGER PASSED THE COMPUTATIONAL TESTS (388 CALLS)
DSYR PASSED THE TESTS OF ERROR-EXITS	DSYR PASSED THE TESTS OF ERROR-EXITS
DSYR PASSED THE COMPUTATIONAL TESTS (121 CALLS)	DSYR PASSED THE COMPUTATIONAL TESTS (121 CALLS)
DSPR PASSED THE TESTS OF ERROR-EXITS	DSPR PASSED THE TESTS OF ERROR-EXITS
DSPR PASSED THE COMPUTATIONAL TESTS (121 CALLS)	DSPR PASSED THE COMPUTATIONAL TESTS (121 CALLS)
DSYR2 PASSED THE TESTS OF ERROR-EXITS	DSYR2 PASSED THE TESTS OF ERROR-EXITS
DSYR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)	DSYR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)
DSPR2 PASSED THE TESTS OF ERROR-EXITS	DSPR2 PASSED THE TESTS OF ERROR-EXITS
DSPR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)	DSPR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)
END OF TESTS	END OF TESTS

3.3 DBLAT3

stdout

FORTRAN 90	C#
[insert]	LINALG 3.5.0
[insert]	Read:'dbl3.out' NAME OF SUMMARY OUTPUT FILE
[insert]	Read:6 UNIT NUMBER OF SUMMARY FILE
[insert]	Read:'DBLAT3.SNAP' NAME OF SNAPSHOT OUTPUT FILE
[insert]	Read:-1 UNIT NUMBER OF SNAPSHOT FILE (NOT USED
[insert]	IF .LT. 0)
[insert]	Read:F LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER
[insert]	EACH RECORD.
[insert]	Read:F LOGICAL FLAG, T TO STOP ON FAILURES.
[insert]	Read:T LOGICAL FLAG, T TO TEST ERROR EXITS.
[insert]	Read:16.0 THRESHOLD VALUE OF TEST RATIO
[insert]	Read:6 NUMBER OF VALUES OF N
[insert]	Read:0 1 2 3 5 9 VALUES OF N
[insert]	Read:3 NUMBER OF VALUES OF ALPHA
[insert]	Read:0.0 1.0 0.7 VALUES OF ALPHA
[insert]	Read:3 NUMBER OF VALUES OF BETA
[insert]	Read:0.0 1.0 1.3 VALUES OF BETA
[insert]	Read:DGEMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTRMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DTRSM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYRK T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:DSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	end.
DBLAT3 Done.	DBLAT3 Done.
1 5.000000000000000	1.000000000000000 5.000000000000000
5 5.000000000000000	5.000000000000000 5.000000000000000
2 5.000000000000000	2.000000000000000 5.000000000000000
6 5.000000000000000	6.000000000000000 5.000000000000000
3 5.000000000000000	3.000000000000000 5.000000000000000
7 5.000000000000000	7.000000000000000 5.000000000000000

dblat3.out

FORTRAN 90	C#
TESTS OF THE DOUBLE PRECISION LEVEL 3 BLAS	TESTS OF THE DOUBLE PRECISION LEVEL 3 BLAS
THE FOLLOWING PARAMETER VALUES WILL BE USED: FOR N 0 1 2 3 5 9 FOR ALPHA 0.0 1.0 0.7 FOR BETA 0.0 1.0 1.3	THE FOLLOWING PARAMETER VALUES WILL BE USED: FOR N 0 1 2 3 5 9 FOR ALPHA 0.0 1.0 0.7 FOR BETA 0.0 1.0 1.3
ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00	ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00
RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2D-16	RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2E-016
DGEMM PASSED THE TESTS OF ERROR-EXITS	DGEMM PASSED THE TESTS OF ERROR-EXITS
DGEMM PASSED THE COMPUTATIONAL TESTS (17496 CALLS)	DGEMM PASSED THE COMPUTATIONAL TESTS (17496 CALLS)
DSYMM PASSED THE TESTS OF ERROR-EXITS	DSYMM PASSED THE TESTS OF ERROR-EXITS
DSYMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	DSYMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
DTRMM PASSED THE TESTS OF ERROR-EXITS	DTRMM PASSED THE TESTS OF ERROR-EXITS
DTRMM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)	DTRMM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)
DTRSM PASSED THE TESTS OF ERROR-EXITS	DTRSM PASSED THE TESTS OF ERROR-EXITS
DTRSM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)	DTRSM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)
DSYRK PASSED THE TESTS OF ERROR-EXITS	DSYRK PASSED THE TESTS OF ERROR-EXITS
DSYRK PASSED THE COMPUTATIONAL TESTS (1944 CALLS)	DSYRK PASSED THE COMPUTATIONAL TESTS (1944 CALLS)
DSYR2K PASSED THE TESTS OF ERROR-EXITS	DSYR2K PASSED THE TESTS OF ERROR-EXITS
DSYR2K PASSED THE COMPUTATIONAL TESTS (1944 CALLS)	DSYR2K PASSED THE COMPUTATIONAL TESTS (1944 CALLS)
END OF TESTS	END OF TESTS

3.4 ZBLAT1

stdout

FORTRAN 90	C#
[insert] Complex BLAS Test Program Results	LINALG 3.5.0 Complex BLAS Test Program Results
Test of subprogram number 1 ZDOTC ----- PASS -----	Test of subprogram number 1 ZDOTC ----- PASS -----
Test of subprogram number 2 ZDOTU ----- PASS -----	Test of subprogram number 2 ZDOTU ----- PASS -----
Test of subprogram number 3 ZAXPY ----- PASS -----	Test of subprogram number 3 ZAXPY ----- PASS -----
Test of subprogram number 4 ZCOPY ----- PASS -----	Test of subprogram number 4 ZCOPY ----- PASS -----
Test of subprogram number 5 ZSWAP ----- PASS -----	Test of subprogram number 5 ZSWAP ----- PASS -----
Test of subprogram number 6 DZNRM2 ----- PASS -----	Test of subprogram number 6 DZNRM2 ----- PASS -----
Test of subprogram number 7 DZASUM ----- PASS -----	Test of subprogram number 7 DZASUM ----- PASS -----
Test of subprogram number 8 ZSCAL ----- PASS -----	Test of subprogram number 8 ZSCAL ----- PASS -----
Test of subprogram number 9 ZDSCAL ----- PASS -----	Test of subprogram number 9 ZDSCAL ----- PASS -----
Test of subprogram number 10 IZAMAX ----- PASS -----	Test of subprogram number 10 IZAMAX ----- PASS -----
ZBLAT1 Done.	ZBLAT1 Done.

3.5 ZBLAT2

stdout

FORTRAN 90	C#
[insert] [insert] [insert] [insert]	LINALG 3.5.0 Read:'zblat2.out' NAME OF SUMMARY OUTPUT FILE Read:6 UNIT NUMBER OF SUMMARY FILE Read:'CBLA2T.SNAP' NAME OF SNAPSHOT OUTPUT FILE

[insert]	Read:-1 UNIT NUMBER OF SNAPSHOT FILE (NOT USED
[insert]	IF .LT. 0)
[insert]	Read:F LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER
[insert]	EACH RECORD.
[insert]	Read:F LOGICAL FLAG, T TO STOP ON FAILURES.
[insert]	Read:T LOGICAL FLAG, T TO TEST ERROR EXITS.
[insert]	Read:16.0 THRESHOLD VALUE OF TEST RATIO
[insert]	Read:6 NUMBER OF VALUES OF N
[insert]	Read:0 1 2 3 5 9 VALUES OF N
[insert]	Read:4 NUMBER OF VALUES OF K
[insert]	Read:0 1 2 4 VALUES OF K
[insert]	Read:4 NUMBER OF VALUES OF INCX AND INCY
[insert]	Read:1 2 -1 -2 VALUES OF INCX AND INCY
[insert]	Read:3 NUMBER OF VALUES OF ALPHA
[insert]	Read:(0.0,0.0) (1.0,0.0) (0.7,-0.9) VALUES OF ALPHA
[insert]	Read:3 NUMBER OF VALUES OF BETA
[insert]	Read:(0.0,0.0) (1.0,0.0) (1.3,-1.1) VALUES OF BETA
[insert]	Read:ZGEMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZGBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHEMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHPMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTRMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTBMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTPMV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTRSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTBSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTPSV T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZGERC T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZGERU T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHER T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHPR T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHER2 T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHPR2 T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	end.
ZBLAT2 Done.	ZBLAT2 Done.

zblat2.out

FORTRAN 90	C#
TESTS OF THE COMPLEX*16 LEVEL 2 BLAS	TESTS OF THE COMPLEX LEVEL 2 BLAS
THE FOLLOWING PARAMETER VALUES WILL BE USED:	THE FOLLOWING PARAMETER VALUES WILL BE USED:
FOR N 0 1 2 3 5 9	FOR N 0 1 2 3 5 9
FOR K 0 1 2 4	FOR K 0 1 2 4
FOR INCX AND INCY 1 2 -1 -2	FOR INCX AND INCY 1 2 -1 -2
FOR ALPHA (0.0, 0.0) (1.0, 0.0) (0.7,-0.9)	FOR ALPHA (0.0, 0.0) (1.0, 0.0) (0.7, -0.9)
FOR BETA (0.0, 0.0) (1.0, 0.0) (1.3,-1.1)	FOR BETA (0.0, 0.0) (1.0, 0.0) (1.3, -1.1)
ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00	ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00
RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2D-16	RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2E-016
ZGEMV PASSED THE TESTS OF ERROR-EXITS	ZGEMV PASSED THE TESTS OF ERROR-EXITS
ZGEMV PASSED THE COMPUTATIONAL TESTS (3460 CALLS)	ZGEMV PASSED THE COMPUTATIONAL TESTS (3460 CALLS)
ZGBMV PASSED THE TESTS OF ERROR-EXITS	ZGBMV PASSED THE TESTS OF ERROR-EXITS
ZGBMV PASSED THE COMPUTATIONAL TESTS (13828 CALLS)	ZGBMV PASSED THE COMPUTATIONAL TESTS (13828 CALLS)
ZHEMV PASSED THE TESTS OF ERROR-EXITS	ZHEMV PASSED THE TESTS OF ERROR-EXITS
ZHEMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)	ZHEMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)
ZHBMV PASSED THE TESTS OF ERROR-EXITS	ZHBMV PASSED THE TESTS OF ERROR-EXITS
ZHBMV PASSED THE COMPUTATIONAL TESTS (5761 CALLS)	ZHBMV PASSED THE COMPUTATIONAL TESTS (5761 CALLS)
ZHPMV PASSED THE TESTS OF ERROR-EXITS	ZHPMV PASSED THE TESTS OF ERROR-EXITS
ZHPMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)	ZHPMV PASSED THE COMPUTATIONAL TESTS (1441 CALLS)
ZTRMV PASSED THE TESTS OF ERROR-EXITS	ZTRMV PASSED THE TESTS OF ERROR-EXITS
ZTRMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	ZTRMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
ZTBMV PASSED THE TESTS OF ERROR-EXITS	ZTBMV PASSED THE TESTS OF ERROR-EXITS
ZTBMV PASSED THE COMPUTATIONAL TESTS (961 CALLS)	ZTBMV PASSED THE COMPUTATIONAL TESTS (961 CALLS)
ZTPMV PASSED THE TESTS OF ERROR-EXITS	ZTPMV PASSED THE TESTS OF ERROR-EXITS
ZTPMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	ZTPMV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
ZTRSV PASSED THE TESTS OF ERROR-EXITS	ZTRSV PASSED THE TESTS OF ERROR-EXITS
ZTRSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	ZTRSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
ZTBSV PASSED THE TESTS OF ERROR-EXITS	ZTBSV PASSED THE TESTS OF ERROR-EXITS
ZTBSV PASSED THE COMPUTATIONAL TESTS (961 CALLS)	ZTBSV PASSED THE COMPUTATIONAL TESTS (961 CALLS)

ZTPSV PASSED THE TESTS OF ERROR-EXITS	ZTPSV PASSED THE TESTS OF ERROR-EXITS
ZTPSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)	ZTPSV PASSED THE COMPUTATIONAL TESTS (241 CALLS)
ZGERC PASSED THE TESTS OF ERROR-EXITS	ZGERC PASSED THE TESTS OF ERROR-EXITS
ZGERC PASSED THE COMPUTATIONAL TESTS (388 CALLS)	ZGERC PASSED THE COMPUTATIONAL TESTS (388 CALLS)
ZGERU PASSED THE TESTS OF ERROR-EXITS	ZGERU PASSED THE TESTS OF ERROR-EXITS
ZGERU PASSED THE COMPUTATIONAL TESTS (388 CALLS)	ZGERU PASSED THE COMPUTATIONAL TESTS (388 CALLS)
ZHER PASSED THE TESTS OF ERROR-EXITS	ZHER PASSED THE TESTS OF ERROR-EXITS
ZHER PASSED THE COMPUTATIONAL TESTS (121 CALLS)	ZHER PASSED THE COMPUTATIONAL TESTS (121 CALLS)
ZHPR PASSED THE TESTS OF ERROR-EXITS	ZHPR PASSED THE TESTS OF ERROR-EXITS
ZHPR PASSED THE COMPUTATIONAL TESTS (121 CALLS)	ZHPR PASSED THE COMPUTATIONAL TESTS (121 CALLS)
ZHER2 PASSED THE TESTS OF ERROR-EXITS	ZHER2 PASSED THE TESTS OF ERROR-EXITS
ZHER2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)	ZHER2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)
ZHPR2 PASSED THE TESTS OF ERROR-EXITS	ZHPR2 PASSED THE TESTS OF ERROR-EXITS
ZHPR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)	ZHPR2 PASSED THE COMPUTATIONAL TESTS (481 CALLS)
END OF TESTS	END OF TESTS

3.6 ZBLAT3

stdout

FORTRAN 90	C#
[insert]	LINALG 3.5.0
[insert]	Read:'zblat3.out' NAME OF SUMMARY OUTPUT FILE
[insert]	Read:6 UNIT NUMBER OF SUMMARY FILE
[insert]	Read:'ZBLAT3.SNAP' NAME OF SNAPSHOT OUTPUT FILE
[insert]	Read:-1 UNIT NUMBER OF SNAPSHOT FILE (NOT USED
[insert]	IF .LT. 0)
[insert]	Read:F LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER
[insert]	EACH RECORD.
[insert]	Read:F LOGICAL FLAG, T TO STOP ON FAILURES.
[insert]	Read:T LOGICAL FLAG, T TO TEST ERROR EXITS.
[insert]	Read:16.0 THRESHOLD VALUE OF TEST RATIO
[insert]	Read:6 NUMBER OF VALUES OF N
[insert]	Read:0 1 2 3 5 9 VALUES OF N
[insert]	Read:3 NUMBER OF VALUES OF ALPHA
[insert]	Read:(0.0,0.0) (1.0,0.0) (0.7,-0.9) VALUES OF ALPHA
[insert]	Read:3 NUMBER OF VALUES OF BETA
[insert]	Read:(0.0,0.0) (1.0,0.0) (1.3,-1.1) VALUES OF BETA
[insert]	Read:ZGEMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHEMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZSYMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTRMM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZTRSM T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHERK T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZSYRK T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZHER2K T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	Read:ZSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
[insert]	end.
ZBLAT3 Done.	ZBLAT3 Done.

zblat3.out

FORTRAN 90	C#
TESTS OF THE COMPLEX*16 LEVEL 3 BLAS	TESTS OF THE COMPLEX LEVEL 3 BLAS
THE FOLLOWING PARAMETER VALUES WILL BE USED:	THE FOLLOWING PARAMETER VALUES WILL BE USED:
FOR N 0 1 2 3 5 9	FOR N 0 1 2 3 5 9
FOR ALPHA (0.0, 0.0) (1.0, 0.0) (0.7,-0.9)	FOR ALPHA (0.0, 0.0) (1.0, 0.0) (0.7, -0.9)
FOR BETA (0.0, 0.0) (1.0, 0.0) (1.3,-1.1)	FOR BETA (0.0, 0.0) (1.0, 0.0) (1.3, -1.1)
[insert]	[insert]
ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00	ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN 16.00
RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2D-16	RELATIVE MACHINE PRECISION IS TAKEN TO BE 2.2E-016
ZGEMM PASSED THE TESTS OF ERROR-EXITS	ZGEMM PASSED THE TESTS OF ERROR-EXITS
ZGEMM PASSED THE COMPUTATIONAL TESTS (17496 CALLS)	ZGEMM PASSED THE COMPUTATIONAL TESTS (17496 CALLS)
ZHEMM PASSED THE TESTS OF ERROR-EXITS	ZHEMM PASSED THE TESTS OF ERROR-EXITS
ZHEMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZHEMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
ZSYMM PASSED THE TESTS OF ERROR-EXITS	ZSYMM PASSED THE TESTS OF ERROR-EXITS
ZSYMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZSYMM PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
ZTRMM PASSED THE TESTS OF ERROR-EXITS	ZTRMM PASSED THE TESTS OF ERROR-EXITS

ZTRMM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)	ZTRMM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)
ZTRSM PASSED THE TESTS OF ERROR-EXITS	ZTRSM PASSED THE TESTS OF ERROR-EXITS
ZTRSM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)	ZTRSM PASSED THE COMPUTATIONAL TESTS (2592 CALLS)
ZHERK PASSED THE TESTS OF ERROR-EXITS	ZHERK PASSED THE TESTS OF ERROR-EXITS
ZHERK PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZHERK PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
ZSYRK PASSED THE TESTS OF ERROR-EXITS	ZSYRK PASSED THE TESTS OF ERROR-EXITS
ZSYRK PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZSYRK PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
ZHER2K PASSED THE TESTS OF ERROR-EXITS	ZHER2K PASSED THE TESTS OF ERROR-EXITS
ZHER2K PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZHER2K PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
ZSYR2K PASSED THE TESTS OF ERROR-EXITS	ZSYR2K PASSED THE TESTS OF ERROR-EXITS
ZSYR2K PASSED THE COMPUTATIONAL TESTS (1296 CALLS)	ZSYR2K PASSED THE COMPUTATIONAL TESTS (1296 CALLS)
END OF TESTS	END OF TESTS

4.0 LICENSING

BLAS Licensing:

The reference BLAS is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web. Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.

Like all software, it is copyrighted. It is not trademarked, but we do ask the following:

- If you modify the source for these routines we ask that you change the name of the routine and comment the changes made to the original.
- We will gladly answer any questions regarding the software. If a modification is done, however, it is the responsibility of the person who modified the routine to provide support.

LAPACK License:

Copyright (c) 1992-2013 The University of Tennessee and The University
of Tennessee Research Foundation. All rights reserved.
Copyright (c) 2000-2013 The University of California Berkeley. All rights reserved.
Copyright (c) 2006-2013 The University of Colorado Denver. All rights reserved.

\$COPYRIGHT\$

Additional copyrights may follow

\$HEADER\$

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5.0 CONCLUSION

The proof of principle study undertaken to rewrite BLAS in the C# language was completed. Comparison of machine floating point precision, error-exit tests, and computational tests between the FORTRAN 90 and C# implementations for Level 1, 2, and 3 BLAS subroutines were completed successfully. Additional comparison tests included DDOT (Level1), DGER (Level 2), and DSYMM (Level 3) that matched the output from the Intel Math Kernel Library BLAS Code. No optimization was attempted.

This working C# language implementation of BLAS defines a starting point to further optimize BLAS in an object-oriented class based .NET language and explore expansion into the areas of other operating systems (.NET Core is designed to work on Windows, Linux, and macOS operating systems) and cloud computing environments.

REFERENCE

- [1] BLAS (Basic Linear Algebra Subprograms) (<http://www.netlib.org/blas/>)
- [1a] BLAS (https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms)
- [2] LAPACK (Linear Algebra PACKage) (<http://www.netlib.org/lapack/>) (LAPACK 3.5.0 used)
- [3] C# Language (<https://docs.microsoft.com/en-us/dotnet/csharp/>)
- [4] Least Squares (<http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html>)
- [5] DotNumerics (<http://www.dotnumerics.com/>)
- [6] Windows PowerShell (<https://docs.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell?view=powershell-5.1>)
- [7] Intel Math Kernel Library (MKL) BLAS Code Examples (<https://software.intel.com/en-us/mkl-developer-reference-c-blas-and-sparse-blas-routines>) and (<https://software.intel.com/en-us/node/522288>)

APPENDIX A.1 – cs_BLAS.cs

```
using System;
using System.IO; //StreamReader/Writer
using System.Collections; //ArrayList
using System.Collections.Generic;
//using System.Linq;
using System.Text;
using System.Numerics; // Complex

namespace cs_BLAS
{
    public static class version
    {
        {
            static int vers_major = 3;
            static int vers_minor = 5;
            static int vers_patch = 0;

            public static void print()
            {
                Console.WriteLine("LINALG {0}.{1}.{2}", vers_major, vers_minor, vers_patch);
            }
            public static void TSTIEE()
            {
                //
                // C# LAPACK auxiliary routine (version 3.4.0) of LAPACK TSTIEE
                //
                // Local Scalars
                int ieeek;
                Console.WriteLine("We are about to check whether infinity arithmetic");
                Console.WriteLine("can be trusted. If this test hangs, set");
                Console.WriteLine("ILAENV = 0 for ISPEC = 10 in LAPACK/SRC/ilaenv.f");

                ieeek = ILAENV(10, "ILAENV", "N", 1, 2, 3, 4);
                Console.WriteLine("");

                if (ieeek == 0)
                {
                    Console.WriteLine("Infinity arithmetic did not perform per the ieee spec");
                }
                else
                {
                    Console.WriteLine("Infinity arithmetic performed as per the ieee spec.");
                    Console.WriteLine("However, this is not an exhaustive test and does not");
                    Console.WriteLine("guarantee that infinity arithmetic meets the ieee spec.");
                }

                Console.WriteLine("");
                Console.WriteLine("We are about to check whether NaN arithmetic");
                Console.WriteLine("can be trusted. If this test hangs, set");
                Console.WriteLine("ILAENV = 0 for ISPEC = 11 in LAPACK/SRC/ilaenv.f");

                ieeek = ILAENV(11, "ILAENV", "N", 1, 2, 3, 4);

                Console.WriteLine("");

                if (ieeek == 0)
                {
                    Console.WriteLine("NaN arithmetic did not perform per the ieee spec");
                }
                else
                {
                    Console.WriteLine("NaN arithmetic performed as per the ieee spec.");
                    Console.WriteLine("However, this is not an exhaustive test and does not");
                    Console.WriteLine("guarantee that NaN arithmetic meets the ieee spec.");
                }
                Console.WriteLine("");
            }
        }
        public static int ILAENV(int ispec, string name, string opts, int n1, int n2, int n3, int n4)
        {
            // C# LAPACK auxiliary routine (version 3.4.0) of LAPACK ILAENV
            //
            // ILAENV is called from the LAPACK routines to choose problem-dependent parameters for the local environment. See ISPEC for a description of
            the parameters.
            //
            // This version provides a set of parameters which should give good, but not optimal, performance on many of the currently available computers.
            Users are encouraged
            // to modify this subroutine to set the tuning parameters for their particular machine using the option and problem size information in the
            arguments.
            //
            // This routine will not function correctly if it is converted to all lower case. Converting it to all upper case is allowed.
            /*
            * Arguments:
            * =====
            *
            * ISPEC (input) INTEGER
            * Specifies the parameter to be returned as the value of ILAENV.
            * = 1: the optimal blocksize; if this value is 1, an unblocked algorithm will give the best performance.
            * = 2: the minimum block size for which the block routine should be used; if the usable block size is less than this value, an
            unblocked routine should be used.
            * = 3: the crossover point (in a block routine, for N less than this value, an unblocked routine should be used)
            * = 4: the number of shifts, used in the nonsymmetric eigenvalue routines
            * = 5: the minimum column dimension for blocking to be used; rectangular blocks must have dimension at least k by m, where k is given
            by ILAENV(2,...) and m by ILAENV(5,...)
            * = 6: the crossover point for the SVD (when reducing an m by n matrix to bidiagonal form, if max(m,n)/min(m,n) exceeds this value, a
            QR factorization is used first to reduce
            * the matrix to a triangular form.)
            * = 7: the number of processors
            * = 8: the crossover point for the multishift QR and QZ methods for nonsymmetric eigenvalue problems.
            * = 9: maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm (used by xGELSD and
            xGESDD)
            * =10: ieee NaN arithmetic can be trusted not to trap
            * =11: infinity arithmetic can be trusted not to trap
            *
            * NAME (input) CHARACTER*(*)
            * The name of the calling subroutine, in either upper case or lower case.
            *
            * OPTS (input) CHARACTER*(*)
            * The character options to the subroutine NAME, concatenated into a single character string. For example, UPLO = 'U', TRANS = 'T',
            and DIAG = 'N' for a triangular routine would
            * be specified as OPTS = 'UTN'.
            *
            * N1 (input) INTEGER
            */
        }
    }
}
```

```

* N2      (input) INTEGER
* N3      (input) INTEGER
* N4      (input) INTEGER
*      Problem dimensions for the subroutine NAME; these may not all be required.
*
* (ILAENV) (output) INTEGER
*      >= 0: the value of the parameter specified by ISPEC
*      < 0: if ILAENV = -k, the k-th argument had an illegal value.
*
* Further Details
* =====
*
* The following conventions have been used when calling ILAENV from the LAPACK routines:
* 1) OPTS is a concatenation of all of the character options to subroutine NAME, in the same order that they appear in the argument list for
NAME, even if they are not used in determining
* the value of the parameter specified by ISPEC.
* 2) The problem dimensions N1, N2, N3, N4 are specified in the order that they appear in the argument list for NAME. N1 is used first, N2
second, and so on, and unused problem dimensions are
* passed a value of -1.
* 3) The parameter value returned by ILAENV is checked for validity in the calling subroutine. For example, ILAENV is used to retrieve the
optimal blocksize for STRTRI as follows:
*
*      NB = ILAENV( 1, 'STRTRI', UPLO // DIAG, N, -1, -1, -1 )
*      IF ( NB.LE.1 ) NB = MAX( 1, N )
*
* =====
*/
// Local Scalars
bool cname, sname;
string c1, c2, c3, c4, subnam;
/*
CHARACTER*1      C1
CHARACTER*2      C2, C4
CHARACTER*3      C3
CHARACTER*6      SUBNAM*/
int i, ic, iz, nb, nbmin, nx;
int ilaenv;

// Intrinsic Functions
//INTRINSIC      CHAR, ICHAR, INT, MIN, REAL

name = name.ToUpper();
switch (ispec)
{
    case 1:
    case 2:
    case 3:
        // Convert NAME to upper case if the first character is lower case.

        //ILAENV = 1
        /*
        SUBNAM = NAME
        IC = ICHAR( SUBNAM( 1:1 ) )
        IZ = ICHAR( 'Z' )
        IF( IZ.EQ.90 .OR. IZ.EQ.122 ) THEN

!
!      ASCII character set
!
        IF( IC.GE.97 .AND. IC.LE.122 ) THEN
            SUBNAM( 1:1 ) = CHAR( IC-32 )
            DO 10 I = 2, 6
                IC = ICHAR( SUBNAM( I:I ) )
                IF( IC.GE.97 .AND. IC.LE.122 )          SUBNAM( I:I ) = CHAR( IC-32 )
10            CONTINUE
            END IF

!
!      EBCDIC character set
!
        IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR. &
            ( IC.GE.145 .AND. IC.LE.153 ) .OR. &
            ( IC.GE.162 .AND. IC.LE.169 ) ) THEN
            SUBNAM( 1:1 ) = CHAR( IC+64 )
            DO 20 I = 2, 6
                IC = ICHAR( SUBNAM( I:I ) )
                IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR. &
                    ( IC.GE.145 .AND. IC.LE.153 ) .OR. &
                    ( IC.GE.162 .AND. IC.LE.169 ) ) &
                    SUBNAM( I:I ) = CHAR( IC+64 )
20            CONTINUE
            END IF

!
!      Prime machines: ASCII+128
!
        IF( IC.GE.225 .AND. IC.LE.250 ) THEN
            SUBNAM( 1:1 ) = CHAR( IC-32 )
            DO 30 I = 2, 6
                IC = ICHAR( SUBNAM( I:I ) )
                IF( IC.GE.225 .AND. IC.LE.250 ) &
                    SUBNAM( I:I ) = CHAR( IC-32 )
30            CONTINUE
            END IF
        END IF

!*/
        //C1 = SUBNAM( 1:1 )
        c1 = name.Substring(0, 1);
        sname = (c1 == "S") | (c1 == "D");
        cname = (c1 == "C") | (c1 == "Z");

        if (!(cname | sname))
        {
            return 1;
        }

        //C2 = SUBNAM( 2:3 )
        c2 = name.Substring(1, 2);
        //C3 = SUBNAM( 4:6 )
        c3 = name.Substring(3, 3);
        //C4 = C3( 2:3 )
        c4 = name.Substring(1, 2);

```

```

switch (ispec)
{
    case 1:
        //
        // ISPEC = 1: block size
        // In these examples, separate code is provided for setting NB for real and complex. We assume that NB will take the same
        value in single or double precision.
        //
        nb = 1;
        if (c2 == "GE")
        {
            if (c3 == "TRF")
            {
                if (sname)
                {
                    nb = 64;
                }
                else
                {
                    nb = 64;
                }
            }
            //ELSE IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR. &
            // C3.EQ.'QLF' ) THEN
        }
        else if ((c3 == "QRF") | (c3 == "RQF") | (c3 == "LQF") | (c3 == "QLF"))
        {
            if (sname)
            {
                nb = 32;
            }
            else
            {
                nb = 32;
            }
        }
        else if (c3 == "HRD")
        {
            if (sname)
            {
                nb = 32;
            }
            else
            {
                nb = 32;
            }
        }
        else if (c3 == "BRD")
        {
            if (sname)
            {
                nb = 32;
            }
            else
            {
                nb = 32;
            }
        }
        else if (c3 == "TRI")
        {
            if (sname)
            {
                nb = 64;
            }
            else
            {
                nb = 64;
            }
        }
    }
    else if (c2 == "FO")
    {
        if (c3 == "TRF")
        {
            if (sname)
            {
                nb = 64;
            }
            else
            {
                nb = 64;
            }
        }
    }
    else if (c2 == "SY")
    {
        if (c3 == "TRF")
        {
            if (sname)
            {
                nb = 64;
            }
            else
            {
                nb = 64;
            }
        }
        else if (sname & (c3 == "TRD"))
        {
            nb = 32;
        }
        else if (sname & (c3 == "GST"))
        {
            nb = 64;
        }
    }
    else if (cname & (c2 == "HE"))
    {
        if (c3 == "TRF")
        {
            nb = 64;
        }
        else if (c3 == "TRD")

```

```

        {
            nb = 32;
        }
        else if (c3 == "GST")
        {
            nb = 64;
        }
    }
    else if (sname & (c2 == "OR"))
    {
        if (c3.Substring(0, 1) == "G")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nb = 32;
            }
        }
        else if (c3.Substring(0, 1) == "M")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nb = 32;
            }
        }
    }
    else if (cname & (c2 == "UN"))
    {
        if (c3.Substring(0, 1) == "G")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nb = 32;
            }
        }
        else if (c3.Substring(0, 1) == "M")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nb = 32;
            }
        }
    }
    else if (c2 == "GB")
    {
        if (c3 == "TRF")
        {
            if (sname)
            {
                {
                    if (n4 <= 64)
                    {
                        nb = 1;
                    }
                    else
                    {
                        nb = 32;
                    }
                }
            }
            else
            {
                if (n4 <= 64)
                {
                    nb = 1;
                }
                else
                {
                    nb = 32;
                }
            }
        }
    }
    else if (c2 == "PB")
    {
        if (c3 == "TRF")
        {
            if (sname)
            {
                {
                    if (n2 <= 64)
                    {
                        nb = 1;
                    }
                    else
                    {
                        nb = 32;
                    }
                }
            }
            else
            {
                if (n2 <= 64)
                {
                    nb = 1;
                }
                else
                {
                    nb = 32;
                }
            }
        }
    }
    else if (c2 == "TR")
    {
        if (c3 == "TRI")
        {
            if (sname)
            {
                nb = 64;
            }
            else
            {
                nb = 64;
            }
        }
    }
    else if (c2 == "LA")
    {

```

```

        if (c3 == "UUM")
        {
            if (sname)
            {
                nb = 64;
            }
            else
            {
                nb = 64;
            }
        }
    }
    else if (sname & (c2 == "ST"))
    {
        if (c3 == "EBZ")
        {
            nb = 1;
        }
    }
    return nb;
break;
case 2:
//
// ISPEC = 2: minimum block size
//
nbmin = 2;
if (c2 == "GE")
{
    if ((c3 == "QRF") | (c3 == "RQF") | (c3 == "LQF") | (c3 == "QLF"))
    {
        if (sname)
        {
            nbmin = 2;
        }
        else
        {
            nbmin = 2;
        }
    }
    else if (c3 == "HRD")
    {
        if (sname)
        {
            nbmin = 2;
        }
        else
        {
            nbmin = 2;
        }
    }
    else if (c3 == "BRD")
    {
        if (sname)
        {
            nbmin = 2;
        }
        else
        {
            nbmin = 2;
        }
    }
    else if (c3 == "TRI")
    {
        if (sname)
        {
            nbmin = 2;
        }
        else
        {
            nbmin = 2;
        }
    }
}
else if (c2 == "SY")
{
    if (c3 == "TRF")
    {
        if (sname)
        {
            nbmin = 8;
        }
        else
        {
            nbmin = 8;
        }
    }
    else if (sname & (c3 == "TRD"))
    {
        nbmin = 2;
    }
}
else if (cname & (c2 == "HE"))
{
    if (c3 == "TRD")
    {
        nbmin = 2;
    }
}
else if (sname & (c2 == "OR"))
{
    if (c3.Substring(0, 1) == "G")
    {
        if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
        {
            nbmin = 2;
        }
    }
    else if (c3.Substring(0, 1) == "M")
    {
        if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
        {
            nbmin = 2;
        }
    }
}
}

```

```

    }
    else if (cname & (c2 == "UN"))
    {
        if (c3.Substring(0, 1) == "G")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nbmin = 2;
            }
        }
        else if (c3.Substring(0, 1) == "M")
        {
            if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
            {
                nbmin = 2;
            }
        }
    }
    return nbmin;
break;
case 3:
//
// ISPEC = 3: crossover point
//
nx = 0;
if (c2 == "GE")
{
    if ((c3 == "QRF") | (c3 == "RQF") | (c3 == "LQF") | (c3 == "QLF"))
    {
        if (sname)
        {
            nx = 128;
        }
        else
        {
            nx = 128;
        }
    }
    else if (c3 == "HRD")
    {
        if (sname)
        {
            nx = 128;
        }
        else
        {
            nx = 128;
        }
    }
    else if (c3 == "BRD")
    {
        if (sname)
        {
            nx = 128;
        }
        else
        {
            nx = 128;
        }
    }
}
else if (c2 == "SY")
{
    if (sname & (c3 == "TRD"))
    {
        nx = 32;
    }
}
else if (cname & (c2 == "HE"))
{
    if (c3 == "TRD")
    {
        nx = 32;
    }
}
else if (sname & (c2 == "OR"))
{
    if (c3.Substring(0, 1) == "G")
    {
        if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
        {
            nx = 128;
        }
    }
}
else if (cname & (c2 == "UN"))
{
    if (c3.Substring(0, 1) == "G")
    {
        if ((c4 == "QR") | (c4 == "RQ") | (c4 == "LQ") | (c4 == "QL") | (c4 == "HR") | (c4 == "TR") | (c4 == "BR"))
        {
            nx = 128;
        }
    }
}
return nx;
}
break;
case 4:
//
// ISPEC = 4: number of shifts (used by xHSEQR)
//
return 6;
break;
case 5:
//
// ISPEC = 5: minimum column dimension (not used)
//
return 2;
break;
case 6:
//
// ISPEC = 6: crossover point for SVD (used by xGELSS and xGESVD)
//

```

```

        return (int)((float)(Math.Min(n1, n2)) * 1.6E0F);
        break;
    case 7:
        //
        // ISPEC = 7: number of processors (not used)
        //
        return 1;
        break;
    case 8:
        //
        // ISPEC = 8: crossover point for multishift (used by xHSEQR)
        //
        return 50;
        break;
    case 9:
        //
        // ISPEC = 9: maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm (used by
xGELSD and xGESDD)
        //
        return 25;
        break;
    case 10:
        //
        // ISPEC = 10: ieee NaN arithmetic can be trusted not to trap
        //
        ilaenv = 1;
        if (ilaenv == 1)
        {
            return IEECK(0, 0.0F, 1.0F);
        }
        break;
    case 11:
        //
        // ISPEC = 11: infinity arithmetic can be trusted not to trap
        //
        ilaenv = 1;
        if (ilaenv == 1)
        {
            return IEECK(1, 0.0F, 1.0F);
        }
        break;
    default:
        // Invalid value for ISPEC
        return -1;
}
return -1; // invalid value for ISPEC
}
public static int IEECK(int ispec, float zero, float one)
{
    // C# LAPACK auxiliary routine (version 3.4.0) of LAPACK IEECK
    //
    // IEECK is called from the ILAENV to verify that Inifinity and possibly NaN arithmetic is safe (i.e. will not trap).
    /*
    * Arguments:
    * =====
    *
    * ISPEC      (input) INTEGER
    *             Specifies whether to test just for inifinity arithmetic
    *             or whether to test for infinity and NaN arithmetic.
    *             = 0: Verify infinity arithmetic only.
    *             = 1: Verify infinity and NaN arithmetic.
    *
    * ZERO       (input) REAL
    *             Must contain the value 0.0
    *             This is passed to prevent the compiler from optimizing
    *             away this code.
    *
    * ONE        (input) REAL
    *             Must contain the value 1.0
    *             This is passed to prevent the compiler from optimizing
    *             away this code.
    *
    * RETURN VALUE:  INTEGER
    *             = 0: Arithmetic failed to produce the correct answers
    *             = 1: Arithmetic produced the correct answers
    */

    // Local Scalars
    int ieeck = 1;
    float posinf, neginf, nan1, nan2, nan3, nan4, nan5, nan6, negzro, newzro;

    posinf = one / zero;

    if (posinf <= one)
    {
        return 0;
    }

    neginf = -one / zero;

    if (neginf >= zero)
    {
        return 0;
    }

    negzro = one / (neginf + one);

    if (negzro != zero)
    {
        return 0;
    }

    neginf = one / negzro;

    if (neginf >= zero)
    {
        return 0;
    }

    newzro = negzro + zero;

    if (newzro != zero)
    {
        return 0;
    }

```

```

    }

    posinf = one / newzro;

    if (posinf <= one)
    {
        return 0;
    }

    neginf = neginf * posinf;

    if (neginf >= zero)
    {
        return 0;
    }

    posinf = posinf * posinf;

    if (posinf <= one)
    {
        return 0;
    }

    // Return if we were only asked to check infinity arithmetic
    if (ispec == 0) return 1;

    nan1 = posinf + neginf;
    nan2 = posinf / neginf;
    nan3 = posinf / posinf;
    nan4 = posinf * zero;
    nan5 = neginf * negzro;
    nan6 = nan5 * 0.0F;

    if (nan1 == nan1)
    {
        return 0;
    }

    if (nan2 == nan2)
    {
        return 0;
    }

    if (nan3 == nan3)
    {
        return 0;
    }

    if (nan4 == nan4)
    {
        return 0;
    }

    if (nan5 == nan5)
    {
        return 0;
    }

    if (nan6 == nan6)
    {
        return 0;
    }

    return 1;
}

}

public static class misc_d
{
    // DLAMCH
    static bool dlamch_first = true;
    public static double _base, emax, emin, eps, prec, rmax, rmin, rnd, sfmin, t;
    // DLAMC1
    static bool dlamc1_first = true, lieee1, lrnd;
    static int dlamc1_lbeta, dlamc1_lt;
    // DLAMC2
    static bool dlamc2_first = true, iwarn = false;
    static int dlamc2_lbeta, lemax, lemin, dlamc2_lt;
    static double leps, lrmax, lrmin;
    // DBLAT1 (COMBLA)
    static int icase, incx, incy, n;
    static bool pass;
    // DBLAT2
    static int infot, noutc;
    static bool lerr, ok;
    static string srnamt;
    static StreamWriter nout; //set in DBLAT2

    static bool myflag;
    // DBEG
    static int i, ic, mi;

    public static double DLAMCH(string cmach)
    {
        // C# LAPACK auxiliary routine (version 1.1) of LAPACK DLAMCH
        //
        // DLAMCH determines double precision machine parameters.
        /*
        * Arguments
        * =====
        *
        * cmach   = Specifies the value to be returned by DLAMCH:
        *          = 'E' or 'e',   DLAMCH := eps
        *          = 'S' or 's',   DLAMCH := sfmin
        *          = 'B' or 'b',   DLAMCH := base
        *          = 'P' or 'p',   DLAMCH := eps*base
        *          = 'N' or 'n',   DLAMCH := t
        *          = 'R' or 'r',   DLAMCH := rnd
        *          = 'M' or 'm',   DLAMCH := emin
        *          = 'U' or 'u',   DLAMCH := rmin
        *          = 'L' or 'l',   DLAMCH := emax
        *          = 'O' or 'o',   DLAMCH := rmax
        */
    }
}

```

```

*
*      where
*
*      eps   = relative machine precision
*      sfmin  = safe minimum, such that 1/sfmin does not overflow
*      base   = base of the machine
*      prec   = eps*base
*      t      = number of (base) digits in the mantissa
*      rnd    = 1.0 when rounding occurs in addition, 0.0 otherwise
*      emin   = minimum exponent before (gradual) underflow
*      rmin   = underflow threshold - base**(emin-1)
*      emax   = largest exponent before overflow
*      rmax   = overflow threshold - (base**emax)*(1-eps)
*
*      =====
*/

// Parameters
const double one = 1.0;
const double zero = 0.0;

// Local Scalars
//static bool first = true;
bool lrnd = false;
int beta = 0, imax = 0, imin = 0, it = 0;
//static double _base, emax, emin, eps, prec, rmax, rmin, rnd, sfmin, t;
double rmach = 0.0, small = 0.0;

if (dlamch_first)
{
    dlamch_first = false;
    DLAMC2(ref beta, ref it, ref lrnd, ref eps, ref imin, ref rmin, ref imax, ref rmax);
    //Console.WriteLine("[0] {1} {2} {3} {4} {5} {6} {7}", beta, it, lrnd, eps, imin, rmin, imax, rmax);
    //Console.ReadKey();
    //Environment.Exit(1);
    _base = beta;
    t = it;

    if (lrnd)
    {
        rnd = one;
        eps = Math.Pow(_base, 1 - it) / 2;
    }
    else
    {
        rnd = zero;
        eps = Math.Pow(_base, 1 - it);
    }

    prec = eps * _base;
    emin = imin;
    emax = imax;
    sfmin = rmin;
    small = one / rmax;

    if (small >= sfmin)
    {
        /*
        * Use SMALL plus a bit, to avoid the possibility of rounding
        * causing overflow when computing 1/sfmin.
        */
        sfmin = small * (one + eps);
    }
}

switch (cmach.ToUpper().Substring(0, 1))
{
    case "E":
        rmach = eps;
        break;
    case "S":
        rmach = sfmin;
        break;
    case "B":
        rmach = _base;
        break;
    case "P":
        rmach = prec;
        break;
    case "N":
        rmach = t;
        break;
    case "R":
        rmach = rnd;
        break;
    case "M":
        rmach = emin;
        break;
    case "U":
        rmach = rmin;
        break;
    case "L":
        rmach = emax;
        break;
    case "O":
        rmach = rmax;
        break;
    //default:
}

return rmach;
}

static void DLAMC1(ref int beta, ref int t, ref bool rnd, ref bool ieee1)
{
    // C# LAPACK auxiliary routine (version 1.1) of the LAPACK DLAMC1
    //
    // DLAMC1 determines the machine parameters given by BETA, T, RND, and IEEE1
    /*
    * Arguments
    * =====
    *
    * BETA      (output) INTEGER
    *           The base of the machine.
    *
    */
}

```

```

* T      (output) INTEGER
*        The number of ( BETA ) digits in the mantissa.
*
* RND     (output) LOGICAL
*        Specifies whether proper rounding ( RND = .TRUE. ) or
*        chopping ( RND = .FALSE. ) occurs in addition. This may not
*        be a reliable guide to the way in which the machine performs
*        its arithmetic.
*
* IEEE1   (output) LOGICAL
*        Specifies whether rounding appears to be done in the IEEE
*        'round to nearest' style.
*
* Further Details
* =====
*
* The routine is based on the routine ENVIRON by Malcolm and
* incorporates suggestions by Gentleman and Marovich. See
*
* Malcolm M. A. (1972) Algorithms to reveal properties of
* Floating-point arithmetic. Comms. of the ACM, 15, 949-951.
*
* Gentleman W. M. and Marovich S. B. (1974) More on algorithms
* that reveal properties of floating point arithmetic units.
* Comms. of the ACM, 17, 276-277.
*
* =====
*/

// Local Scalars
//static bool first=true, lieee1, lrnd;
//static int lbeta, lt;
double a, b, c, f, one, qtr, savec, t1, t2;

if (dlamcl_first)
{
    dlamcl_first = false;
    one = 1;

    /*
    *      LBETA, LIEEE1, LT and LRND are the local values of BETA,
    *      IEEE1, T and RND.
    *
    *      Throughout this routine we use the function DLAMC3 to ensure
    *      that relevant values are stored and not held in registers, or
    *      are not affected by optimizers.
    *
    *      Compute a = 2.0**m with the smallest positive integer m such
    *      that
    *
    *          fl( a + 1.0 ) = a.
    */

    a = 1;
    c = 1;

    while (c == one)
    {
        a = 2 * a;
        c = DLAMC3(a, one);
        c = DLAMC3(c, -a);
    }

    /*
    *      Now compute b = 2.0**m with the smallest positive integer m
    *      such that
    *
    *          fl( a + b ) .gt. a.
    */

    b = 1;
    c = DLAMC3(a, b);

    while (c == a)
    {
        b = 2 * b;
        c = DLAMC3(a, b);
    }

    /*
    *      Now compute the base. a and c are neighbouring floating point
    *      numbers in the interval ( beta**t, beta**( t + 1 ) ) and so
    *      their difference is beta. Adding 0.25 to c is to ensure that it
    *      is truncated to beta and not ( beta - 1 ).
    */
    //Console.WriteLine("{0} {1} {2}", a, b, c);
    //Console.ReadKey();
    //Environment.Exit(1);
    qtr = one / 4;
    savec = c;
    c = DLAMC3(c, -a);
    //Console.WriteLine("c,qtr={0} {1}",c,qtr);
    dlamcl_lbeta = (int)(c + qtr);
    //Console.WriteLine("lbeta={0}", dlamcl_lbeta);

    /*
    *      Now determine whether rounding or chopping occurs, by adding a
    *      bit less than beta/2 and a bit more than beta/2 to a.
    */

    b = dlamcl_lbeta;
    f = DLAMC3(b / 2, -b / 100);
    c = DLAMC3(f, a);

    if (c == a)
    {
        lrnd = true;
    }
    else
    {
        lrnd = false;
    }

    f = DLAMC3(b / 2, b / 100);

```

```

c = DLAMC3(f, a);
//Console.WriteLine("{0} {1:E20} {2:E20} {3} {4:E20} {5} {6}", lrnd, c, a, qtr, savec, b, f);
//Console.ReadKey();
// Environment.Exit(1);
//Console.WriteLine("lrnd, c, a {0} {1:E20} {2:E20}", lrnd, c, a);

if ((lrnd) & (c == a))
{
    lrnd = false;
}
//Console.WriteLine("lrnd = false");
/*
 *      Try and decide whether rounding is done in the IEEE 'round to
 *      nearest' style. B/2 is half a unit in the last place of the two
 *      numbers A and SAVEC. Furthermore, A is even, i.e. has last bit
 *      zero, and SAVEC is odd. Thus adding B/2 to A should not change
 *      A, but adding B/2 to SAVEC should change SAVEC.
 */

t1 = DLAMC3(b / 2, a);
t2 = DLAMC3(b / 2, savec);
lieeel = (t1 == a) & (t2 > savec) & lrnd;

/*
 *      Now find the mantissa, t. It should be the integer part of
 *      log to the base beta of a, however it is safer to determine t
 *      by powering. So we find t as the smallest positive integer for
 *      which
 *
 *      fl( beta**t + 1.0 ) = 1.0.
 */

dlamcl_lt = 0;
a = 1.0;
c = 1.0;

while (c == one)
{
    dlamcl_lt = dlamcl_lt + 1;
    a = a * dlamcl_lbeta;
    c = DLAMC3(a, one);
    c = DLAMC3(c, -a);
}

beta = dlamcl_lbeta;
t = dlamcl_lt;
rnd = lrnd;
ieeel = lieeel;
//Console.WriteLine("{0} {1} {2} {3}", beta, t2, rnd, ieeel);
//Console.ReadKey();
//Environment.Exit(1);
}

}

static void DLAMC2(ref int beta, ref int t, ref bool rnd, ref double eps, ref int emin, ref double rmin, ref int emax, ref double rmax)
{
    // C# LAPACK auxiliary routine (version 1.1) of the LAPACK DLAMC2
    //
    // DLAMC2 determines the machine parameters specified in its argument list.
    /*
    * Arguments
    * =====
    *
    * BETA      (output) INTEGER
    *           The base of the machine.
    *
    * T         (output) INTEGER
    *           The number of ( BETA ) digits in the mantissa.
    *
    * RND       (output) LOGICAL
    *           Specifies whether proper rounding ( RND = .TRUE. ) or
    *           chopping ( RND = .FALSE. ) occurs in addition. This may not
    *           be a reliable guide to the way in which the machine performs
    *           its arithmetic.
    *
    * EPS       (output) DOUBLE PRECISION
    *           The smallest positive number such that
    *
    *           fl( 1.0 - EPS ) .LT. 1.0,
    *
    *           where fl denotes the computed value.
    *
    * EMIN      (output) INTEGER
    *           The minimum exponent before (gradual) underflow occurs.
    *
    * RMIN      (output) DOUBLE PRECISION
    *           The smallest normalized number for the machine, given by
    *           BASE**( EMIN - 1 ), where BASE is the floating point value
    *           of BETA.
    *
    * EMAX      (output) INTEGER
    *           The maximum exponent before overflow occurs.
    *
    * RMAX      (output) DOUBLE PRECISION
    *           The largest positive number for the machine, given by
    *           BASE**EMAX * ( 1 - EPS ), where BASE is the floating point
    *           value of BETA.
    *
    * Further Details
    * =====
    *
    * The computation of EPS is based on a routine PARANOIA by
    * W. Kahan of the University of California at Berkeley.
    *
    * =====
    */

    // Local Scalars
    //static bool first=true, iwarn=false;
    bool ieee = false, lieeel = false, lrnd = false;
    //static int lbeta, lemax, lemin, lt;
    int gnmin = 0, gpmin = 0, i, ngnmin = 0, ngpmin = 0;
    //static double leps, lrmx, lrmin;
    double a, b, c, half, one, rbase, sixth, small, third, two, zero;

```

```

        string hstr = ""
WARNING. The value EMIN may be incorrect: EMIN = {0}
If, after inspection, the value EMIN looks acceptable please comment out
the IF block as marked within the code of routine DLAMC2
otherwise supply EMIN explicitly.
";
    if (dlamc2_first)
    {
        dlamc2_first = false;
        zero = 0.0;
        one = 1.0;
        two = 2.0;

        /*
         *      LBETA, LT, LRND, LEPS, LEMIN and LRMIN are the local values of
         *      BETA, T, RND, EPS, EMIN and RMIN.
         *
         *      Throughout this routine we use the function DLAMC3 to ensure
         *      that relevant values are stored and not held in registers, or
         *      are not affected by optimizers.
         *
         *      DLAMC1 returns the parameters LBETA, LT, LRND and LIEEEL.
         */

        DLAMC1(ref dlamc2_lbata, ref dlamc2_lt, ref lrnd, ref lieeel);
        //Console.WriteLine("{0} {1} {2} {3}", dlamc2_lbata, dlamc2_lt, lrnd, lieeel);
        //Console.ReadKey();
        //Environment.Exit(1);

        /*
         *      Start to find EPS.
         */

        b = dlamc2_lbata;
        //Console.WriteLine("b.1={0}", b);
        //Console.WriteLine("dlamc2_lt={0}", dlamc2_lt);
        a = Math.Pow(b, -dlamc2_lt);
        //Console.WriteLine("a={0}", a);
        leps = a;

        /*
         *      Try some tricks to see whether or not this is the correct EPS.
         */

        b = two / 3.0;
        //Console.WriteLine("b.2={0}", b);
        half = one / 2.0;
        sixth = DLAMC3(b, -half);
        third = DLAMC3(sixth, sixth);
        b = DLAMC3(third, -half);
        //Console.WriteLine("b.3={0}", b);
        b = DLAMC3(b, sixth);
        //Console.WriteLine("b.4={0}", b);
        b = Math.Abs(b);
        //Console.WriteLine("b.5={0}", b);
        //Console.WriteLine("b,leps={0} {1}", b, leps);
        if (b < leps)
        {
            b = leps;
            // Console.WriteLine("b=leps {0}", leps);
        }
        leps = 1.0;
        /*
         *      Console.WriteLine("b={0}", b);
         *      Console.WriteLine("half={0}", half);
         *      Console.WriteLine("sixth={0}", sixth);
         *      Console.WriteLine("third={0}", third);
         *      Console.WriteLine("leps={0}", leps);
         *      Console.ReadKey();
         *      Environment.Exit(1);
         */

        while ((leps > b) & (b > zero))
        {
            leps = b;
            c = DLAMC3(half * leps, Math.Pow(two, 5) * Math.Pow(leps, 2));
            c = DLAMC3(half, -c);
            b = DLAMC3(half, c);
            c = DLAMC3(half, -b);
            b = DLAMC3(half, c);
        }

        if (a < leps)
        {
            leps = a;
        }

        /*
         *      Computation of EPS complete.
         *
         *      Now find EMIN. Let A = + or - 1, and + or - (1 + BASE**(-3)).
         *      Keep dividing A by BETA until (gradual) underflow occurs. This
         *      is detected when we cannot recover the previous A.
         */

        rbase = one / dlamc2_lbata;
        small = one;

        for (i = 1; i <= 3; i = i + 1)
        {
            small = DLAMC3(small * rbase, zero);
        }

        a = DLAMC3(one, small);
        DLAMC4(ref ngpmin, one, dlamc2_lbata);
        DLAMC4(ref ngnmin, -one, dlamc2_lbata);
        DLAMC4(ref gpmin, a, dlamc2_lbata);
        DLAMC4(ref gnmin, -a, dlamc2_lbata);
        ieee = false;

        if ((ngpmin == ngnmin) & (gpmin == gnmin))
        {
            if (ngpmin == gpmin)
            {

```

```

        lemin = ngpmin;

        /*          ( Non twos-complement machines, no gradual underflow;
         *           e.g., VAX )
         */

    }
    else if ((gpmin - ngpmin) == 3)
    {
        lemin = ngpmin - 1 + dlamc2_lt;
        ieee = true;

        /*          ( Non twos-complement machines, with gradual underflow;
         *           e.g., IEEE standard followers )
         */
    }
    else
    {
        lemin = Math.Min(ngpmin, gpmin);
        /*          ( A guess; no known machine )
         */
        iwarn = true;
    }
}
else if ((ngpmin == gpmin) & (ngnmin == gnmin))
{
    if (Math.Abs(ngpmin - ngnmin) == 1)
    {
        lemin = Math.Max(ngpmin, ngnmin);

        /*          ( Twos-complement machines, no gradual underflow;
         *           e.g., CYBER 205 )
         */
    }
    else
    {
        lemin = Math.Min(ngpmin, ngnmin);
        /*          ( A guess; no known machine )
         */
        iwarn = true;
    }
}
else if ((Math.Abs(ngpmin - ngnmin) == 1) & (gpmin == gnmin))
{
    if ((gpmin - Math.Min(ngpmin, ngnmin)) == 3)
    {
        lemin = Math.Max(ngpmin, ngnmin) - 1 + dlamc2_lt;

        /*          ( Twos-complement machines with gradual underflow;
         *           no known machine )
         */
    }
    else
    {
        lemin = Math.Min(ngpmin, ngnmin);
        /*          ( A guess; no known machine )
         */
        iwarn = true;
    }
}
else
{
    lemin = Math.Min(Math.Min(Math.Min(ngpmin, ngnmin), gpmin), gnmin);
    /*          ( A guess; no known machine )
     */
    iwarn = true;
}

/* Comment out this if block if EMIN is ok
if (iwarn)
{
    dlamc2_first = true;
    Console.WriteLine(hstr, lemin);
}
/* Comment out this if block if EMIN is ok

/*          Assume IEEE arithmetic if we found denormalised numbers above,
 *          or if arithmetic seems to round in the IEEE style, determined
 *          in routine DLAMC1. A true IEEE machine should have both things
 *          true; however, faulty machines may have one or the other.
 */

ieee = ieee | lieee1;

/*
 *          Compute RMIN by successive division by BETA. We could compute
 *          RMIN as BASE** ( EMIN - 1 ), but some machines underflow during
 *          this computation.
 */

lrmin = 1.0;
for (i = 1; i <= 1 - lemin; i = i + 1)
{
    lrmin = DLAMC3(lrmin * rbase, zero);
}

/*
 *          Finally, call DLAMC5 to compute EMAX and RMAX.
 */

DLAMC5(dlamc2_lbeta, dlamc2_lt, lemin, ieee, ref lemax, ref lrmax);
}

beta = dlamc2_lbeta;
t = dlamc2_lt;
rnd = lrnd;
eps = leps;
emin = lemin;
rmin = lrmin;
emax = lemax;
rmax = lrmax;
/*          Console.WriteLine("beta={0}",beta);
          Console.WriteLine("t={0}",t);
          Console.WriteLine("rnd={0}",rnd);
          Console.WriteLine("eps={0}",eps);
          Console.WriteLine("emin={0}",emin);

```

```

        Console.WriteLine("rmin={0}",rmin);
        Console.WriteLine("emax={0}",emax);
        Console.WriteLine("rmax={0}",rmax);

        Console.ReadKey();
        Environment.Exit(1);
    }
}

static double DLAMC3(double a, double b)
{
    // C# LAPACK auxiliary routine (version 1.1) of LAPACK DLAMC3
    //
    /* DLAMC3 is intended to force A and B to be stored prior to doing
     * the addition of A and B, for use in situations where optimizers
     * might hold one of these in a register.
     */
    /*
     * Arguments
     * =====
     *
     * A, B      (input) DOUBLE PRECISION
     *           The values A and B.
     *
     * =====
     */
    return (a + b);
}

static void DLAMC4(ref int emin, double start, int _base)
{
    // C# LAPACK auxiliary routine (version 1.1) of LAPACK DLAMC4
    //
    // DLAMC4 is a service routine for DLAMC2.
    //
    /*
     * Arguments
     * =====
     *
     * EMIN      (output) EMIN
     *           The minimum exponent before (gradual) underflow, computed by
     *           setting A = START and dividing by BASE until the previous A
     *           can not be recovered.
     *
     * START     (input) DOUBLE PRECISION
     *           The starting point for determining EMIN.
     *
     * BASE      (input) INTEGER
     *           The base of the machine.
     *
     * =====
     */

    // Local Scalars
    int i;
    double a, b1, b2, c1, c2, d1, d2, one, rbase, zero;

    a = start;
    one = 1.0;
    rbase = one / _base;
    zero = 0.0;
    emin = 1;
    b1 = DLAMC3(a * rbase, zero);
    c1 = a;
    c2 = a;
    d1 = a;
    d2 = a;

    while ((c1 == a) & (c2 == a) & (d1 == a) & (d2 == a))
    {
        emin = emin - 1;
        a = b1;
        b1 = DLAMC3(a / _base, zero);
        c1 = DLAMC3(b1 * _base, zero);
        d1 = zero;

        for (i = 1; i <= _base; i = i + 1)
        {
            d1 = d1 + b1;
        }

        b2 = DLAMC3(a * rbase, zero);
        c2 = DLAMC3(b2 / rbase, zero);
        d2 = zero;

        for (i = 1; i <= _base; i = i + 1)
        {
            d2 = d2 + b2;
        }
    }
}

static void DLAMC5(int beta, int p, int emin, bool ieee, ref int emax, ref double rmax)
{
    // C# LAPACK auxiliary routine (version 1.1) of LAPACK DLAMC5
    //
    /* DLAMC5 attempts to compute RMAX, the largest machine floating-point
     * number, without overflow. It assumes that EMAX + abs(EMIN) sum
     * approximately to a power of 2. It will fail on machines where this
     * assumption does not hold, for example, the Cyber 205 (EMIN = -28625,
     * EMAX = 28718). It will also fail if the value supplied for EMIN is
     * too large (i.e. too close to zero), probably with overflow.
     */
    /*
     * Arguments
     * =====
     *
     * BETA      (input) INTEGER
     *           The base of floating-point arithmetic.
     *
     * P         (input) INTEGER
     *           The number of base BETA digits in the mantissa of a
     *           floating-point value.
     *
     * EMIN      (input) INTEGER
     *           The minimum exponent before (gradual) underflow.
     *
     * IEEE     (input) LOGICAL
     *           A logical flag specifying whether or not the arithmetic
    */

```

```

*      system is thought to comply with the IEEE standard.
*
*      EMAX      (output) INTEGER
*                  The largest exponent before overflow
*
*      RMAX      (output) DOUBLE PRECISION
*                  The largest machine floating-point number.
*
*      =====
*/

// Parameters
const double zero = 0.0, one = 1.0;

// Local Scalars
int exbits, expsum, i, lexp, nbits, _try, uexp;
double oldy = 0.0, recbas, y, z;

/*      First compute LEXP and UEXP, two powers of 2 that bound
*      abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
*      approximately to the bound that is closest to abs(EMIN).
*      (EMAX is the exponent of the required number RMAX).
*/

lexp = 1;
exbits = 1;

/*      10 CONTINUE
      TRY = LEXP*2
      IF( TRY.LE.( -EMIN ) ) THEN
          LEXP = TRY
          EXBITS = EXBITS + 1
          GO TO 10
      END IF
*/

_try = lexp * 2;
while (_try <= (-emin))
{
    lexp = _try;
    exbits = exbits + 1;
    _try = lexp * 2;
}

if (lexp == -emin)
{
    uexp = lexp;
}
else
{
    uexp = _try;
    exbits = exbits + 1;
}

/*      Now -LEXP is less than or equal to EMIN, and -UEXP is greater
*      than or equal to EMIN. EXBITS is the number of bits needed to
*      store the exponent.
*/

if ((uexp + emin) > (-lexp - emin))
{
    expsum = 2 * lexp;
}
else
{
    expsum = 2 * uexp;
}

/*      EXPSUM is the exponent range, approximately equal to
*      EMAX - EMIN + 1 .
*/

emax = expsum + emin - 1;
nbits = 1 + exbits + p;

/*      NBITS is the total number of bits needed to store a
*      floating-point number.
*/

if (((nbits % 2) == 1) & (beta == 2))
{
    /*      Either there are an odd number of bits used to store a
    *      floating-point number, which is unlikely, or some bits are
    *      not used in the representation of numbers, which is possible,
    *      (e.g. Cray machines) or the mantissa has an implicit bit,
    *      (e.g. IEEE machines, Dec Vax machines), which is perhaps the
    *      most likely. We have to assume the last alternative.
    *      If this is true, then we need to reduce EMAX by one because
    *      there must be some way of representing zero in an implicit-bit
    *      system. On machines like Cray, we are reducing EMAX by one
    *      unnecessarily.
    */

    emax = emax - 1;
}

if (ieee)
{
    /*      Assume we are on an IEEE machine which reserves one exponent
    *      for infinity and NaN.
    */

    emax = emax - 1;
}

/*      Now create RMAX, the largest machine number, which should
*      be equal to (1.0 - BETA**(-P)) * BETA**EMAX .
*
*      First compute 1.0 - BETA**(-P), being careful that the
*      result is less than 1.0 .
*/

```

```

        recbas = one / beta;
        z = beta - one;
        y = zero;
        for (i = 1; i <= p; i = i + 1)
        {
            z = z * recbas;
            if (y < one)
            {
                oldy = y;
            }
            y = DLAMC3(y, z);
        }

        if (y >= one)
        {
            y = oldy;
        }

        /*      Now multiply by BETA**EMAX to get RMAX.
        */

        for (i = 1; i <= emax; i = i + 1)
        {
            y = DLAMC3(y * beta, zero);
        }

        rmax = y;
    }
    public static long DSECND()
    {
        // C# of LAPACK auxiliary routine DSECND returns the user time for a process in seconds.
        //long gg = DateTimeOffset.UtcNow.UtcTicks;

        return DateTimeOffset.UtcNow.UtcTicks;
    }
    public static void DSECNDTST()
    {
        // C# of LAPACK test routine
        // Parameters
        const int nmax = 1000;
        const int its = 50000;

        // Local Scalars
        int i, j;
        double alpha, avg, tnosec, total;
        long t1, t2;
        double tnosec2;
        double tnosecm, tnosec2m, avgm;

        // Local Arrays
        double[] x = new double[nmax];
        double[] y = new double[nmax];
        //Console.WriteLine("{0}", y.Length);
        //return;
        // Figure TOTAL flops ..
        total = (double)nmax * (double)its * 2.0;

        // Initialize X and Y
        for (i = 0; i < nmax; i = i + 1)
        {
            //Console.WriteLine("i={0}", i);
            x[i] = 1.0 / ((double)i);
            y[i] = (double)(nmax - i) / ((double)nmax);
        }
        alpha = 0.315;

        // Time TOTAL DAXPY operations
        t1 = DSECND();

        for (j = 0; j < its; j = j + 1)
        {
            for (i = 0; i < nmax; i = i + 1)
            {
                y[i] = y[i] + alpha * x[i];
            }
            alpha = -alpha;
        }

        t2 = DSECND();
        tnosec = (new TimeSpan(t2 - t1)).TotalSeconds;
        //===tnosecm = (new TimeSpan(t2 - t1)).TotalMilliseconds;
        //Console.WriteLine("{0} {1} {2}", t1, t2, new TimeSpan(t2-t1).Ticks);
        Console.WriteLine("Time for {0,10:G3} DAXPY ops = {1,10:G3} seconds", total, tnosec);
        //Console.WriteLine("Time for {0,10:G3} DAXPY ops = {1,10:G3} milliseconds", total, tnosecm);
        if (tnosec > 0.0)
        //if (tnosecm > 0.0)
        {
            Console.WriteLine("DAXPY performance rate      = {0,10:G3} mflops", (total / 1.0e6) / tnosec);
            //Console.WriteLine("DAXPY performance rate      = {0,10:G3} mflops", (total / 1.0e3) / tnosecm);
        }
        else
        {
            Console.WriteLine("*** Warning: Time for operations was less or equal than zero => timing in TESTING might be dubious");
        }

        // Time TOTAL DAXPY operations with DSECND in the outer loop
        t1 = DSECND();

        for (j = 0; j < its; j = j + 1)
        {
            for (i = 0; i < nmax; i = i + 1)
            {
                y[i] = y[i] + alpha * x[i];
            }
            alpha = -alpha;
            t2 = DSECND();

            tnosec2 = (new TimeSpan(t2 - t1)).TotalSeconds;
            //===tnosec2m = (new TimeSpan(t2 - t1)).TotalMilliseconds;
            // Compute the time used in milliseconds used by an average call to DSECND.
            Console.WriteLine("Including DSECND, time      = {0,10:G3} seconds", tnosec2);
            //Console.WriteLine("Including DSECND, time      = {0,10:G3} milliseconds", tnosec2m);
            avg = (tnosec2 - tnosec) * 1000.0 / ((double)its);

```

```

//avgm = (tnosec2m - tnosecm) / ((double)its);
if (avg > 0.0)
//if (avgm > 0.0)
{
    Console.WriteLine("Average time for DSECND      = {0,10:G3} milliseconds", avg);
    //Console.WriteLine("Average time for DSECND      = {0,10:G3} milliseconds", avgm);
}

// Compute the equivalent number of floating point operations used by an average call to DSECND.
if ( ( avg > 0.0 ) & ( tnosec > 0.0 ) )
//if ( (avgm > 0.0) & (tnosecm > 0.0))
{
    Console.WriteLine("Equivalent floating point ops = {0,10:G3} ops", (avg / 1000) * total / tnosec);
    //Console.WriteLine("Equivalent floating point ops = {0,10:G3} ops", (avgm) * total / tnosecm);
}
MYSUB(nmax, x, y);
}
public static void MYSUB(int nmax, double[] x, double[] y)
{
    return;
}
/*
public static void DROTG(double a, double b, double c, double d)
{
    return;
}
public static double DNRM2(double a, double[] b, double c)
{
    return 1.0;
}
public static void DROTMG(double a, double b, double c, double d, double e)
{
    return;
}
*/
public static void DBLAT1()
{
    // C# version of Test program for the DOUBLE PRECISION Level 1 BLAS (version 3.4.1)

    // Parameters
    const int nout = 6;

    // Local Scalars
    double sfac = 9.765625E-4;
    int ic;
    /* .. External Subroutines ..
    //      EXTERNAL      CHECK0, CHECK1, CHECK2, CHECK3, HEADER

    // Executable Statements
    Console.WriteLine("Real BLAS Test Program Results");
    Console.WriteLine();

    for (ic = 1; ic <= 13; ic = ic + 1)
    {
        icense = ic;
        HEADER();

        // Initialize PASS, INCX, and INCY for a new case. the value 9999 for INCX or INCY will appear in the detailed output, if any,
        // for cases that do not involve these parameters
        pass = true;
        incx = 9999;
        incy = 9999;
        if ((icase == 3) | (icase == 11))
        {
            CHECK0(sfac);
            //Console.WriteLine("*****");
            //Environment.Exit(1);
        }
        else if ((icase == 7) | (icase == 8) | (icase == 9) | icase == 10)
        {
            CHECK1(sfac);
        }
        else if ((icase == 1) | (icase == 2) | (icase == 5) | (icase == 6) | (icase == 12) | (icase == 13))
        {
            CHECK2(sfac);
        }
        else if (icase == 4)
        {
            CHECK3(sfac);
        }

        if (pass)
        {
            Console.WriteLine("          ----- PASS -----");
        }
    }
} //TEST uses misc.d.eps, TESTDSDOT uses misc.s.eps call DLAMCH/SLAMCH first
static double[] get_darr(int start, int end, double[] arr)
{
    double[] dtmp = new double[end - start + 1];
    int cnt = 0;
    for (int ii = start; ii <= end; ii = ii + 1)
    {
        dtmp[cnt] = arr[ii];
        cnt++;
    }
    return dtmp;
}
static double[] get_darr_full(int l, int start, int end, double[] arr)
{
    //double[] dtmp = new double[end - start + 1];
    double[] dtmp = new double[1];
    int cnt = 0;
    for (int ii = start; ii <= end; ii = ii + 1)
    {
        dtmp[cnt] = arr[ii];
        cnt++;
    }
    return dtmp;
}
static float[] get_darrf(int start, int end, double[] arr)
{

```

```

float[] dtmp = new float[end - start + 1];
int cnt = 0;
for (int ii = start; ii <= end; ii = ii + 1)
{
    dtmp[cnt] = (float)arr[ii];
    cnt++;
}
return dtmp;
}
static double[] get_darr2d(int start, int end, int endl, double[,] arr)
{
    double[] dtmp = new double[end - start + 1];
    int cnt = 0;
    for (int ii = start; ii <= end; ii = ii + 1)
    {
        dtmp[cnt] = arr[ii, endl];
        cnt++;
    }
    return dtmp;
}
static float[] get_darr2dfError! Bookmark not defined.(int start, int end, int endl, double[,] arr)
{
    float[] dtmp = new float[end - start + 1];
    int cnt = 0;
    for (int ii = start; ii <= end; ii = ii + 1)
    {
        dtmp[cnt] = (float)arr[ii, endl];
        cnt++;
    }
    return dtmp;
}
/*
static double mysign(double a, double b)
{
    Console.WriteLine("{0} {1} {2}", a, b, Math.Sign(b));
    if (b < 0.0) {
        return - Math.Abs(a);
    } else {
        return Math.Abs(a);
    }
}
*/
static void HEADER()
{
    // Parameters
    const int nout = 6;

    // Local Arrays
    String[] l = {
        "DDOT ",
        "DAXPY ",
        "DROTG ",
        "DROT ",
        "DCOPY ",
        "DSWAP ",
        "DNRM2 ",
        "DSUM ",
        "DSCAL ",
        "IDAMAX",
        "DROTMG",
        "DROTM ",
        "DSDOT "};
    //          COMMON          /COMBLA/ICASE, N, INCX, INCY, PASS

    /*{ index[,alignment][ :formatString] }
    Console.WriteLine("");
    /*,'I3,I2X,A6)ICASE, L(ICASE)
    Console.WriteLine("Test of subprogram number {0,3:D}          {1:-6}", icase, l[icase - 1]);
    */
}
static void CHECK0(double sfac)
{
    // Parameters
    const int nout = 6;

    // Local Scalars
    double sa, sb, sc = 0.0, ss = 0.0, d12 = 4096.0;

    int i, k;
    double[] dparam = new double[5];
    // Local Arrays
    // DOUBLE PRECISION  DAL(8), DATRUE(8), DB1(8), DBTRUE(8), DCL(8),
    // $                DSI(8), DAB(4,9), DTEMP(9), DTRUE(9,9)
    //          EXTERNAL  DROTG, DROTMG, STEST1
    double[] dal = { 0.3, 0.4, -0.3, -0.4, -0.3, 0.0, 0.0, 1.0 };
    double[] datrue = { 0.5, 0.5, 0.5, -0.5, -0.5, 0.0, 1.0, 1.0 };
    double[] db1 = { 0.4, 0.3, 0.4, 0.3, -0.4, 0.0, 1.0, 0.0 };
    double[] dbtrue = { 0.0, 0.6, 0.0, -0.6, 0.0, 0.0, 1.0, 0.0 };
    double[] dcl = { 0.6, 0.8, -0.6, 0.8, 0.6, 1.0, 0.0, 1.0 };
    double[] dsl = { 0.8, 0.6, 0.8, -0.6, 0.8, 0.0, 1.0, 0.0 };
    // column major for fortran
    double[,] dab = {{0.1,0.7,0.0, 4.0,6.0e-10, 4.0e10, 2.0e-10, 2.0e10, 4.0},
        {0.3, 0.2, 0.0, -1.0, 2.0e-2, 2.0e-2, 4e-2, 4.0e-2, -2.0},
        {1.2,0.6,0.0, 2.0, 1.0e5, 1.0e-5, 1.0e5, 1.0e-5, 8.0},
        {0.2,4.2,0.0, 4.0, 10.0, 10.0, 10.0, 10.0, 4.0}};

    /* INPUT FOR MODIFIED GIVENS
    DATA DAB/ .1D0,.3D0,1.2D0,.2D0,
    A          .7D0,.2D0,.6D0,4.2D0,
    B          0.D0,0.D0,0.D0,0.D0,
    C          4.D0,-1.D0,2.D0,4.D0,
    D          6.D-10,2.D-2,1.D5,10.D0,
    E          4.D10,2.D-2,1.D-5,10.D0,
    F          2.D-10,4.D-2,1.D5,10.D0,
    G          2.D10,4.D-2,1.D-5,10.D0,
    H          4.D0,-2.D0,8.D0,4.D0 */

    double[] dtemp = { 0, 0, 0, 0, 0, 0, 0, 0 };
    double[,] dtrue = { {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0, 15e-3, 0.0, 0.0, 0.0, 0.0},
        {1.3, 4.5, 0.0, 0.0, 0.0, 6144e-5, 15.0, 15.0, 7.0},
        {0.2, 4.2, 0.0, 4.0, 10.0, 10.0, 10.0, 10.0, 4.0},
        {0.0, 1.0, -2.0,-1.0,-1.0, -1.0, -1.0, -1.0, 0.0},
        {0.0, 0.5, 0.0, 0.0, 0.0, 4096.0, 5e-5, 5e5, 0.0},
    }

```

```

        {0.0, 0.0, 0.0, 0.0, -1e-4, -1e6, 0.0, -4096.0, -0.5},
        {0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, -0.25},
        {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 4096e-6, 0.0});
/* TRUE RESULTS FOR MODIFIED GIVENS
DATA DTRUE/0.0D,0.0D, 1.3D0, .2D0, 0.0D,0.0D,0.0D, .5D0, 0.0D,
A 0.0D,0.0D, 4.5D0, 4.2D0, 1.D0, .5D0, 0.0D,0.0D,0.0D,
B 0.0D,0.0D,0.0D,0.0D, -2.D0, 0.0D,0.0D,0.0D,0.0D,
C 0.0D,0.0D,0.0D, 4.D0, -1.D0, 0.0D,0.0D,0.0D,0.0D,
D 0.0D, 15.D-3, 0.0D, 10.D0, -1.D0, 0.0D, -1.D-4, 0.D0, 1.D0,
F 0.0D,0.0D, 6144.D-5, 10.D0, -1.D0, 4096.D0, -1.D6, 0.D0, 1.D0,
I 0.0D,0.0D,15.D0,10.D0,-1.D0, 5.D-5, 0.D0,1.D0,0.D0,
K 0.0D,0.0D, 7.D0, 4.D0, 0.0D,0.0D, -.5D0, -.25D0, 0.D0/
* 4096 = 2 ** 12}); */

dtrue[0, 0] = 12.0 / 130.0;
dtrue[1, 0] = 36.0 / 130.0;
dtrue[6, 0] = -1.0 / 6.0;
dtrue[0, 1] = 14.0 / 75.0;
dtrue[1, 1] = 49.0 / 75.0;
dtrue[8, 1] = 1.0 / 7.0;
dtrue[0, 4] = 45e-11 * (d12 * d12);
dtrue[2, 4] = 4.0e5 / (3.0 * d12);
dtrue[5, 4] = 1.0 / d12;
dtrue[7, 4] = 1e4 / (3.0 * d12);
dtrue[0, 5] = 4e10 / (1.5 * d12 * d12);
dtrue[1, 5] = 2e-2 / 1.5;
dtrue[7, 5] = 5e-7 * d12;
dtrue[0, 6] = 4.0 / 150.0;
dtrue[1, 6] = (2e-10 / 1.5) * (d12 * d12);
dtrue[6, 6] = -dtrue[5, 4];
dtrue[8, 6] = 1e4 / d12;
dtrue[0, 7] = dtrue[0, 6];
dtrue[1, 7] = 2e10 / (1.5 * d12 * d12);
dtrue[0, 8] = 32.0 / 7.0;
dtrue[1, 8] = -16.0 / 7.0;

// Compute true values which cannot be prestored in decimal notation

dbtrue[0] = 1.0 / 0.6;
dbtrue[2] = -1.0 / 0.6;
dbtrue[4] = 1.0 / 0.6;

for (k = 1; k <= 8; k = k + 1)
{
    // Set N=K for identification in output if any
    n = k;
    if (icase == 3)
    {
        // DROTG
        if (k > 8)
        {
            return;
        }
        sa = dal[k - 1];
        //Console.WriteLine("{0}", sa);

        sb = dbl[k - 1];
        //Console.WriteLine("{0} {1} {2} {3}", sa, sb, sc, ss);
        DROTG(ref sa, ref sb, ref sc, ref ss);
        //Console.WriteLine("{0} {1} {2} {3}", sa, sb, sc, ss);
        //Console.WriteLine("{0}", sa);
        //Environment.Exit(1);
        //STEST1(sa, (datrue[k-1]), (datrue[k-1]), sfac);
        //Console.WriteLine("====1");
        STEST(1, new double[] { sa }, get_darr(k - 1, 7, datrue), get_darr(k - 1, 7, datrue), sfac);
        //STEST1(sb, dbtrue[k-1], dbtrue[k-1], sfac);
        //Console.WriteLine("====2");
        STEST(1, new double[] { sb }, get_darr(k - 1, 7, dbtrue), get_darr(k - 1, 7, dbtrue), sfac);
        //STEST1(sc, dcl[k-1], dcl[k-1], sfac);
        //Console.WriteLine("====3");
        STEST(1, new double[] { sc }, get_darr(k - 1, 7, dcl), get_darr(k - 1, 7, dcl), sfac);
        //STEST1(ss, dsl[k-1], dsl[k-1], sfac);
        //Console.WriteLine("====4");
        STEST(1, new double[] { ss }, get_darr(k - 1, 7, dsl), get_darr(k - 1, 7, dsl), sfac);
        //Console.WriteLine("====5");
    }
    else if (icase == 11)
    {
        // DROTMG
        for (i = 1; i <= 4; i = i + 1)
        {
            dtemp[i - 1] = dab[i - 1, k - 1];
            dtemp[i + 3] = 0.0;
        }
        dtemp[8] = 0.0;
        dparam = get_darr(4, 8, dtemp);
        //DROTMG(dtemp[0], dtemp[1], dtemp[2], dtemp[3], get_darr(4,8,dtemp));
        dparam[0] = dtemp[4];
        dparam[1] = dtemp[5];
        dparam[2] = dtemp[6];
        dparam[3] = dtemp[7];
        dparam[4] = dtemp[8];
        /*Console.WriteLine("{0}",dtemp[0]);
        Console.WriteLine("{0}", dtemp[1]);
        Console.WriteLine("{0}", dtemp[2]);
        Console.WriteLine("{0}", dtemp[3]);
        Console.WriteLine("{0}", dtemp[4]);
        Console.WriteLine("{0}", dtemp[5]);
        Console.WriteLine("{0}", dtemp[6]);
        Console.WriteLine("{0}", dtemp[7]);
        Console.WriteLine("{0}", dtemp[8]);*/
        DROTMG(ref dtemp[0], ref dtemp[1], ref dtemp[2], ref dtemp[3], ref dparam);

        dtemp[4] = dparam[0];
        dtemp[5] = dparam[1];
        dtemp[6] = dparam[2];
        dtemp[7] = dparam[3];
        dtemp[8] = dparam[4];
        /*
        Console.WriteLine("{0}", dtemp[0]);
        Console.WriteLine("{0}", dtemp[1]);
        Console.WriteLine("{0}", dtemp[2]);
        Console.WriteLine("{0}", dtemp[3]);

```

```

        Console.WriteLine("{0}", dtemp[4]);
        Console.WriteLine("{0}", dtemp[5]);
        Console.WriteLine("{0}", dtemp[6]);
        Console.WriteLine("{0}", dtemp[7]);
        Console.WriteLine("{0}", dtemp[8]);
        Environment.Exit(1);*/
        STEST(9, dtemp, get_darr2d(0, 8, k - 1, dtrue), get_darr2d(0, 8, k - 1, dtrue), sfac);
    }
    else
    {
        Console.WriteLine("Shouldn't be here in CHECK0");
        Environment.Exit(1);
    }
}

static void CHECK1(double sfac)
{
    // Parameters
    const int nout = 6;

    // Local Scalars
    int i, len, npl;

    //DOUBLE PRECISION   DTRUE1(5), DTRUE3(5), DTRUE5(8,5,2), DV(8,5,2),
    //+                  SA(10), STEMP(1), STRUE(8), SX(8)
    //INTEGER             ITRUE2(5)
    // Local Arrays
    double[] dtrue1 = { 0.0, 0.3, 0.5, 0.7, 0.6 };
    double[] dtrue3 = { 0.0, 0.3, 0.7, 1.1, 1.0 };
    double[, ] dtrue5 = { { {0.1, 0.1}, {-0.3, 0.09}, {0.0, 0.09}, {0.2, 0.06}, {0.03, 0.03}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {-0.6, 3.0}, {-0.09, 4.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, -0.12}, {0.3, -0.18}, {0.15, -0.09}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 5.0}, {-0.03, 6.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 0.09}, {6.0, -0.15}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, 7.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, -0.03}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, 3.0}} };
    double[, ] dv = { { {0.1, 0.1}, {0.3, 0.3}, {0.3, 0.3}, {0.2, 0.2}, {0.1, 0.1}},
        { {2.0, 8.0}, {3.0, 9.0}, {-0.4, 2.0}, {-0.6, 3.0}, {-0.3, 4.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, -0.4}, {0.3, -0.6}, {0.5, -0.3}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 5.0}, {-0.1, 6.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 0.3}, {6.0, -0.5}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, 7.0}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, -0.1}},
        { {2.0, 8.0}, {3.0, 9.0}, {4.0, 2.0}, {5.0, 2.0}, {6.0, 3.0}} };
    double[] sa = { 0.3, -1.0, 0.0, 1.0, 0.3, 0.3, 0.3, 0.3, 0.3 };
    double[] stemp = { 0 };
    double[] strue = { 0, 0, 0, 0, 0, 0, 0, 0 };
    double[] sx = { 0, 0, 0, 0, 0, 0, 0, 0 };
    int[] itrue2 = { 0, 1, 2, 2, 3 };

    //COMMON          /COMBLA/ICASE, N, INCX, INCY, PASS

    // Executable Statements
    for (incx = 1; incx <= 2; incx = incx + 1)
    {
        for (npl = 1; npl <= 5; npl = npl + 1)
        {
            n = npl - 1;
            len = 2 * Math.Max(n, 1);
            // Set vector arguments
            for (i = 1; i <= len; i = i + 1)
            {
                sx[i - 1] = dv[i - 1, npl - 1, incx - 1];
            }
            if (icase == 7)
            {
                // DNRM2
                stemp[0] = dtrue1[npl - 1];
                //STEST1(DNRM2(n,sx,incx), stemp[0],stemp,sfac);
                STEST(1, new double[] { DNRM2(n, sx, incx) }, new double[] { stemp[0] }, stemp, sfac);
            }
            else if (icase == 8)
            {
                // DASUM
                stemp[0] = dtrue3[npl - 1];
                //STEST1(DASUM(n, sx, incx), stemp[0], stemp, sfac);
                STEST(1, new double[] { DASUM(n, sx, incx) }, new double[] { stemp[0] }, stemp, sfac);
            }
            else if (icase == 9)
            {
                // DSCAL
                DSCAL(n, sa[(incx - 1) * 5 + npl - 1], ref sx, incx);
                for (i = 1; i <= len; i = i + 1)
                {
                    strue[i - 1] = dtrue5[i - 1, npl - 1, incx - 1];
                }
                STEST(len, sx, strue, strue, sfac);
            }
            else if (icase == 10)
            {
                // IDAMAX
                ITTEST1(IDAMAX(n, sx, incx), itrue2[npl - 1]);
            }
            else
            {
                Console.WriteLine("Shouldn't be here in CHECK1");
                Environment.Exit(1);
            }
        }
    }
}

static void CHECK2(double sfac)
{
    // Parameters
    const int nout = 6;

    // Local Scalars
    double sa = 0.3;
    int i, j, ki, kn, kni, kpar, ksize, lenx, leny, mx, my;

    /*      DOUBLE PRECISION  DT10X(7,4,4), DT10Y(7,4,4), DT7(4,4),
       $                      DT8(7,4,4), DX1(7),
       $                      DY1(7), SSIZE1(4), SSIZE2(14,2), SSIZE(7),

```

```

$          STX(7), STY(7), SX(7), SY(7),
$          DPAR(5,4), DT19X(7,4,16),DT19XA(7,4,4),
$          DT19XB(7,4,4), DT19XC(7,4,4),DT19XD(7,4,4),
$          DT19Y(7,4,16), DT19YA(7,4,4),DT19YB(7,4,4),
$          DT19YC(7,4,4), DT19YD(7,4,4), DTEMP(5)
$          INCXS(4), INCYS(4), LENS(4,2), NS(4) */
// Local Arrays
double[, ] dt10x = {{{0.6, 0.6, 0.6, 0.6 } , {0.5, 0.5, 0.5, 0.5} , {0.5, 0.3, -0.9, 0.5} , {0.5, 0.8, 0.7, 0.5}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {-0.9, 0.1, 0.1, 0.3} , {-0.9, 0.1, 0.1, 0.3}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.5, 0.5, 0.0} , {0.3, -0.6, 0.3, -0.6}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.7, 0.8, 0.8, 0.8}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.3, -0.9, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, -0.3, -0.3, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.5, 0.5, 0.0}}}} ,
double[, ] dt10y = {{{0.5, 0.5, 0.5, 0.5} , {0.6, 0.6, 0.6, 0.6} , {0.6, -0.5, -0.5, 0.6} , {0.6, -0.4, -0.4, 0.6}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.1, -0.9, 0.6, -0.9} , {0.1, -0.9, 0.9, -0.9}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.6, 0.0, 0.1} , {-0.5, 0.9, -0.5, 0.1}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.8, 0.7, 0.6, 0.7}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, -0.5, 0.0, -0.5}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.2, 0.0, 0.2}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.6, 0.0, 0.8}}}} ,
double[, ] dt7 = { { 0.0, 0.0, 0.0, 0.0 } , { 0.3, 0.3, 0.3, 0.3 } , { 0.21, -0.07, -0.79, 0.33 } , { 0.62, 0.85, -0.74, 1.27 } } ;
//          DT7/0.000, 0.300D0, 0.21D0, 0.62D0, 0.00D0, 0.30D0, -0.07D0, 0.85D0, 0.00D0, 0.30D0, -0.79D0, -0.74D0, 0.00D0, 0.30D0, 0.33D0,
1.27D0/

double[, ] dt8 = {{{0.5, 0.5, 0.5, 0.5} , {0.68, 0.68, 0.68, 0.68} , {0.68, 0.35, 0.35, 0.68} , {0.68, 0.38, 0.38, 0.68}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {-0.87, -0.9, -0.72, -0.9} , {-0.87, -0.9, -0.63, -0.9}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.48, 0.0, 0.33} , {0.15, 0.57, 0.15, 0.33}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.94, 0.7, 0.88, 0.7}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, -0.75, 0.0, -0.75}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.2, 0.0, 0.2}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.98, 0.0, 1.04}}}} ;
double[] dx1 = { 0.6, 0.1, -0.5, 0.8, 0.9, -0.3, -0.4 } ;
double[] dy1 = { 0.5, -0.9, 0.3, 0.7, -0.6, 0.2, 0.8 } ;
double[] ssize1 = { 0.0, 0.3, 1.6, 3.2 } ;
double[, ] ssize2 = { { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , {
0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } , { 0.0, 1.17 } } ;
double[] stx = new double[7];
double[] sty = new double[7];
double[] sx = new double[7];
double[] sy = new double[7];
double[, ] dpar = { { -2.0, -1.0, 0.0, 1.0 } , { 0.0, 2.0, 0.0, 5.0 } , { 0.0, -3.0, 2.0, 2.0 } , { 0.0, -4.0, -3.0, 0.0 } , { 0.0, 5.0, 0.0, -4.0 }
};

double[, ] dt19x = new double[7, 4, 16];
double[, ] dt19xa = {{{0.6, 0.6, 0.6, 0.6} , {0.6, -0.8, -0.8, -0.8} , {0.6, -0.9, -0.9, -0.9} , {0.6, 3.5, 3.5, 3.5}} ,
{{{0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 3.8, 3.8} , {0.0, 0.0, 2.8, 2.8} , {0.0, 0.0, -0.4, -0.4}} ,
{{{0.0, 0.0, 0.0, -0.5} , {0.0, 0.0, 0.0, -2.2} , {0.0, 0.0, 0.0, -1.4} , {0.0, 0.0, 0.0, -2.2}} ,
{{{0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, -1.2} , {0.0, 0.0, 0.0, -1.3} , {0.0, 0.0, 0.0, 4.7}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}}}} ;
/**          TRUE X RESULTS FOR ROTATIONS DROTm

double[, ] dt19xb = {{{0.6, 0.6, 0.6, 0.6} , {0.6, -0.8, 0.0, -2.0} , {0.6, -0.9, -0.3, -1.8} , {0.6, 3.5, 3.3, 3.8}} ,
{{{0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1}} ,
{{{0.0, 0.0, -0.5, -0.5} , {0.0, 0.0, -3.0, 1.4} , {0.0, 0.0, -2.0, 1.3} , {0.0, 0.0, -2.0, -3.1}} ,
{{{0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8}} ,
{{{0.0, 0.0, 0.0, 0.9} , {0.0, 0.0, 0.0, 0.6} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 4.8}} ,
{{{0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3}} ,
{{{0.0, 0.0, 0.0, -0.4} , {0.0, 0.0, 0.0, -2.8} , {0.0, 0.0, 0.0, -1.9} , {0.0, 0.0, 0.0, -1.5}}}} ;

double[, ] dt19xc = {{{0.6, 0.6, 0.6, 0.6} , {0.6, -0.8, 4.8, -1.6} , {0.6, -0.9, 3.3, -1.5} , {0.6, 3.5, 2.1, 3.7}} ,
{{{0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, 0.1, 0.1}} ,
{{{0.0, 0.0, -0.5, -0.5} , {0.0, 0.0, -3.0, -2.2} , {0.0, 0.0, -2.0, -1.4} , {0.0, 0.0, -2.0, -2.2}} ,
{{{0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 0.8}} ,
{{{0.0, 0.0, 0.0, 0.9} , {0.0, 0.0, 0.0, 5.4} , {0.0, 0.0, 0.0, 3.6} , {0.0, 0.0, 0.0, 3.6}} ,
{{{0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3} , {0.0, 0.0, 0.0, -0.3}} ,
{{{0.0, 0.0, 0.0, -0.4} , {0.0, 0.0, 0.0, -2.8} , {0.0, 0.0, 0.0, -1.9} , {0.0, 0.0, 0.0, -1.5}}}} ;

double[, ] dt19xd =
{{{0.6, 0.6, 0.6, 0.6} , {0.6, -0.8, -0.8, -0.8} , {0.6, -0.9, -0.9, -0.9} , {0.6, 3.5, 3.5, 3.5}} ,
{{{0.0, 0.0, 0.1, 0.1} , {0.0, 0.0, -1.0, -1.0} , {0.0, 0.0, -0.8, -0.8} , {0.0, 0.0, 0.8, 0.8}} ,
{{{0.0, 0.0, 0.0, -0.5} , {0.0, 0.0, 0.0, 1.4} , {0.0, 0.0, 0.0, 1.3} , {0.0, 0.0, 0.0, -3.1}} ,
{{{0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, -1.6} , {0.0, 0.0, 0.0, -1.6} , {0.0, 0.0, 0.0, 4.8}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}}}} ;

double[, ] dt19ya = new double[7, 4, 16];
double[, ] dt19yb =
{{{0.5, 0.5, 0.5, 0.5} , {0.5, 0.7, 0.7, 0.7} , {0.5, 1.7, 1.7, 1.7} , {0.5, -2.6, -2.6, -2.6}} ,
{{{0.0, 0.0, -0.9, -0.9} , {0.0, 0.0, -4.8, -4.8} , {0.0, 0.0, -0.7, -0.7} , {0.0, 0.0, 3.5, 3.5}} ,
{{{0.0, 0.0, 0.0, 0.3} , {0.0, 0.0, 0.0, 3.0} , {0.0, 0.0, 0.0, -0.7} , {0.0, 0.0, 0.0, -0.7}} ,
{{{0.0, 0.0, 0.0, 0.7} , {0.0, 0.0, 0.0, 1.1} , {0.0, 0.0, 0.0, 2.3} , {0.0, 0.0, 0.0, -3.6}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}}}} ;
//UE Y RESULTS FOR ROTATIONS DROTm
double[, ] dt19yb =
{{{0.5, 0.5, 0.5, 0.5} , {0.5, 0.7, 4.0, 3.7} , {0.5, 1.7, -0.5, -0.3} , {0.5, -2.6, -1.5, -1.6}} ,
{{{0.0, 0.0, -0.9, -0.9} , {0.0, 0.0, -0.9, -0.9} , {0.0, 0.0, -0.9, -0.9} , {0.0, 0.0, -0.9, -0.9}} ,
{{{0.0, 0.0, 0.3, 0.3} , {0.0, 0.0, -0.3, -1.2} , {0.0, 0.0, 1.5, 2.1} , {0.0, 0.0, -1.8, -2.1}} ,
{{{0.0, 0.0, 0.0, 0.7} , {0.0, 0.0, 0.0, 0.7} , {0.0, 0.0, 0.0, 0.7} , {0.0, 0.0, 0.0, 0.7}} ,
{{{0.0, 0.0, 0.0, -0.6} , {0.0, 0.0, 0.0, -1.5} , {0.0, 0.0, 0.0, -1.6} , {0.0, 0.0, 0.0, 2.9}} ,
{{{0.0, 0.0, 0.0, 0.2} , {0.0, 0.0, 0.0, 0.2} , {0.0, 0.0, 0.0, 0.2} , {0.0, 0.0, 0.0, 0.2}} ,
{{{0.0, 0.0, 0.0, 0.8} , {0.0, 0.0, 0.0, 2.2} , {0.0, 0.0, 0.0, 2.0} , {0.0, 0.0, 0.0, -3.8}}}} ;

double[, ] dt19yc = {{{0.5, 0.5, 0.5, 0.5} , {0.5, 0.7, 4.0, 3.7} , {0.5, 1.7, -0.5, -0.3} , {0.5, -2.6, -1.5, -1.6}} ,
{{{0.0, 0.0, -0.9, -0.9} , {0.0, 0.0, -6.3, -7.2} , {0.0, 0.0, 0.3, 0.9} , {0.0, 0.0, 3.0, 2.7}} ,
{{{0.0, 0.0, 0.0, 0.3} , {0.0, 0.0, 0.0, 3.0} , {0.0, 0.0, 0.0, -0.7} , {0.0, 0.0, 0.0, -0.7}} ,
{{{0.0, 0.0, 0.0, 0.7} , {0.0, 0.0, 0.0, 1.7} , {0.0, 0.0, 0.0, 1.9} , {0.0, 0.0, 0.0, -3.4}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}} ,
{{{0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0} , {0.0, 0.0, 0.0, 0.0}}}} ;

```

```

double[, ] dt19yd =
    {{(0.5, 0.5, 0.5, 0.5), (0.5, 0.7, 0.7, 0.7), (0.5, 1.7, 1.7, 1.7), (0.5, -2.6, -2.6, -2.6)},
    {{(0.0, 0.0, -0.9, -0.9), (0.0, 0.0, -0.9, -0.9), (0.0, 0.0, -0.9, -0.9), (0.0, 0.0, -0.9, -0.9)},
    {{(0.0, 0.0, 0.3, 0.3), (0.0, 0.0, 1.2, 1.2), (0.0, 0.0, 0.5, 0.5), (0.0, 0.0, -1.3, -1.3)},
    {{(0.0, 0.0, 0.0, 0.7), (0.0, 0.0, 0.0, 0.7), (0.0, 0.0, 0.0, 0.7), (0.0, 0.0, 0.0, 0.7)},
    {{(0.0, 0.0, 0.0, -0.6), (0.0, 0.0, 0.0, -1.5), (0.0, 0.0, 0.0, -1.6), (0.0, 0.0, 0.0, 2.9)},
    {{(0.0, 0.0, 0.0, 0.2), (0.0, 0.0, 0.0, 0.2), (0.0, 0.0, 0.0, 0.2), (0.0, 0.0, 0.0, 0.2)},
    {{(0.0, 0.0, 0.0, 0.8), (0.0, 0.0, 0.0, 1.6), (0.0, 0.0, 0.0, 2.4), (0.0, 0.0, 0.0, -4.0)}};

double[] dtemp = new double[5];
int[] incxs = { 1, 2, -2, -1 };
int[] incys = { 1, -2, 1, -2 };
int[, ] lens = { { 1, 1 }, { 1, 1 }, { 2, 3 }, { 4, 7 } };

int[] ns = { 0, 1, 2, 4 };

/* DOUBLE PRECISION DT10X(7,4,4), DT10Y(7,4,4), DT7(4,4),
$      DT8(7,4,4), DX1(7),
$      DY1(7), SSIZE1(4), SSIZE2(14,2), SSIZE(7),
$      STX(7), STY(7), SX(7), SY(7),
$      DPAR(5,4), DT19X(7,4,16),DT19XA(7,4,4),
$      DT19XB(7,4,4), DT19XC(7,4,4),DT19XD(7,4,4),
$      DT19Y(7,4,16), DT19YA(7,4,4),DT19YB(7,4,4),
$      DT19YC(7,4,4), DT19YD(7,4,4), DTEMP(5)
$
INTEGER
* .. External Functions ..
DOUBLE PRECISION DDOT, DSDOT
EXTERNAL
* .. External Subroutines ..
EXTERNAL DAXPY, DCOPY, DROT, DSWAP, STEST, STEST1
* .. Intrinsic Functions ..
INTRINSIC ABS, MIN
* .. Common blocks ..
COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
* .. Data statements ..
      * 7,4,16 7,4,4
EQUIVALENCE (DT19X(1,1,1),DT19XA(1,1,1)),(DT19X(1,1,5),
A DT19XB(1,1,1)),(DT19X(1,1,9),DT19XC(1,1,1)),
B (DT19X(1,1,13),DT19XD(1,1,1))
EQUIVALENCE (DT19Y(1,1,1),DT19YA(1,1,1)),(DT19Y(1,1,5),
A DT19YB(1,1,1)),(DT19Y(1,1,9),DT19YC(1,1,1)),
B (DT19Y(1,1,13),DT19YD(1,1,1))

(7,4,16)
dt19x(1,1,1) = dt19xa(7,4,4)
(1,1,5) = dt19xb(7,4,4)
(1,1,9) = dt19xc(7,4,4)
(1,1,13) = dt19xd(7,4,4)

*/
// COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
/* .. Data statements ..
EQUIVALENCE (DT19X(1,1,1),DT19XA(1,1,1)),(DT19X(1,1,5),
A DT19XB(1,1,1)),(DT19X(1,1,9),DT19XC(1,1,1)),
B (DT19X(1,1,13),DT19XD(1,1,1))
EQUIVALENCE (DT19Y(1,1,1),DT19YA(1,1,1)),(DT19Y(1,1,5),
A DT19YB(1,1,1)),(DT19Y(1,1,9),DT19YC(1,1,1)),
B (DT19Y(1,1,13),DT19YD(1,1,1))
(7,4,16)
dt19x(1,1,1) = dt19xa(7,4,4)
(1,1,5) = dt19xb(7,4,4)
(1,1,9) = dt19xc(7,4,4)
(1,1,13) = dt19xd(7,4,4) */
// Executable Statements

for (int ii = 1; ii <= 7; ii = ii + 1)
{
    for (int jj = 1; jj <= 4; jj = jj + 1)
    {
        for (int kk = 1; kk <= 4; kk = kk + 1)
        {
            dt19x[ii - 1, jj - 1, kk - 1] = dt19xa[ii - 1, jj - 1, kk - 1];
            dt19x[ii - 1, jj - 1, kk - 1 + 4] = dt19xb[ii - 1, jj - 1, kk - 1];
            dt19x[ii - 1, jj - 1, kk - 1 + 8] = dt19xc[ii - 1, jj - 1, kk - 1];
            dt19x[ii - 1, jj - 1, kk - 1 + 12] = dt19xd[ii - 1, jj - 1, kk - 1];

            dt19y[ii - 1, jj - 1, kk - 1] = dt19ya[ii - 1, jj - 1, kk - 1];
            dt19y[ii - 1, jj - 1, kk - 1 + 4] = dt19yb[ii - 1, jj - 1, kk - 1];
            dt19y[ii - 1, jj - 1, kk - 1 + 8] = dt19yc[ii - 1, jj - 1, kk - 1];
            dt19y[ii - 1, jj - 1, kk - 1 + 12] = dt19yd[ii - 1, jj - 1, kk - 1];
            //Console.WriteLine("{0,3:D}{1,3:D}{2,3:D}{3,12:F3}{4,12:F3}", ii, jj, kk, dt19x[ii - 1, jj - 1, kk
- 1]);
        }
    }
}
//Environment.Exit(1);
for (ki = 1; ki <= 4; ki = ki + 1)
{ //120
    incx = incxs[ki - 1];
    incy = incys[ki - 1];
    mx = Math.Abs(incx);
    my = Math.Abs(incy);

    for (kn = 1; kn <= 4; kn = kn + 1)
    { //100
        n = ns[kn - 1];
        ksize = Math.Min(2, kn);
        lenx = lens[kn - 1, mx - 1];
        leny = lens[kn - 1, my - 1];

        // Initialize all argument arrays
        for (i = 1; i <= 7; i = i + 1)
        { //20
            sx[i - 1] = dx1[i - 1];
            sy[i - 1] = dy1[i - 1];
        }
        if (icase == 1)
        {
            // DDOT
            //STEST1(DDOT(n, sx, incx, sy, incy), dt7[kn-1,ki-1], ssize1[kn-1], sfac);
            //STEST1(DDOT(n, sx, incx, sy, incy), dt7[kn-1,ki-1], ssize1[kn-1], sfac);

```

```

        STEST(1, new double[] { DDOT(n, sx, incx, sy, incy) }, get_darr2d(kn - 1, 3, ki - 1, dt7), get_darr(kn - 1, 3, ssize1), sfac);
    }
    else if (icase == 2)
    {
        // DAXPY
        DAXPY(n, sa, sx, incx, ref sy, incy);

        for (j = 1; j <= leny; j = j + 1)
        { //40
            sty[j - 1] = dt8[j - 1, kn - 1, ki - 1];
        }
        //STEST(leny, sy, sty, ssize2[0, ksize-1], sfac);
        STEST(leny, sy, sty, get_darr2d(0, 13, ksize - 1, ssize2), sfac);
    }
    else if (icase == 5)
    {
        // DCOPY
        for (i = 1; i <= 7; i = i + 1)
        { //60
            sty[i - 1] = dt10y[i - 1, kn - 1, ki - 1];
        }
        DCOPY(n, sx, incx, ref sy, incy);
        //STEST(leny, sy, sty, ssize2[0,0], 1.0);
        STEST(leny, sy, sty, get_darr2d(0, 13, 0, ssize2), 1.0);
    }
    else if (icase == 6)
    {
        // DSWAP
        DSWAP(n, ref sx, incx, ref sy, incy);
        for (i = 1; i <= 7; i = i + 1)
        { //80
            stx[i - 1] = dt10x[i - 1, kn - 1, ki - 1];
            sty[i - 1] = dt10y[i - 1, kn - 1, ki - 1];
        }
        //STEST(lenx, sx, stx, ssize2[0,0], 1.0);
        STEST(lenx, sx, stx, get_darr2d(0, 13, 0, ssize2), 1.0);
        //STEST(leny, sy, sty, ssize2[0,0], 1.0);
        STEST(leny, sy, sty, get_darr2d(0, 13, 0, ssize2), 1.0);
    }
    else if (icase == 12)
    {
        // DROTM
        kni = kn + 4 * (ki - 1);
        for (kpar = 1; kpar <= 4; kpar = kpar + 1)
        {
            for (i = 1; i <= 7; i = i + 1)
            {
                sx[i - 1] = dx1[i - 1];
                sy[i - 1] = dyl[i - 1];
                stx[i - 1] = dt19x[i - 1, kpar - 1, kni - 1];
                sty[i - 1] = dt19y[i - 1, kpar - 1, kni - 1];
            }
            for (i = 1; i <= 5; i = i + 1)
            {
                dtemp[i - 1] = dpar[i - 1, kpar - 1];
            }
            for (i = 1; i <= lenx; i = i + 1)
            {
                ssize[i - 1] = stx[i - 1];
            }
            // SEE REMARK ABOVE ABOUT DT11X(1,2,7) AND DT11X(5,3,8).
            if ((kpar == 2) & (kni == 7))
            {
                ssize[0] = 2.4;
            }
            if ((kpar == 3) & (kni == 8))
            {
                ssize[4] = 1.8;
            }
            // Console.WriteLine("bef");
            //for (i = 1; i <= 7; i = i + 1)
            //{
            //    Console.WriteLine("{0} {1}", sx[i - 1], sy[i - 1]);
            //}
            DROTM(n, ref sx, incx, ref sy, incy, dtemp);
            // Console.WriteLine("bef");
            //for (i = 1; i <= 7; i = i + 1)
            //{
            //    Console.WriteLine("{0} {1}", sx[i - 1], sy[i - 1]);
            //}
            //Environment.Exit(1);
            STEST(lenx, sx, stx, ssize, sfac);

            STEST(leny, sy, sty, sty, sfac);
        }
        //Console.WriteLine("end DROTM");//Environment.Exit(1);
    }
    else if (icase == 13)
    {
        //DSDOT
        // CALL TESTDSDOT( REAL( DSDOT(N,REAL(SX),INCX,REAL(SY),INCY)),
        // REAL( DT7(KN,KI)), REAL(SSIZE1(KN)), .3125E-1)
        float ftmp = (float)DSDOT(n, get_darrf(0, 6, sx), incx, get_darrf(0, 6, sy), incy);
        TESTDSDOT(ftmp, (float)dt7[kn - 1, ki - 1], (float)ssize1[kn - 1], 0.3125e-1f);
    }
    else
    {
        Console.WriteLine("Shouldn't be here in CHECK2");
        Environment.Exit(1);
    }
}
}

static void CHECK3(double sfac)
{
    // Parameters
    const int nout = 6;

    // Local Scalars
    double sc = 0.8, ss = 0.6;

    int i, k, ki, kn, ksize, lenx, leny, mx, my;

    // Local Arrays

```

```

/*DOUBLE PRECISION COPYX(5), COPYY(5), DT9X(7,4,4), DT9Y(7,4,4),
+ DX1(7), DY1(7), MWPC(11), MWPS(11), MWPSTX(5),
+ MWPSTY(5), MWPTX(11,5), MWPTY(11,5), MWPIX(5),
+ MWPLY(5), SSIZE2(14,2), STX(7), STY(7), SX(7),
+ SY(7)
+ INTEGER INCXS(4), INCYS(4), LENS(4,2), MWPINX(11),
+ MWPINY(11), MWPN(11), NS(4)
+ */
// COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
double[] copyx = new double[5];
double[] copyy = new double[5];
double[, ] dt9x = {{{{0.6, 0.6, 0.6, 0.6}, {0.78, 0.78, 0.78, 0.78}, {0.78, 0.66, -0.06, 0.78}, {0.78, 0.96, 0.9, 0.78}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {-0.46, 0.1, 0.1, 0.26}, {-0.46, 0.1, 0.1, 0.26}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, -0.1, -0.1, 0.0}, {-0.22, -0.76, -0.22, -0.76}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {1.06, 0.8, 0.8, 1.12}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.9, 0.18, 0.0}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, -0.3, -0.3, 0.0}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, -0.02, -0.02, 0.0}}}}};

double[, ] dt9y = {{{{0.5, 0.5, 0.5, 0.5}, {0.04, 0.04, 0.04, 0.04}, {0.04, 0.7, 0.7, 0.04}, {0.04, 0.64, 0.64, 0.04}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {-0.78, -0.9, -1.08, -0.9}, {-0.78, -0.9, -1.26, -0.9}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, -0.12, 0.0, 0.18}, {0.54, -0.3, 0.54, 0.18}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.08, 0.7, 0.2, 0.7}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, -0.18, 0.0, -0.18}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.2, 0.0, 0.2}},
{{{0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0}, {0.0, 0.28, 0.0, 0.16}}}}};

double[] dx1 = { 0.6, 0.1, -0.5, 0.8, 0.9, -0.3, -0.4 };
double[] dy1 = { 0.5, -0.9, 0.3, 0.7, -0.6, 0.2, 0.8 };
double[] mwpc = new double[11];
double[] mwps = new double[11];
double[] mwpstx = new double[5];
double[] mwpsty = new double[5];
double[, ] mwptx = new double[11, 5];
double[, ] mwpty = new double[11, 5];
double[] mwpx = new double[5];
double[] mwpy = new double[5];
double[, ] ssize2 = {{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17},
{0.0, 1.17}}};

double[] stx = new double[7];
double[] sty = new double[7];
double[] sx = new double[7];
double[] sy = new double[7];
int[] incxs = { 1, 2, -2, -1 };
int[] incys = { 1, -2, 1, -2 };

int[, ] lens = { { 1, 1 }, { 1, 1 }, { 2, 3 }, { 4, 7 } };
int[] mwpinx = new int[11];
int[] mwpiny = new int[11];
int[] mwpn = new int[11];
int[] ns = { 0, 1, 2, 4 };

// Executable Statements
for (ki = 1; ki <= 4; ki = ki + 1)
{
//60
incx = incxs[ki - 1];
incy = incys[ki - 1];
mx = Math.Abs(incx);
my = Math.Abs(incy);

for (kn = 1; kn <= 4; kn = kn + 1)
{
//40
n = ns[kn - 1];
ksize = Math.Min(2, kn);
lenx = lens[kn - 1, mx - 1];
leny = lens[kn - 1, my - 1];

if (icase == 4)
{
// DROT
for (i = 1; i <= 7; i = i + 1)
{
//20
sx[i - 1] = dx1[i - 1];
sy[i - 1] = dy1[i - 1];
stx[i - 1] = dt9x[i - 1, kn - 1, ki - 1];
sty[i - 1] = dt9y[i - 1, kn - 1, ki - 1];
}
DROT(n, ref sx, incx, ref sy, incy, sc, ss);
//STEST(lenx, sx, stx, ssize2[0, ksize-1], sfac);
STEST(lenx, sx, stx, get_darr2d(0, 13, ksize - 1, ssize2), sfac);
//STEST(leny, sy, sty, ssize2[0, ksize-1], sfac);
STEST(leny, sy, sty, get_darr2d(0, 13, ksize - 1, ssize2), sfac);
}
else
{
Console.WriteLine("Shouldn't be here in CHECK3");
Environment.Exit(1);
}
}
}

mwpc[0] = 1;

for (i = 2; i <= 11; i = i + 1)
{
mwpc[i - 1] = 0;
}

mwps[0] = 0;

for (i = 2; i <= 6; i = i + 1)
{
mwps[i - 1] = 1;
}

for (i = 7; i <= 11; i = i + 1)
{

```

```

        mwps[i - 1] = -1;
    }

    mwpinx[0] = 1;
    mwpinx[1] = 1;
    mwpinx[2] = 1;
    mwpinx[3] = -1;
    mwpinx[4] = 1;
    mwpinx[5] = -1;
    mwpinx[6] = 1;
    mwpinx[7] = 1;
    mwpinx[8] = -1;
    mwpinx[9] = 1;
    mwpinx[10] = -1;

    mwpy[0] = 1;
    mwpy[1] = 1;
    mwpy[2] = -1;
    mwpy[3] = -1;
    mwpy[4] = 2;
    mwpy[5] = 1;
    mwpy[6] = 1;
    mwpy[7] = -1;
    mwpy[8] = -1;
    mwpy[9] = 2;
    mwpy[10] = 1;

    for (i = 1; i <= 11; i = i + 1)
    {
        mwpn[i - 1] = 5;
    }
    mwpn[4] = 3;
    mwpn[9] = 3;

    for (i = 1; i <= 5; i = i + 1)
    {
        mwpx[i - 1] = i;
        mwpy[i - 1] = i;
        mwptx[0, i - 1] = i;
        mwpty[0, i - 1] = i;
        mwptx[1, i - 1] = i;
        mwpty[1, i - 1] = -1;
        mwptx[2, i - 1] = 6 - i;
        mwpty[2, i - 1] = i - 6;
        mwptx[3, i - 1] = i;
        mwpty[3, i - 1] = -i;
        mwptx[5, i - 1] = 6 - i;
        mwpty[5, i - 1] = i - 6;
        mwptx[6, i - 1] = -i;
        mwpty[6, i - 1] = i;
        mwptx[7, i - 1] = i - 6;
        mwpty[7, i - 1] = 6 - i;
        mwptx[8, i - 1] = -i;
        mwpty[8, i - 1] = i;
        mwptx[10, i - 1] = i - 6;
        mwpty[10, i - 1] = 6 - i;
    }
    mwptx[4, 0] = 1;
    mwptx[4, 1] = 3;
    mwptx[4, 2] = 5;
    mwptx[4, 3] = 4;
    mwptx[4, 4] = 5;
    mwpty[4, 0] = -1;
    mwpty[4, 1] = 2;
    mwpty[4, 2] = -2;
    mwpty[4, 3] = 4;
    mwpty[4, 4] = -3;

    mwptx[9, 0] = -1;
    mwptx[9, 1] = -3;
    mwptx[9, 2] = -5;
    mwptx[9, 3] = 4;
    mwptx[9, 4] = 5;
    mwpty[9, 0] = 1;
    mwpty[9, 1] = 2;
    mwpty[9, 2] = 2;
    mwpty[9, 3] = 4;
    mwpty[9, 4] = 3;

    for (i = 1; i <= 11; i = i + 1)
    {
        incx = mwpinx[i - 1];
        incy = mwpy[i - 1];
        for (k = 1; k <= 5; k = k + 1)
        {
            copyx[k - 1] = mwpx[k - 1];
            copyy[k - 1] = mwpy[k - 1];
            mwpstx[k - 1] = mwptx[i - 1, k - 1];
            mwpsty[k - 1] = mwpty[i - 1, k - 1];
        }

        DROT(mwpn[i - 1], ref copyx, incx, ref copyy, incy, mwpc[i - 1], mwps[i - 1]);
        STEST(5, copyx, mwpstx, mwpstx, sfac);
        STEST(5, copyy, mwpsty, mwpsty, sfac);
    }
}

static void STEST(int len, double[] scomp, double[] strue, double[] ssize, double sfac)
{
    // THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
    // NEGLIGIBLE. C. L. LAWSON, JPL, 1974 DEC 10

    // Parameters
    const int nout = 6;
    const double zero = 0.0;

    // Local Scalars
    double sd;
    int i;
    // COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
    /* Console.WriteLine("len: {0}", len);
    Console.WriteLine("scomp: {0}", scomp[0]);
    Console.WriteLine("strue: {0}", strue[0]);
    Console.WriteLine("ssize: {0}", ssize[0]);

```

```

        Console.WriteLine("sfac: {0}", sfac);
        Console.WriteLine("d_eps: {0}", cs_BLAS.misc_d.eps);
        Console.WriteLine("s_eps: {0}", cs_BLAS.misc_s.eps); */
        for (i = 1; i <= len; i = i + 1)
        {
            //Console.WriteLine("{0,3:D}{1,12:F3}{2,12:F3}{3,12:F3}{4,12:F3}{5,3:D}" ,i,scomp[i-1],strue[i-1],ssize[i-1],sfac,len);
        }

        for (i = 1; i <= len; i = i + 1)
        {
            sd = scomp[i - 1] - strue[i - 1];
            // IF (ABS(SFAC*SD) .LE. ABS(SSIZE(I))*EPSILON(ZERO))
            if (Math.Abs(sfac * sd) <= (Math.Abs(ssize[i - 1]) * cs_BLAS.misc_d.eps))
            {
                //epsilon(zero)
                continue;
            }
            // HERE SCOMP(I) IS NOT CLOSE TO STRUE(I).
            if (pass)
            {
                pass = false;
                Console.WriteLine("
                Console.WriteLine("CASE  N INCX INCY  I                                FAIL");
                COMP(I)                                TRUE(I)  DIFFERENCE
SIZE(I)");
            }
            Console.WriteLine("{0,4:D}{1,3:D}{2,5:D}{3,5:D}{4,3:D}{5,36:F8}{6,36:F8}{7,12:F4}{8,12:F4}" , icase, n, incx, incy, i, scomp[i - 1], strue[i
- 1], sd, ssize[i - 1]);
            /*      IF ( .NOT. PASS) GO TO 20
                PRINT FAIL MESSAGE AND HEADER.

                PASS = .FALSE.
                WRITE (NOUT,99999)
                WRITE (NOUT,99998)
                20  WRITE (NOUT,99997) ICASE, N, INCX, INCY, I, SCOMP(I),
                +    STRUE(I), SD, SSIZE(I) */
                //99997 FORMAT (1X,I4,I3,2I5,I3,2D36.8,2D12.4)
            }
        }
        static void TESTSDOT(float scomp, float strue, float ssize, float sfac)
        {
            // THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
            // NEGLIGIBLE. - C. L. LAWSON, JPL, 1974 DEC 10

            // Parameters
            const int nout = 6;
            float zero = 0.0F;

            // Local Scalars
            float sd;
            // COMMON /COMBLA/ICASE, N, INCX, INCY, PASS

            sd = scomp - strue;
            if (Math.Abs(sfac * sd) <= (Math.Abs(ssize) * cs_BLAS.misc_s.eps))
            {
                //epsilon(zero) {
                return;
            }
            // HERE SCOMP(I) IS NOT CLOSE TO STRUE(I).
            if (pass)
            {
                pass = false;
                Console.WriteLine("
                Console.WriteLine("CASE  N INCX INCY                                FAIL");
                COMP(I)                                TRUE(I)  DIFFERENCE  SIZE(I)");
            }
            Console.WriteLine("{0,4:N}{1,3:N}{2,5:N}{3,3:N}{4,36:E8}{5,36:E8}{6,12:E4}{7,12:E4}" , icase, n, incx, incy, scomp, strue, sd, ssize);
            /* 99997 FORMAT (1X,I4,I3,1I5,I3,2E36.8,2E12.4)
                IF ( .NOT. PASS) GO TO 20
                PRINT FAIL MESSAGE AND HEADER.

                PASS = .FALSE.
                WRITE (NOUT,99999)
                WRITE (NOUT,99998)
                20  WRITE (NOUT,99997) ICASE, N, INCX, INCY, SCOMP,
                +    STRUE, SD, SSIZE
            40 CONTINUE
            RETURN */
        }
        static void _STEST1 (double scomp1, double strue1, double[] ssize, double sfac)
        {
            // THIS IS AN INTERFACE SUBROUTINE TO ACCOMODATE THE FORTRAN REQUIREMENT THAT WHEN A DUMMY ARGUMENT IS AN ARRAY, THE
            // ACTUAL ARGUMENT MUST ALSO BE AN ARRAY OR AN ARRAY ELEMENT. -- C.L. LAWSON, JPL, 1978 DEC 6
            // Local Arrays
            double[] scomp = new double[1], strue = new double[1];

            scomp[0] = scomp1;
            strue[0] = strue1;
            //STEST1(sa, datrue[k - 1], datrue[k - 1], sfac);
            STEST(1, scomp, strue, ssize, sfac);
        }
        static double SDIFF(double sa, double sb)
        {
            // COMPUTES DIFFERENCE OF TWO NUMBERS. C. L. LAWSON, JPL 1974 FEB 15
            return (sa - sb);
        }
        static void ITEST1(int icomp, int itrue)
        {
            // THIS SUBROUTINE COMPARES THE VARIABLES ICOMP AND ITRUE FOR EQUALITY. -- C. L. LAWSON, JPL, 1974 DEC 10

            // Parameters
            const int nout = 6;
            // Local Scalars
            int id;
            // COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
            if (icomp == itrue)
            {
                return;
            }
            // HERE ICOMP IS NOT EQUAL TO ITRUE.
            if (pass)
            {
                pass = false;
                Console.WriteLine("
                Console.WriteLine("CASE  N INCX INCY                                FAIL");
                COMP                                TRUE  DIFFERENCE");
            }
            id = icomp - itrue;
            Console.WriteLine("{0,4:N}{1,3:N}{2,5:N}{3,5:N}{4,36:N}{5,36:N}{6,12:N}" , icase, n, incx, incy, icomp, itrue, id);

```

```

//9997 FORMAT (IX,I4,I3,2I5,2I36,I12)
}
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DATA L(1)/' DDOT '/'
! DATA L(2)/'DAXPY '/'
! DATA L(3)/'DROTG '/'
! DATA L(4)/' DROT '/'
! DATA L(5)/'DCOPY '/'
! DATA L(6)/'DSWAP '/'
! DATA L(7)/'DNRM2 '/'
! DATA L(8)/'DASUM '/'
! DATA L(9)/'DSCAL '/'
! DATA L(10)/'IDAMAX'/'
! DATA L(11)/'DROTMG'/'
! DATA L(12)/'DROTM '/'
! DATA L(13)/'DSDOT '/'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
public static double DDOT(int n, double[] dx, int incx, double[] dy, int incy)
{
    // C# version of DDOT
    //
    // DDOT forms the dot product of two vectors. uses unrolled loops for increments equal to one.

    // Local Scalars
    double dtemp;
    int i, ix, iy, m, m1;

    //DDOT = 0.0d0
    dtemp = 0.0;

    if (n <= 0)
    {
        return 0.0;
    }

    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        // clean-up loop
        m = n % 5;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dtemp = dtemp + dx[i - 1] * dy[i - 1];
            }
            if (n < 5)
            {
                return dtemp;
            }
        }
        m1 = m + 1;
        for (i = m1; i <= n; i = i + 5)
        {
            dtemp = dtemp + dx[i - 1] * dy[i - 1] + dx[i] * dy[i] + dx[i + 1] * dy[i + 1] + dx[i + 2] * dy[i + 2] + dx[i + 3] * dy[i + 3];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dtemp = dtemp + dx[ix - 1] * dy[iy - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
    return dtemp;
}

public static void DAXPY(int n, double da, double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DAXPY
    //
    // DAXPY constant times a vector plus a vector. uses unrolled loops for increments equal to one.

    // Local Scalars
    int i, ix, iy, m, m1;

    if (n <= 0)
    {
        return;
    }
    if (da == 0.0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        // clean-up loop
        m = n % 4;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dy[i - 1] = dy[i - 1] + da * dx[i - 1];
            }
        }
        if (n < 4)
        {
            return;
        }
        m1 = m + 1;
        for (i = m1; i <= n; i = i + 4)
    }

```

```

        {
            dy[i - 1] = dy[i - 1] + da * dx[i - 1];
            dy[i] = dy[i] + da * dx[i];
            dy[i + 1] = dy[i + 1] + da * dx[i + 1];
            dy[i + 2] = dy[i + 2] + da * dx[i + 2];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dy[iy - 1] = dy[iy - 1] + da * dx[ix - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}

public static void DROTG(ref double da, ref double db, ref double c, ref double s)
{
    // C# version of DROTG
    //
    // DROTG construct gives plane rotation.

    // Local Scalars
    double r, roe, scale, z;

    roe = db;
    if (Math.Abs(da) > Math.Abs(db))
    {
        roe = da;
    }
    scale = Math.Abs(da) + Math.Abs(db);
    if (scale == 0.0)
    {
        c = 1.0;
        s = 0.0;
        r = 0.0;
        z = 0.0;
    }
    else
    {
        r = scale * Math.Sqrt(Math.Pow(da / scale, 2) + Math.Pow(db / scale, 2));
        //condition ? first_expression : second_expression
        r = (roe >= 0.0 ? 1.0 : -1.0) * r;
        c = da / r;
        s = db / r;
        z = 1.0;
        if (Math.Abs(da) > Math.Abs(db))
        {
            z = s;
        }
        if ((Math.Abs(db) >= Math.Abs(da)) & (c != 0.0))
        {
            z = 1.0 / c;
        }
    }
    da = r;
    db = z;
}

public static void DROT(int n, ref double[] dx, int incx, ref double[] dy, int incy, double c, double s)
{
    // C# version of DROT
    //
    // DROT applies a plane rotation

    // Local Scalars
    double dtemp;
    int i, ix, iy;

    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        for (i = 1; i <= n; i = i + 1)
        {
            dtemp = c * dx[i - 1] + s * dy[i - 1];
            dy[i - 1] = c * dy[i - 1] - s * dx[i - 1];
            dx[i - 1] = dtemp;
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dtemp = c * dx[ix - 1] + s * dy[iy - 1];
            dy[iy - 1] = c * dy[iy - 1] - s * dx[ix - 1];
            dx[ix - 1] = dtemp;
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}

```

```

    }
}
}
public static void DCOPY(int n, double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DCOPY
    //
    // DCOPY copies a vector, x, to a vector, y uses unrolled loops for increments equal to one

    // Local Scalars
    int i, ix, iy, m, mpl;
    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        // clean-up loop
        m = n % 7;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dy[i - 1] = dx[i - 1];
            }
            if (n < 7)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 7)
        {
            dy[i - 1] = dx[i - 1];
            dy[i] = dx[i];
            dy[i + 1] = dx[i + 1];
            dy[i + 2] = dx[i + 2];
            dy[i + 3] = dx[i + 3];
            dy[i + 4] = dx[i + 4];
            dy[i + 5] = dx[i + 5];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dy[iy - 1] = dx[ix - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}

public static void DSWAP(int n, ref double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DSWAP
    //
    // interchanges two vectors. uses unrolled loops for increments equal one.

    // Local Scalars
    double dtemp;
    int i, ix, iy, m, mpl;

    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        //clean-up loop
        m = n % 3;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dtemp = dx[i - 1];
                dx[i - 1] = dy[i - 1];
                dy[i - 1] = dtemp;
            }
            if (n < 3)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 3)
        {
            dtemp = dx[i - 1];
            dx[i - 1] = dy[i - 1];
            dy[i - 1] = dtemp;

            dtemp = dx[i];
            dx[i] = dy[i];
            dy[i] = dtemp;

            dtemp = dx[i + 1];
            dx[i + 1] = dy[i + 1];
            dy[i + 1] = dtemp;
        }
    }
    else

```

```

{
    // code for unequal increments or equal increments not equal to 1
    ix = 1;
    iy = 1;
    if (incx < 0)
    {
        ix = (-n + 1) * incx + 1;
    }
    if (incy < 0)
    {
        iy = (-n + 1) * incy + 1;
    }
    for (i = 1; i <= n; i = i + 1)
    {
        dtemp = dx[ix - 1];
        dx[ix - 1] = dy[iy - 1];
        dy[iy - 1] = dtemp;
        ix = ix + incx;
        iy = iy + incy;
    }
}
}
public static double DNRM2(int n, double[] x, int incx)
{
    // C# version of DNRM2
    //
    // DNRM2 returns the euclidean norm of a vector via the function name, so that DNRM2 := sqrt( x'*x )

    // Parameters
    const double one = 1.0, zero = 0.0;

    // Local Scalars
    double absxi, norm, scale, ssq;
    int ix;

    if ((n < 1) | (incx < 1))
    {
        return zero;
    }
    else if (n == 1)
    {
        return Math.Abs(x[0]);
    }
    else
    {
        scale = zero;
        ssq = one;
        // The following loop is equivalent to this call to the LAPACK auxiliary routine:
        // CALL DLAASSQ( N, X, INCX, SCALE, SSQ )

        for (ix = 1; ix <= 1 + (n - 1) * incx; ix = ix + incx)
        {
            if (x[ix - 1] != zero)
            {
                absxi = Math.Abs(x[ix - 1]);
                if (scale < absxi)
                {
                    ssq = one + ssq * Math.Pow(scale / absxi, 2);
                    scale = absxi;
                }
                else
                {
                    ssq = ssq + Math.Pow(absxi / scale, 2);
                }
            }
        }
        return scale * Math.Sqrt(ssq);
    }
}
public static double DASUM(int n, double[] dx, int incx)
{
    // C# version of DASUM
    //
    // DASUM takes the sum of the absolute values

    // Local Scalars
    double dtemp;
    int i, m, m1, nincx;

    //DASUM = 0.0;
    dtemp = 0.0;
    if ((n <= 0) | (incx <= 0))
    {
        return 0.0;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        // clean-up loop

        m = n % 6;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dtemp = dtemp + Math.Abs(dx[i - 1]);
            }
            if (n < 6)
            {
                return dtemp;
            }
        }
        m1 = m + 1;
        for (i = m1; i <= n; i = i + 6)
        {
            dtemp = dtemp + Math.Abs(dx[i - 1]) + Math.Abs(dx[i]) + Math.Abs(dx[i + 1]) + Math.Abs(dx[i + 2]) + Math.Abs(dx[i + 3]) + Math.Abs(dx[i + 4]) + Math.Abs(dx[i + 5]);
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)

```

```

        {
            dtemp = dtemp + Math.Abs(dx[i - 1]);
        }
    }
    return dtemp;
}
public static void DSCAL(int n, double da, ref double[] dx, int incx)
{
    // C# version of DSCAL
    //
    // DSCAL scales a vector by a constant. uses unrolled loops for increment equal to one.
    //
    // Local Scalars
    int i, m, mpl, nincx;
    if ((n <= 0) | (incx <= 0))
    {
        return;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        // clean-up loop
        m = n % 5;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dx[i - 1] = da * dx[i - 1];
            }
            if (n < 5)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 5)
        {
            dx[i - 1] = da * dx[i - 1];
            dx[i] = da * dx[i];
            dx[i + 1] = da * dx[i + 1];
            dx[i + 2] = da * dx[i + 2];
            dx[i + 3] = da * dx[i + 3];
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            dx[i - 1] = da * dx[i - 1];
        }
    }
}
public static int IDAMAX(int n, double[] dx, int incx)
{
    // C# version of IDAMAX
    //
    // IDAMAX finds the index of element having max. absolute value.
    //
    // Local Scalars
    double dmax;
    int i, ix, idamax;
    idamax = 0;
    if ((n < 1) | (incx <= 0))
    {
        return idamax;
    }
    idamax = 1;
    if (n == 1)
    {
        return idamax;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        dmax = Math.Abs(dx[0]);
        for (i = 2; i <= n; i = i + 1)
        {
            if (Math.Abs(dx[i - 1]) > dmax)
            {
                idamax = i;
                dmax = Math.Abs(dx[i - 1]);
            }
        }
    }
    else
    {
        // code for increment not equal to 1
        ix = 1;
        dmax = Math.Abs(dx[0]);
        ix = ix + incx;
        for (i = 2; i <= n; i = i + 1)
        {
            if (Math.Abs(dx[ix - 1]) > dmax)
            {
                idamax = i;
                dmax = Math.Abs(dx[ix - 1]);
            }
            ix = ix + incx;
        }
    }
    return idamax;
}
public static void DROTMG(ref double dd1, ref double dd2, ref double dx1, ref double dyl, ref double[] dparam)
{
    // C# version of DROTMG
    //
    // CONSTRUCT THE MODIFIED GIVENS TRANSFORMATION MATRIX H WHICH ZEROS
    // THE SECOND COMPONENT OF THE 2-VECTOR (DSQRT(DD1)*DX1,DSQRT(DD2))*> DY2)**T.
    // WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..

```

```

// DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0

//      (DH11 DH12)      (1.D0 DH12)      (DH11 1.D0)      (1.D0 0.D0)
// H=(      )      (      )      (      )      (      )
//      (DH21 DH22),      (DH21 1.D0),      (-1.D0 DH22),      (0.D0 1.D0).

// LOCATIONS 2-4 OF DPARAM CONTAIN DH11, DH21, DH12, AND DH22
// RESPECTIVELY. (VALUES OF 1.D0, -1.D0, OR 0.D0 IMPLIED BY THE
// VALUE OF DPARAM(1) ARE NOT STORED IN DPARAM.)

// THE VALUES OF GAMSQ AND RGAMSQ SET IN THE DATA STATEMENT MAY BE
// INEXACT. THIS IS OK AS THEY ARE ONLY USED FOR TESTING THE SIZE
// OF DD1 AND DD2. ALL ACTUAL SCALING OF DATA IS DONE USING GAM.

// DPARAM(1)=DFLAG
// DPARAM(2)=DH11
// DPARAM(3)=DH21
// DPARAM(4)=DH12
// DPARAM(5)=DH22

// Local Scalars
double dflag = 0.0, dh11 = 0.0, dh12 = 0.0, dh21 = 0.0, dh22 = 0.0, dp1, dp2, dq1, dq2, dtemp;
double du, gam = 4096.0, gamsq = 16777216.0, one = 1.0, rgamsq = 5.9604645e-8, two = 2.0, zero = 0.0;

if (ddl < zero)
{
    // GO ZERO-H-D-AND-DX1
    dflag = -one;
    dh11 = zero;
    dh12 = zero;
    dh21 = zero;
    dh22 = zero;

    ddl = zero;
    dd2 = zero;
    dx1 = zero;
}
else
{
    // CASE-DD1-NONNEGATIVE
    dp2 = dd2 * dyl;
    if (dp2 == zero)
    {
        dflag = -two;
        dparam[0] = dflag;
        return;
    }
    // REGULAR-CASE
    dp1 = dd1 * dx1;
    dq2 = dp2 * dyl;
    dq1 = dp1 * dx1;
    if (Math.Abs(dq1) > Math.Abs(dq2))
    {
        dh21 = -dyl / dx1;
        dh12 = dp2 / dp1;

        du = one - dh12 * dh21;

        if (du > zero)
        {
            dflag = zero;
            dd1 = dd1 / du;
            dd2 = dd2 / du;
            dx1 = dx1 * du;
        }
    }
    else
    {
        if (dq2 < zero)
        {
            // GO ZERO-H-D-AND-DX1
            dflag = -one;
            dh11 = zero;
            dh12 = zero;
            dh21 = zero;
            dh22 = zero;

            ddl = zero;
            dd2 = zero;
            dx1 = zero;
        }
        else
        {
            dflag = one;
            dh11 = dp1 / dp2;
            dh22 = dx1 / dyl;
            du = one + dh11 * dh22;
            dtemp = dd2 / du;
            dd2 = dd1 / du;
            dd1 = dtemp;
            dx1 = dyl * du;
        }
    }
    // PROCEDURE..SCALE-CHECK
    if (ddl != zero)
    {
        while ((ddl <= rgamsq) | (ddl >= gamsq))
        {
            if (dflag == zero)
            {
                dh11 = one;
                dh22 = one;
                dflag = -one;
            }
            else
            {
                dh21 = -one;
                dh12 = one;
                dflag = -one;
            }
            if (ddl <= rgamsq)
            {
                ddl = dd1 * Math.Pow(gam, 2);
            }
        }
    }
}

```

```

        dx1 = dx1 / gam;
        dh11 = dh11 / gam;
        dh12 = dh12 / gam;
    }
    else
    {
        ddl = ddl / Math.Pow(gam, 2);
        dx1 = dx1 * gam;
        dh11 = dh11 * gam;
        dh12 = dh12 * gam;
    }
}
}
if (dd2 != zero)
{
    while ((Math.Abs(dd2) <= rgamsq) | (Math.Abs(dd2) >= gamsq))
    {
        if (dflag == zero)
        {
            dh11 = one;
            dh22 = one;
            dflag = -one;
        }
        else
        {
            dh21 = -one;
            dh12 = one;
            dflag = -one;
        }
        if (Math.Abs(dd2) <= rgamsq)
        {
            dd2 = dd2 * Math.Pow(gam, 2);
            dh21 = dh21 / gam;
            dh22 = dh22 / gam;
        }
        else
        {
            dd2 = dd2 / Math.Pow(gam, 2);
            dh21 = dh21 * gam;
            dh22 = dh22 * gam;
        }
    }
}
if (dflag < zero)
{
    dparam[1] = dh11;
    dparam[2] = dh21;
    dparam[3] = dh12;
    dparam[4] = dh22;
}
else if (dflag == zero)
{
    dparam[2] = dh21;
    dparam[3] = dh12;
}
else
{
    dparam[1] = dh11;
    dparam[4] = dh22;
}
dparam[0] = dflag;
}
public static void DROTM(int n, ref double[] dx, int incx, ref double[] dy, int incy, double[] dparam)
{
    // C# version of DROTM
    //
    // APPLY THE MODIFIED GIVENS TRANSFORMATION, H, TO THE 2 BY N MATRIX
    //
    // (DX**T) , WHERE **T INDICATES TRANSPOSE. THE ELEMENTS OF DX ARE IN
    // (DY**T)
    //
    // DX(LX+I*INCX), I = 0 TO N-1, WHERE LX = 1 IF INCX .GE. 0, ELSE
    // LX = (-INCX)*N, AND SIMILARLY FOR SY USING LY AND INCY.
    // WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..
    //
    // DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0
    //
    // (DH11 DH12)      (1.D0 DH12)      (DH11 1.D0)      (1.D0 0.D0)
    // H= ( )            ( )            ( )            ( )
    //      (DH21 DH22), (DH21 1.D0), (-1.D0 DH22), (0.D0 1.D0).
    // SEE DROTMG FOR A DESCRIPTION OF DATA STORAGE IN DPARAM.
    //
    // DPARAM(1)=DFLAG
    // DPARAM(2)=DH11
    // DPARAM(3)=DH21
    // DPARAM(4)=DH12
    // DPARAM(5)=DH22
    //
    // Local Scalars
    double dflag, dh11, dh12, dh21, dh22, two = 2.0, w, z, zero = 0.0;
    int i, kx, ky, nsteps;

    dflag = dparam[0];
    if ((n <= 0) | (dflag + two == zero))
    {
        return;
    }
    if ((incx == incy) & (incx > 0))
    {
        nsteps = n * incx;
        if (dflag < zero)
        {
            dh11 = dparam[1];
            dh12 = dparam[3];
            dh21 = dparam[2];
            dh22 = dparam[4];
            for (i = 1; i <= nsteps; i = i + incx)
            {
                w = dx[i - 1];
                z = dy[i - 1];
                dx[i - 1] = w * dh11 + z * dh12;
                dy[i - 1] = w * dh21 + z * dh22;
            }
        }
    }
}

```

```

    }
    else if (dflag == zero)
    {
        dh12 = dparam[3];
        dh21 = dparam[2];
        for (i = 1; i <= nsteps; i = i + incx)
        {
            w = dx[i - 1];
            z = dy[i - 1];
            dx[i - 1] = w + z * dh12;
            dy[i - 1] = w * dh21 + z;
        }
    }
    else
    {
        dh11 = dparam[1];
        dh22 = dparam[4];
        for (i = 1; i <= nsteps; i = i + incx)
        {
            w = dx[i - 1];
            z = dy[i - 1];
            dx[i - 1] = w * dh11 + z;
            dy[i - 1] = -w + dh22 * z;
        }
    }
}
else
{
    kx = 1;
    ky = 1;
    if (incx < 0)
    {
        kx = 1 + (1 - n) * incx;
    }
    if (incy < 0)
    {
        ky = 1 + (1 - n) * incy;
    }
    if (dflag < zero)
    {
        dh11 = dparam[1];
        dh12 = dparam[3];
        dh21 = dparam[2];
        dh22 = dparam[4];
        for (i = 1; i <= n; i = i + 1)
        {
            w = dx[kx - 1];
            z = dy[ky - 1];
            dx[kx - 1] = w * dh11 + z * dh12;
            dy[ky - 1] = w * dh21 + z * dh22;
            kx = kx + incx;
            ky = ky + incy;
        }
    }
    else if (dflag == zero)
    {
        dh12 = dparam[3];
        dh21 = dparam[2];
        for (i = 1; i <= n; i = i + 1)
        {
            w = dx[kx - 1];
            z = dy[ky - 1];
            dx[kx - 1] = w + z * dh12;
            dy[ky - 1] = w * dh21 + z;
            kx = kx + incx;
            ky = ky + incy;
        }
    }
    else
    {
        dh11 = dparam[1];
        dh22 = dparam[4];
        for (i = 1; i <= n; i = i + 1)
        {
            w = dx[kx - 1];
            z = dy[ky - 1];
            dx[kx - 1] = w * dh11 + z;
            dy[ky - 1] = -w + dh22 * z;
            kx = kx + incx;
            ky = ky + incy;
        }
    }
}
}
public static double DSDOT(int n, float[] sx, int incx, float[] sy, int incy)
{
    // C# version of DSDOT
    //
    // Compute the inner product of two vectors with extended precision accumulation and result.
    //
    // Returns D.P. dot product accumulated in D.P., for S.P. SX and SY
    // DSDOT = sum for I = 0 to N-1 of SX(LX+I*INCX) * SY(LY+I*INCY),
    // where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
    // defined in a similar way using INCY.
    //
    // Local Scalars
    int i, kx, ky, ns;
    double dsdot;

    dsdot = 0.0;
    if (n <= 0)
    {
        return dsdot;
    }
    if ((incx == incy) & (incx > 0))
    {
        // Code for equal, positive, non-unit increments.
        ns = n * incx;
        for (i = 1; i <= ns; i = i + incx)
        {
            dsdot = dsdot + (double)sx[i - 1] * (double)sy[i - 1];
        }
    }
    else

```

```

    {
        // Code for unequal or nonpositive increments.
        kx = 1;
        ky = 1;
        if (incx < 0)
        {
            kx = 1 + (1 - n) * incx;
        }
        if (incy < 0)
        {
            ky = 1 + (1 - n) * incy;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dsdot = dsdot + (double)sx[kx - 1] * (double)sy[ky - 1];
            kx = kx + incx;
            ky = ky + incy;
        }
    }
    return dsdot;
}

public static void _NOOP()
{
}

public static void DBLAT2() // uses misc_d.eps, call DLAMCH first
{
    // Test program for the DOUBLE PRECISION Level 2 Blas.

    /*
The program must be driven by a short data file. The first 18 records
of the file are read using list-directed input, the last 16 records
are read using the format ( A6, L2 ). An annotated example of a data
file can be obtained by deleting the first 3 characters from the
following 34 lines:
'dblat2.out'      NAME OF SUMMARY OUTPUT FILE
6                UNIT NUMBER OF SUMMARY FILE
'DBLAT2.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
-1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F               LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F               LOGICAL FLAG, T TO STOP ON FAILURES.
T               LOGICAL FLAG, T TO TEST ERROR EXITS.
16.0            THRESHOLD VALUE OF TEST RATIO
6              NUMBER OF VALUES OF N
0 1 2 3 5 9     VALUES OF N
4              NUMBER OF VALUES OF K
0 1 2 4         VALUES OF K
4              NUMBER OF VALUES OF INCX AND INCY
1 2 -1 -2       VALUES OF INCX AND INCY
3              NUMBER OF VALUES OF ALPHA
0.0 1.0 0.7     VALUES OF ALPHA
3              NUMBER OF VALUES OF BETA
0.0 1.0 0.9     VALUES OF BETAC
DGEMV  T PUT F FOR NO TEST. SAME COLUMNS.
DGBMV  T PUT F FOR NO TEST. SAME COLUMNS.
DSYMV  T PUT F FOR NO TEST. SAME COLUMNS.
DSBMV  T PUT F FOR NO TEST. SAME COLUMNS.
DSPMV  T PUT F FOR NO TEST. SAME COLUMNS.
DTRMV  T PUT F FOR NO TEST. SAME COLUMNS.
DTBMV  T PUT F FOR NO TEST. SAME COLUMNS.
DTPMV  T PUT F FOR NO TEST. SAME COLUMNS.
DTRSV  T PUT F FOR NO TEST. SAME COLUMNS.
DTBSV  T PUT F FOR NO TEST. SAME COLUMNS.
DTPSV  T PUT F FOR NO TEST. SAME COLUMNS.
DGER   T PUT F FOR NO TEST. SAME COLUMNS.
DSYR   T PUT F FOR NO TEST. SAME COLUMNS.
DSPR   T PUT F FOR NO TEST. SAME COLUMNS.
DSYR2  T PUT F FOR NO TEST. SAME COLUMNS.
DSPR2  T PUT F FOR NO TEST. SAME COLUMNS.
*/

    // Parameters
    //const int nin = 5;
    const int nsubs = 16; // snames.Length
    const double zero = 0.0;
    const double one = 1.0;
    const int nmax = 65;
    const int incmax = 2;
    const int ninmax = 7;
    const int nidmax = 9;
    const int nkmax = 7;
    const int nalmax = 7;
    const int nbemax = 7;

    // Local Scalars
    double eps, err = 0.0, thresh;
    //int i, isnum, j, n, nalf, nbet, nidim, ninc, nkb, nout, ntra;

    int n, nalf, nbet, nidim, ninc, nkb, nout, ntra;
    bool fatal = false, ltestt, rewi, same, sfatal, trace, tsterr;
    String trans, snamet, snaps, summary;

    /*
CHARACTER*1      TRANS
CHARACTER*6      SNAMET
CHARACTER*32     SNAPS, SUMMARY*/

    // Local Arrays

    double[,] a = new double[nmax, nmax];
    double[,] aa = new double[nmax * nmax];
    double[] alf = new double[nalmax];
    double[] _as = new double[nmax * nmax];
    double[] _bet = new double[nbemax];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax * incmax];
    double[] xx = new double[nmax * incmax];
    double[] y = new double[nmax];
    double[] ys = new double[nmax * incmax];
    double[] yt = new double[nmax];
    double[] yy = new double[nmax * incmax];
    double[] z = new double[2 * nmax];

    int[] idim = new int[nidmax];

```

```

int[] inc = new int[ninmax];
int[] kb = new int[nkbmax];

bool[] ltest = new bool[nsubs];
//string[] snames = new string[nsubs];
string[] snames = {
    "DGEMV ",
    "DGBMV ",
    "DSYBV ",
    "DSBMV ",
    "DSPMV ",
    "DTRMV ",
    "DTBMV ",
    "DTPMV ",
    "DTRSV ",
    "DTBSV ",
    "DTFSV ",
    "DGER ",
    "DSYR ",
    "DSPR ",
    "DSYR2 ",
    "DSPR2 "};

// file I/O
//String fin = @"..\dlat2.in";
String fin = @"STDIN";
ArrayList tmp = new ArrayList();
String[] findata = { };
StreamWriter fout1, fout2;

/*.. External Functions ..
DOUBLE PRECISION  DDIFF
LOGICAL          LDE
EXTERNAL          DDIFF, LDE */

// Executable Statements

// read input file
try
{
    tmp.Clear();
    //using (StreamReader infile = new StreamReader(fin))
    using (StreamReader infile = new StreamReader(Console.OpenStandardInput()))
    {
        string line;
        int i = 0;
        while ((line = infile.ReadLine()) != null)
        {
            tmp.Add(line.Trim());
            Console.WriteLine("Read:{0}", line.Trim());
        }
        findata = (String[])tmp.ToArray(typeof(string));
    }
}
catch (Exception e)
{
    Console.WriteLine("Error Reading {0}", fin);
    Environment.Exit(1);
}

// Read name and unit number for summary output file and open file.
summary = findata[0].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
nout = int.Parse(findata[1].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

fout1 = new StreamWriter(summary, false); // true = append
misc.d.nout = fout1; // for XERBLA
fout2 = null;
noutc = nout;

// Read name and unit number for snapshot output file and open file.
snaps = findata[2].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
ntra = int.Parse(findata[3].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

trace = (ntra >= 0);
//Console.WriteLine("ntra: {0}, trace: {1}", ntra, trace);
if (trace)
{
    fout2 = new StreamWriter(snaps, true); // true = append
}

// Read the flag that directs rewinding of the snapshot file.
//Console.WriteLine("4052: {0}", findata[4]);
//Console.WriteLine("4052: {0}", findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
//rewi = bool.Parse(findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    rewi = true;
} else {
    rewi = false;
}
//Console.WriteLine("4052: {0}", rewi);
rewi = rewi & trace;

// Read the flag that directs stopping on any failure.
//sfatal = bool.Parse(findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    sfatal = true;
} else {
    sfatal = false;
}

// Read the flag that indicates whether error exits are to be tested.
//tsterr = bool.Parse(findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    tsterr = true;
} else {
    tsterr = false;
}

// Read the threshold value of the test ratio
thresh = double.Parse(findata[7].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

// Read and check the parameter values for the tests.

```

```

// Values of N
nidim = int.Parse(findata[8].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

if ((nidim < 1) || nidim > nidmax)
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "N", nidmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4095:");
    Environment.Exit(1);
}

for (int i = 1; i <= nidim; i = i + 1)
{
    idim[i - 1] = int.Parse(findata[9].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((idim[i - 1] < 0) || (idim[i - 1] > nmax))
    {
        fout1.WriteLine("VALUE OF N IS LESS THAN 0 OR GREATER THAN {0,2:D}", nmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4111:");
        Environment.Exit(1);
    }
}

// Values of K
nkb = int.Parse(findata[10].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nkb < 1) || (nkb > nkbmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "K", nkbmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4127:");
    Environment.Exit(1);
}

for (int i = 1; i <= nkb; i = i + 1)
{
    kb[i - 1] = int.Parse(findata[11].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if (kb[i - 1] < 0)
    {
        fout1.WriteLine("VALUE OF K IS LESS THAN 0");
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4143:");
        Environment.Exit(1);
    }
}

// Values of INCX and INCY
ninc = int.Parse(findata[12].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((ninc < 1) || (ninc > ninmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "INCX AND INCY", ninmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4159:");
    Environment.Exit(1);
}

for (int i = 1; i <= ninc; i = i + 1)
{
    inc[i - 1] = int.Parse(findata[13].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((inc[i - 1] == 0) || Math.Abs(inc[i - 1]) > incmax)
    {
        fout1.WriteLine("ABSOLUTE VALUE OF INCX OR INCY IS 0 OR GREATER THAN {0,2:D}", incmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4175:");
        Environment.Exit(1);
    }
}

// Values of ALPHA
nalf = int.Parse(findata[14].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nalf < 1) || (nalf > nalmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "ALPHA", nalmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4191:");
    Environment.Exit(1);
}

for (int i = 1; i <= nalf; i = i + 1)
{
    alf[i - 1] = double.Parse(findata[15].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
}

```

```

}

// Values of BETA
nbt = int.Parse(findata[16].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nbt < 1) || (nbt > nbemax))
{
    foutl.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "BETA", nbemax);
    foutl.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        foutl.Flush();
        foutl.Close();
    }
    Console.WriteLine("4210:");
    Environment.Exit(1);
}

for (int i = 1; i <= nbt; i = i + 1)
{
    bet[i - 1] = double.Parse(findata[17].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
}

// Report values of parameters
foutl.WriteLine("TESTS OF THE DOUBLE PRECISION LEVEL 2 BLAS \r\n\r\n THE FOLLOWING PARAMETER VALUES WILL BE USED:");
foutl.Write(" FOR N ");
for (int i = 1; i <= nidim; i = i + 1)
{
    foutl.Write("{0,6:D}", idim[i - 1]);
}
foutl.WriteLine();

foutl.Write(" FOR K ");
for (int i = 1; i <= nkb; i = i + 1)
{
    foutl.Write("{0,6:D}", kb[i - 1]);
}
foutl.WriteLine();

foutl.Write(" FOR INCX AND INCY ");
for (int i = 1; i <= ninc; i = i + 1)
{
    foutl.Write("{0,6:D}", inc[i - 1]);
}
foutl.WriteLine();

foutl.Write(" FOR ALPHA ");
for (int i = 1; i <= nalf; i = i + 1)
{
    foutl.Write("{0,6:F1}", alf[i - 1]);
}
foutl.WriteLine();

foutl.Write(" FOR BETA ");
for (int i = 1; i <= nbt; i = i + 1)
{
    foutl.Write("{0,6:F1}", bet[i - 1]);
}
foutl.WriteLine();
if (!tsterr)
{
    foutl.WriteLine();
    foutl.WriteLine("ERROR-EXITS WILL NOT BE TESTED");
}

foutl.WriteLine();
foutl.WriteLine("ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN {0,8:F2}", thresh);
foutl.WriteLine();

// Read names of subroutines and flags which indicate whether they are to be tested.
for (int i = 1; i <= nsubs; i = i + 1)
{
    ltest[i - 1] = false;
}

for (int i = 1; i <= nsubs; i = i + 1)
{
    snamet = findata[18 + i - 1].Substring(0, 6);
    //ltestt = bool.Parse(findata[18 + i - 1].Substring(6, 2).Trim());
    if (findata[18 + i - 1].Substring(6, 2).Trim().ToUpper() == "T") {
        ltestt = true;
    } else {
        ltestt = false;
    }
    if (snamet == snames[i - 1])
    {
        ltest[i - 1] = ltestt;
    }
    else
    {
        foutl.WriteLine("SUBPROGRAM NAME {0} NOT RECOGNIZED\n***** TESTS ABANDONED *****", snamet);
        foutl.Flush();
        foutl.Close();
        Console.WriteLine("4276:");
        Environment.Exit(1);
    }
}

//foutl.Flush();
//foutl.Close();
//Console.WriteLine("4282:");
// Compute EPS (the machine precision).
// uses misc_d.eps, call DLAMCH first
//eps = misc_d.eps;
eps = misc_d.prec;
foutl.WriteLine("RELATIVE MACHINE PRECISION IS TAKEN TO BE {0,9:E1}", eps);
//Console.WriteLine("{0} {1} {0,9:E1}", eps, eps*10, eps*10);
//Console.WriteLine("4295:");
// Check the reliability of DMVCH using exact data.
n = Math.Min(32, nmax);
for (int j = 1; j <= n; j = j + 1)
{
    for (int i = 1; i <= n; i = i + 1)
    {
        a[i - 1, j - 1] = Math.Max(i - j + 1, 0);
    }
    x[j - 1] = j;
    y[j - 1] = zero;
}

```

```

for (int j = 1; j <= n; j = j + 1)
{
    yy[j - 1] = j * ((j + 1) * j) / 2 - ((j + 1) * j * (j - 1)) / 3;
}
// YY holds the exact result. On exit from DMVCH YT holds the result computed by DMVCH
trans = "N";
//DMVCH(trans, n, n, one, a, nmax, x, 1, zero, y, 1, yt, g, yy, eps, err, fatal, nout, true);
DMVCH(trans, n, n, one, a, nmax, x, 1, zero, y, 1, ref yt, ref g, yy, eps, ref err, ref fatal, foutl, true);

//static double[,] get_2dfromld(int l, int lda, double[] arr)
//static void DMVCH(string trans, int m, int n, double alpha, double[,] a, int nmax, double[] x, int indx, double beta, double[] y, int incy,
double[] yt,
//double[] g, double[] yy, double eps, double err, bool fatal, StreamWriter nout, bool mv
//Console.WriteLine("4319:");
same = LDE(yy, yt, n);
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMVCH WAS CALLED WITH TRANS = {0} AND RETURNED SAME = {1} AND ERR = {2,12:F3}", trans, same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
trans = "T";
//DMVCH(trans, n, n, one, a, nmax, x, -1, zero, y, -1, yt, g, yy, eps, err, fatal, nout, true);
//foutl.WriteLine("2nd DMVCH");
DMVCH(trans, n, n, one, a, nmax, x, -1, zero, y, -1, ref yt, ref g, yy, eps, ref err, ref fatal, foutl, true);
//noutl.Flush();
//noutl.close();
//Environment.Exit(1);
same = LDE(yy, yt, n);
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMVCH WAS CALLED WITH TRANS = {0} AND RETURNED SAME = {1} AND ERR = {2,12:F3}", trans, same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
//Console.WriteLine("4341:");
// Test each subroutine in turn
for (int isnum = 1; isnum <= nsubs; isnum = isnum + 1)
{
    //foutl.WriteLine("***SNAME={0} {1}", snames[isnum - 1], isnum);
    foutl.WriteLine();
    if (!test[isnum - 1])
    {
        // Subprogram is not to be tested
        foutl.WriteLine(" {0,6} WAS NOT TESTED", snames[isnum - 1]);
    }
    else {
        srnamt = snames[isnum - 1];
        // Test error exits
        if (tsterr)
        {
            DCHKE(isnum, snames[isnum - 1], foutl);
            foutl.WriteLine();
        }
        // Test computations
        infot = 0;
        ok = true;
        fatal = false;
        //foutl.WriteLine("^^^isnum={0}", isnum);

        switch (isnum)
        //switch (isnum+50)
        {
            case 1:
            case 2:
                // Test DGEMV, 01, and DGBMV, 02
                DCHK1(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, nalf, alf, nbet, bet, ninc,
inc, nmax,
                    incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g);
                //foutl.Flush();
                //foutl.Close();
                //Environment.Exit(1);

                break;
            case 3:
            case 4:
            case 5:
                // Test DSYMV, 03, DSBMV, 04, and DSPMV, 05
                DCHK2(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, nalf, alf, nbet, bet, ninc,
inc, nmax,
                    incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g);
                break;
            case 6:
            case 7:
            case 8:
            case 9:
            case 10:
            case 11:
                // Test DTRMV, 06, DTBMV, 07, DTPMV, 08, DTRSV, 09, DTBSV, 10, and DTPSV, 11
                DCHK3(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, ninc, inc, nmax,
                    incmax, ref a, ref aa, _as, ref y, ref yy, ys, yt, g, z);
                break;
            case 12:
                // Test DGER, 12
                DCHK4(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,
                    incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, z);
                break;
            case 13:
            case 14:
                //*****
                // if not ref a, aa, x, xx, y, yy then aa not same for DCHK5
                //
                // Test DSYR, 13, and DSPR, 14
                DCHK5(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,
                    incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, z);
                break;
            case 15:
            case 16:
                // Test DSYR2, 15, and DSPR2, 16
                DCHK6(snames[isnum - 1], eps, thresh, foutl, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,

```

```

                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, get_2dfromld(2 * nmax, 0, nmax, z));
            break;
        default:
            break;
    }
    if (fatal || sfatal)
    {
        fout1.WriteLine("\n***** FATAL ERROR - TESTS ABANDONED *****");
        if (trace)
        {
            fout2.Flush();
            fout2.Close();
        }
        fout1.Flush();
        fout1.Close();

        Environment.Exit(1);
    }
}

//Console.WriteLine("4421:");
fout1.WriteLine("\nEND OF TESTS");
if (trace)
{
    fout2.Flush();
    fout2.Close();
}

Console.WriteLine("end.");
fout1.Flush();
fout1.Close();
}

static double[,] _orig_get_2dfromld(int l, int start, int lda, double[] arr)
{
    int k = 1 / lda;
    double[,] tmp = new Double[lda, k];
    int cnt = 0;
    for (int ii = start; ii < k * lda; ii = ii + 1)
    {
        tmp[ii % lda, cnt] = arr[ii];
        if ((ii % lda) == lda - 1)
        {
            cnt++;
        }
    }
    return tmp;
}

public static double[,] get_2dfromld(int l, int start, int lda, double[] arr)
{
    //updated 2015-09-02
    int k = 1 / lda;
    double[,] tmp = new Double[lda, k];
    int cnt1 = 0;
    int cnt2 = 0;
    //Console.WriteLine("2dfromld D1:{0,4:D}", arr.GetLength(0));
    //Console.WriteLine("l:{0,4:D} lda:{1,4:D} k:{2,4:D}", l, lda, k);
    for (int i = start; i < k * lda; i = i + 1)
    {
        tmp[cnt1, cnt2] = arr[i];
        //Console.WriteLine("{0,4:D}{1,8:F5}{2,4:D}{3,4:D}{4,8:F5}", i, arr[i], cnt1, cnt2, tmp[cnt1,cnt2]);
        if (cnt1 == (lda-1))
        {
            cnt2++;
            cnt1 = 0;
        }
        else
        {
            cnt1++;
        }
    }
    return tmp;
}

static void print_2dfromld(int l, int start, int lda, double[] arr, double[,] arr2d)
{
    int k = 0;
    k = 1 / lda;
    //Console.WriteLine("{0}", 1 / lda);
    //double[,] tmp = new Double[lda, k];
    int cnt1 = 0;
    int cnt2 = 0;
    Console.WriteLine("D1:{0,4:D} D2:{1,4:D}", arr2d.GetLength(0), arr2d.GetLength(1));
    Console.WriteLine("l:{0,4:D} lda:{1,4:D} k:{2,4:D}", l, lda, k);
    for (int i = 0; i < l; i = i + 1)
    {
        Console.WriteLine("{0,4:D}{1,8:F5}{2,4:D}{3,4:D}{4,8:F5}", i, arr[i], cnt1, cnt2, arr2d[cnt1,cnt2]);
        if (cnt1 == (lda-1))
        {
            cnt2++;
            cnt1=0;
        }
        else
        {
            cnt1++;
        }
    }
}

static double[,] _orig_get_ldfrom2d(int l, int start, int lda, double[,] arr)
{
    int k = 1 / lda;
    double[] tmp = new Double[l];
    int cnt = 0;
    for (int ii = start; ii < k * lda; ii = ii + 1)
    {
        tmp[ii] = arr[ii % lda, cnt];
        if ((ii % lda) == lda - 1)
        {
            cnt++;
        }
    }
    return tmp;
}

static double[,] _2nd_get_ldfrom2d(int l, int start, int lda, double[,] arr)
{
    int k = 1 / lda;

```

```

double[] tmp = new Double[1];
int cnt1 = 0;
int cnt2 = 0;
for (int i = start; i < k * lda; i = i + 1)
{
    tmp[i] = arr[cnt1,cnt2];
    if (cnt1 == (lda - 1))
    {
        cnt2++;
        cnt1 = 0;
    }
    else
    {
        cnt1++;
    }
}
return tmp;
}
static void copy_ld(double[] src1, int s1, int l1, double[] src2, int s2)
{
    // zero based
    int cnt1 = s2;
    for (int i = s1; i < l1; i = i + 1)
    {
        src2[cnt1] = src1[i];
        cnt1++;
    }
}

public static double[] get_ldfrom2d(int l, int start, int lda, double[,] arr)
{
    int k = l / lda;
    double[] tmp = new Double[1];
    int cnt1 = start;
    int cnt2 = 0;
    for (int i = 0; i < k * lda-start; i = i + 1)
    {
        tmp[i] = arr[cnt1, cnt2];
        if (cnt1 == (lda - 1))
        {
            cnt2++;
            cnt1 = 0;
        }
        else
        {
            cnt1++;
        }
    }
    return tmp;
}
//,get_ldfrom2d_starti(nmax,0,2*nmax,nmax+1-1,ab)),
static double[] get_ldfrom2d_starti(int l1, int start1, int l2, int start2, double[,] arr)
{
    // start zero based
    //int k = l / lda;
    int l = (l1 - start1) * (l2 - start2);
    double[] tmp = new Double[1];
    int cnt1 = start1;
    int cnt2 = start2;
    for (int i = 0; i < l; i = i + 1)
    {
        tmp[i] = arr[cnt1, cnt2];
        cnt1++;
        if (cnt1 >= l1) {
            cnt1 = 0;
            cnt2++;
        }
    }
    return tmp;
}
static double[] get_ldfrom2d_starti_full(int l1, int start1, int l2, int start2, double[,] arr)
{
    // start zero based
    //int k = l / lda;
    int l = (l1 - start1) * (l2 - start2);
    int lnew = l1 * l2; //see get_ldfrom2d_starti (works in some cases) where shortens length based on start positions for new array--here we use
    original length --
    double[] tmp = new Double[1];
    double[] tmp2 = new Double[lnew];
    int cnt1 = start1;
    int cnt2 = start2;
    for (int i = 0; i < l; i = i + 1)
    {
        tmp[i] = arr[cnt1, cnt2];
        cnt1++;
        if (cnt1 >= l1)
        {
            cnt1 = 0;
            cnt2++;
        }
    }
    return tmp;
}
/*static double[,] get_2dfrom2d(int f1, int f2, int s1, int s2, double[,] arr)
{
    int cnt = 0;
    for (int j = 0; j < f2; j = j + 1)
    {
        for (int i = 0; i < f1; i = i + 1)
        {
            tmp[i % s1, cnt] = arr[i, j];
            if ((i % s1) == s1)
            {
                cnt++;
            }
        }
    }
    return tmp;
}*/
static void DCHK1(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
    int nkb, int[] kb, int nalf, double[] alf, int nbet, double[] bet, int ninc, int[] inc, int nmax, int incmax,

```

```

        ref double[,] a, ref double[] aa, double[] _as, ref double[] x, ref double[] xx, double[] xs, ref double[] y, ref double[] yy, double[] ys,
double[] yt, double[] g)
{
    // Tests DGEMV and DGBMV

    // Parameters
    const double zero = 0.0;
    const double half = 0.5;

    /**/ Scalar Arguments
    double eps, thresh;
    int incmax, nalf, nbet, nidim, ninc, nkb, nmax; //, nout, ntra;
    StringWriter nout, ntra;
    bool fatal, rew1, trace;
    string sname;

    // Array Arguments
    double[,] a = new double[nmax,nmax];
    double[] aa = new double[nmax*nmax];
    double[] alf = new double[nalf];
    double[] _as = new double[nmax*nmax];
    double[] bet = new double[nbet];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax*incmax];
    double[] xx = new double[nmax*incmax];
    double[] y = new double[nmax];
    double[] ys = new double[nmax*incmax];
    double[] yt = new double[nmax];
    double[] yy = new double[nmax*incmax];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc];
    int[] kb = new int[nkb];*/

    // Local Scalars
    double alpha, als, beta, bls, err, errmax, transl;
    //int i, ia, ib, ic, iku, im, _in, incx, incxs, incy, incys, ix, iy, kl, kls, ku, kus, laa, lda, ldas, lx, ly, m, ml, ms, n, nargs, nc, nd, nk,
nl, ns;

    int incx, incxs, incy, incys, kl, kls, ku, kus, laa, lda, ldas, lx, ly, m, ml, ms, n, nargs, nc, nd, nk, nl, ns;
    bool banded, full, _null, reset, same, tran;
    string trans, transs, ich;
    /*
    CHARACTER*1          TRANS, TRANSS
    CHARACTER*3          ICH */

    // Local Arrays
    bool[] isame = new bool[13];

    double[,] tmp2d_x;
    double[,] tmp2d_y;
    /* External Functions
    LOGICAL      LDE, LDERES
    EXTERNAL      LDE, LDERES */

    err = 0.0; // added
    ich = "NTC";

    // Executable Statements
    full = (sname.ToUpper().Substring(2, 1) == "E");
    banded = (sname.ToUpper().Substring(2, 1) == "B");
    //nout.WriteLine("^^^DCHK1");
    // Define the number of arguments
    if (full) {
        nargs = 11;
    } else if (banded)
    {
        nargs = 13;
    } else {
        nargs = 0; // added
    }

    nc = 0;
    reset = true;
    errmax = zero;
    m = 0; // added
    //l20
    for (int _in = 1; _in <= nidim; _in = _in + 1)
    {
        //Console.WriteLine("in={0}", _in);
        //nout.WriteLine("^^^_in={0}", _in);
        n = idim[_in - 1];
        nd = n / 2 + 1;

        //l10
        for (int im = 1; im <= 2; im = im + 1)
        {
            //Console.WriteLine("im={0}", im);
            switch (im)
            {
                case 1:
                    m = Math.Max(n - nd, 0);
                    break;
                case 2:
                    m = Math.Min(n + nd, nmax);
                    break;
                default:
                    break;
            }
            if (banded){
                nk = nkb;
            } else {
                nk = 1;
            }
            //l00
            for (int iku = 1; iku <= nk; iku = iku + 1)
            {
                //Console.WriteLine("iku={0}", iku);
                if (banded) {
                    ku = kb[iku - 1];
                    kl = Math.Max(ku - 1, 0);
                } else {
                    ku = n - 1;
                    kl = m - 1;
                }
            }
        }
    }

```

```

// Set LDA to 1 more than minimum value if room
if (banded) {
    lda = kl + ku + 1;
} else {
    lda = m;
}
if (lda < nmax) {
    lda = lda + 1;
}

// Skip tests if not enough room
if (lda > nmax) {
    continue;
}
laa = lda * n;
_null = (n <= 0) || (m <= 0);

// Generate the matrix A
transl = zero;
//Console.WriteLine("bef DMAKE");
DMAKE(sname.Substring(1, 2), " ", " ", m, n, ref a, nmax, ref aa, lda, kl, ku, ref reset, transl);
//Console.WriteLine("aft DMAKE");
//90
for (int ic = 1; ic <= 3; ic = ic + 1) {
    //Console.WriteLine("ic={0}", ic);
    trans = ich.ToUpper().Substring(ic - 1, 1);
    tran = (trans == "T") || (trans == "C");
    if (tran) {
        ml = n;
        nl = m;
    } else {
        ml = m;
        nl = n;
    }

    //80
    for (int ix = 1; ix <= ninc; ix = ix + 1) {
        //Console.WriteLine("ix={0}", ix);
        incx = inc[ix - 1];
        lx = Math.Abs(incx) * nl;

        // Generate the vector X
        transl = half;
        //DMAKE("GE", " ", " ", 1, nl, x, 1, xx, Math.Abs(incx), 0, nl - 1, reset, transl);
        //Console.WriteLine("bef DMAKE");
        tmp2d_x = get_2dfromld(nmax, 0, 1, x);
        DMAKE("GE", " ", " ", 1, nl, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, nl - 1, ref reset, transl);
        x = get_ldfrom2d(nmax, 0, 1, tmp2d_x);
        //Console.WriteLine("aft DMAKE");
        if (nl > 1) {
            x[nl / 2 - 1] = zero;
            xx[1 + Math.Abs(incx) * (nl / 2 - 1) - 1] = zero;
        }
        //70
        for (int iy = 1; iy <= ninc; iy = iy + 1) {
            //Console.WriteLine("iy={0}", iy);
            incy = inc[iy - 1];
            ly = Math.Abs(incy) * ml;
            //60
            for (int ia = 1; ia <= nalf; ia = ia + 1) {
                //Console.WriteLine("ia={0}", ia);
                alpha = alf[ia - 1];
                //50
                for (int ib = 1; ib <= nbet; ib = ib + 1) {
                    //Console.WriteLine("ib={0}", ib);
                    //nout.WriteLine("ib={0}", ib);
                    beta = bet[ib - 1];

                    // Generate the vector Y
                    transl = zero;
                    //DMAKE("GE", " ", " ", 1, ml, y, 1, yy, Math.Abs(incy), 0, ml - 1, reset, transl);
                    //Console.WriteLine("bef DMAKE");
                    tmp2d_y = get_2dfromld(nmax, 0, 1, y);
                    DMAKE("GE", " ", " ", 1, ml, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, ml - 1, ref reset, transl);
                    y = get_ldfrom2d(nmax, 0, 1, tmp2d_y);
                    //Console.WriteLine("bef DMAKE");
                    nc = nc + 1;

                    // Save every datum before calling the subroutine
                    transs = trans;
                    ms = m;
                    ns = n;
                    kls = kl;
                    kus = ku;
                    als = alpha;

                    for (int i = 1; i <= laa; i = i + 1) {
                        _as[i - 1] = aa[i - 1];
                    }

                    ldas = lda;
                    for (int i = 1; i <= lx; i = i + 1) {
                        xs[i - 1] = xx[i - 1];
                    }

                    incxs = incx;
                    bls = beta;
                    for (int i = 1; i <= ly; i = i + 1) {
                        ys[i - 1] = yy[i - 1];
                    }

                    incys = incy;
                    //Console.WriteLine("4660:");
                    // Call the subroutine
                    if (full) {
                        if (trace) {
                            //fout2 = new StreamWriter(snaps, true); // true = append
                            ntra.WriteLine(" {0,6:D}: {1,-6}, {(2,1)}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7:2:D}, {8,4:F1},
                                Y, {9,2:D} .",
                                nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
                        }
                        if (rewi) {
                            //REWIND NTRA - do nothing
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    DGBMV(trans, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, xx, incx, beta, ref yy, incy);
}
else if (banded) {
    if (trace) {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D}, X,
(9,2:D), {10,4:F1}, Y, {11,2:D}) .",
            nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
    }
    if (rewi) {
        //REWIND NTRA - do nothing
    }
    //nout.WriteLine("calling DGBMV in DCHK1");
    DGBMV(trans, m, n, kl, ku, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, xx, incx, beta, ref yy, incy);
}

// Check if error-exit was taken incorrectly
if (!ok) {
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 130
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full) {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1},
Y, {9,2:D}) .",
            nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
    } else if (banded) {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D}, X,
(9,2:D), {10,4:F1}, Y, {11,2:D}) .",
            nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (trans == transs);
isame[1] = (ms == m);
isame[2] = (ns == n);
if (full) {
    isame[3] = (als == alpha);
    isame[4] = LDE(_as, aa, laa);
    isame[5] = (ldas == lda);
    isame[6] = LDE(xs, xx, lx);
    isame[7] = (incxs == incx);
    isame[8] = (bls == beta);
    if (_null) {
        isame[9] = LDE(ys, yy, ly);
    } else {
        //isame[9] = LDERES("GE", " ", 1, ml, ys, yy, Math.Abs(incy)); // double[] ys = new
double[nmax*incmax];
        isame[9] = LDERES("GE", " ", 1, ml, get_2dfromld(nmax * incmax, 0, Math.Abs(incy), ys),
get_2dfromld(nmax * incmax, 0, Math.Abs(incy), yy), Math.Abs(incy)); //nmax*incmax
    }
    isame[10] = (incys == incy);
} else if (banded) {
    isame[3] = (kls == kl);
    isame[4] = (kus == ku);
    isame[5] = (als == alpha);
    isame[6] = LDE(_as, aa, laa);
    isame[7] = (ldas == lda);
    isame[8] = LDE(xs, xx, lx);
    isame[9] = (incxs == incx);
    isame[10] = (bls == beta);
    if (_null) {
        isame[11] = LDE(ys, yy, ly);
    } else {
        //isame[11] = LDERES("GE", " ", 1, ml, ys, yy, Math.Abs(incy));
        isame[11] = LDERES("GE", " ", 1, ml, get_2dfromld(nmax * incmax, 0, Math.Abs(incy), ys),
get_2dfromld(nmax * incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[12] = (incys == incy);
}

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1) {
    same = (same && isame[i - 1]);
    if (!isame[i - 1]) {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same) {
    fatal = true;
    //GOTO 130
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full) {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1},
Y, {9,2:D}) .",
            nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
    } else if (banded) {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D}, X,
(9,2:D), {10,4:F1}, Y, {11,2:D}) .",
            nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
    }
    return;
}

if (!_null) {
    // Check the result
    //nout.WriteLine("DMVCH in DCHK1");
    DMVCH(trans, m, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal,
nout, true);
    errmax = Math.Max(errmax, err);

    // If got really bad answer, report and return
    if (fatal) {
        //GO TO 130
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full) {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D},
(8,4:F1), Y, {9,2:D}) .",
                nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
        } else if (banded) {

```

```

X, {9,2:D}, {10,4:F1}, Y, {11,2:D}) .",
                                nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D},
                                nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
                                }
                                return;
                                } else {
                                // Avoid repeating tests with M.le.0 or N.le.0.
                                goto Loop110;
                                }
                                } //50
                                } //60
                                } //70
                                } //80
                                } //90
                                } //100
                                Loop110: ;
                                //NOOP();
                                } //110
                                } //120

// Report result
if (errmax < thresh) {
    //nout.WriteLine("^^^1");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
} else {
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
    nout.WriteLine("***** BUT WITH MAXIMUM TEST RATIO {0,8:F2} - SUSPECT *****", errmax);
}
return;
}
static void DCHK2(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
    int nkb, int[] kb, int nalf, double[] alf, int nbet, double[] bet, int ninc, int[] inc, int nmax, int incmax,
    ref double[, ] a, ref double[] aa, double[] _as, ref double[] x, ref double[] xx, double[] xs, ref double[] y, ref double[] yy, double[] ys,
double[] yt, double[] g)
{
    // Tests DSYMV, DSBMV and DSPMV

    // Parameters
    const double zero = 0.0;
    const double half = 0.5;

    // Scalar Arguments
    /*double eps, thresh;
    int incmax, nalf, nbet, nidim, ninc, nkb, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    double[, ] a = new double[nmax,nmax];
    double[] aa = new double[nmax*nmax];
    double[] alf = new double[nalf];
    double[] _as = new double[nmax*nmax];
    double[] bet = new double[nbet];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax*incmax];
    double[] xx = new double[nmax*incmax];
    double[] y = new double[nmax];
    double[] ys = new double[nmax*incmax];
    double[] yt = new double[nmax];
    double[] yy = new double[nmax*incmax];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc];
    int[] kb = new int[nkb];*/

    // Local Scalars
    double alpha, als, beta, bls, err, errmax, transl;
    //int i, ia, ib, ic, ik, _in, incx, incxs, incy, incys, ix, iy, k, ks, laa, lda, ldas, lx, ly, n, nargs, nc, nk, ns;
    int incx, incxs, incy, incys, k, ks, laa, lda, ldas, lx, ly, n, nargs, nc, nk, ns;
    bool banded, full, _null, packed, reset, same;
    string uplo, uplos, ich;
    /*
        CHARACTER*1          UPLO, UPLOS
        CHARACTER*2          ICH

    */
    // Local Arrays
    bool[] isame = new bool[13];

    /* External Functions
    LOGICAL    LDE, LDERES
    EXTERNAL   LDE, LDERES */

    double[, ] tmp2d_x;
    double[, ] tmp2d_y;

    err = 0.0; // added
    ich = "UL";

    // Executable Statements
    full = (sname.Substring(2, 1) == "Y");
    banded = (sname.Substring(2, 1) == "B");
    packed = (sname.Substring(2, 1) == "P");

    // Define the number of arguments
    if (full) {
        nargs = 10;
    } else if (banded) {
        nargs = 11;
    } else if (packed) {
        nargs = 9;
    } else {
        nargs = 0; // added
    }

    nc = 0;
    reset = true;
    errmax = zero;

//110

```

```

for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    if (banded)
    {
        nk = nkb;
    }
    else
    {
        nk = 1;
    }

    //100
    for (int ik = 1; ik <= nk; ik = ik + 1)
    {
        if (banded)
        {
            k = kb[ik - 1];
        }
        else
        {
            k = n - 1;
        }

        // Set LDA to 1 more than minimum value if room
        if (banded)
        {
            lda = k + 1;
        }
        else
        {
            lda = n;
        }
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            //GO TO 100
            continue;
        }
        if (packed)
        {
            laa = (n * (n + 1)) / 2;
        }
        else
        {
            laa = lda * n;
        }
        _null = (n <= 0);

    //90
    for (int ic = 1; ic <= 2; ic = ic + 1)
    {
        uplo = ich.ToUpper().Substring(ic - 1, 1);

        // Generate the matrix A
        transl = zero;
        DMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, k, k, ref reset, transl);

    //80
    for (int ix = 1; ix <= ninc; ix = ix + 1)
    {
        incx = inc[ix - 1];
        lx = Math.Abs(incx) * n;

        // Generate the vector X
        transl = half;
        tmp2d_x = get_2dfrom1d(nmax, 0, 1, x);
        DMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
        x = get_ldfrom2d(nmax, 0, 1, tmp2d_x);
        if (n > 1)
        {
            x[(n / 2) - 1] = zero;
            xx[1 + Math.Abs(incx) * (n / 2 - 1) - 1] = zero;
        }

    // 70
    for (int iy = 1; iy <= ninc; iy = iy + 1)
    {
        incy = inc[iy - 1];
        ly = Math.Abs(incy) * n;

    //60
    for (int ia = 1; ia <= nalf; ia = ia + 1)
    {
        alpha = alf[ia - 1];

    //50
    for (int ib = 1; ib <= nbet; ib = ib + 1)
    {
        beta = bet[ib - 1];

        // Generate the vector Y
        transl = zero;
        tmp2d_y = get_2dfrom1d(nmax, 0, 1, y);
        DMAKE("GE", " ", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
        y = get_ldfrom2d(nmax, 0, 1, tmp2d_y);
        nc = nc + 1;

        // Save every datum before calling the subroutine
        uplos = uplo;
        ns = n;
        ks = k;
        als = alpha;

        for (int i = 1; i <= laa; i = i + 1)
        {
            _as[i - 1] = aa[i - 1];
        }
    }
    }
    }
}

```

```

ldas = lda;
for (int i = 1; i <= lx; i = i + 1)
{
    xs[i - 1] = xx[i - 1];
}

incxs = incx;
bls = beta;
for (int i = 1; i <= ly; i = i + 1)
{
    ys[i - 1] = yy[i - 1];
}

incys = incy;

// Call the subroutine
if (full)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, A, {5,3:D}, X, {6,2:D}, {7,4:F1}, Y, {8,2:D})

        nc, sname, uplo, n, alpha, lda, incx, beta, incy);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    DSYMV(uplo, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, xx, incx, beta, ref yy, incy);
}
else if (banded)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1}, Y,

        nc, sname, uplo, n, k, alpha, lda, incx, beta, incy);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    DSBMV(uplo, n, k, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, xx, incx, beta, ref yy, incy);
}
else if (packed)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, AP, X, {5,2:D}, {6,4:F1}, Y, {7,2:D})

        nc, sname, uplo, n, alpha, incx, beta, incy);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    /* nout.WriteLine("NC:{0}",nc);
    nout.WriteLine("UPLO, N, ALPHA, INCX, BETA, INCY");
    nout.WriteLine("{0,1}{1,4:D}{2,10:F3}{3,4:D}{4,10:F3}{5,4:D}",uplo,n,alpha,incx,beta,incy);
    nout.WriteLine("^^^AA");
    for (int i = 1; i <= nmax*nmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",aa[i-1]);
    }
    nout.WriteLine("^^^XX");
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",xx[i-1]);
    }
    nout.WriteLine("^^^YY");
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",yy[i-1]);
    }
    */
    /* nout.Flush();
    nout.Close();
    Environment.Exit(1);*/

    //misc_d.myflag = true;
    DSPMV(uplo, n, alpha, aa, xx, incx, beta, ref yy, incy);
    /*nout.WriteLine("^^^OUT YY");
    for (int i = 1; i <= nmax * incmax; i = i + 1)
    {
        nout.WriteLine("{0,18:F5}", yy[i - 1]);
    }
    */

}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, A, {5,3:D}, X, {6,2:D}, {7,4:F1}, Y, {8,2:D})

        nc, sname, uplo, n, alpha, lda, incx, beta, incy);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1}, Y,

        nc, sname, uplo, n, k, alpha, lda, incx, beta, incy);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, AP, X, {5,2:D}, {6,4:F1}, Y, {7,2:D})

        nc, sname, uplo, n, alpha, incx, beta, incy);
    }
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (ns == n);

```

```

if (full)
{
    isame[2] = (als == alpha);
    isame[3] = LDE(_as, aa, laa);
    isame[4] = (lda == lda);
    isame[5] = LDE(xs, xx, lx);
    isame[6] = (incxs == incx);
    isame[7] = (bls == beta);
    if (_null)
    {
        isame[8] = LDE(ys, yy, ly);
    }
    else
    {
        //isame[8] = LDERES("GE", " ", 1, n, ys, yy, Math.Abs(incy));
        isame[8] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incy), ys), get_2dfrold(nmax *
incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[9] = (incys == incy);
}
else if (banded)
{
    isame[2] = (ks == k);
    isame[3] = (als == alpha);
    isame[4] = LDE(_as, aa, laa);
    isame[5] = (lda == lda);
    isame[6] = LDE(xs, xx, lx);
    isame[7] = (incxs == incx);
    isame[8] = (bls == beta);
    if (_null)
    {
        isame[9] = LDE(ys, yy, ly);
    }
    else
    {
        //isame[9] = LDERES("GE", " ", 1, n, ys, yy, Math.Abs(incy));
        isame[9] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incy), ys), get_2dfrold(nmax *
incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[10] = (incys == incy);
}
else if (packed)
{
    isame[2] = (als == alpha);
    isame[3] = LDE(_as, aa, laa);
    isame[4] = LDE(xs, xx, lx);
    isame[5] = (incxs == incx);
    isame[6] = (bls == beta);
    if (_null)
    {
        isame[7] = LDE(ys, yy, ly);
    }
    else
    {
        isame[7] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incy), ys), get_2dfrold(nmax *
incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[8] = (incys == incy);
}

// If data was incorrectly changed, report and return
same = true;
//40
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, A, {5,3:D}, X, {6,2:D}, {7,4:F1}, Y, {8,2:D})
.",
            nc, sname, uplo, n, alpha, lda, incx, beta, incy);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1}, Y,
(9,2:D))
.",
            nc, sname, uplo, n, k, alpha, lda, incx, beta, incy);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, AP, X, {5,2:D}, {6,4:F1}, Y, {7,2:D})
.",
            nc, sname, uplo, n, alpha, incx, beta, incy);
    }
    return;
}
if (!_null)
{
    // Check the result
    //DMVCH("N", n, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal, nout,
true);
    //nout.WriteLine("take out");
    //misc_d.myflag = true;
    DMVCH("N", n, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal, nout,
true);
    //misc_d.myflag = false;
    errmax = Math.Max(errmax, err);
    // If got really bad answer, report and return.
    if (fatal)
    {
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full)

```

```

        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, A, {5,3:D}, X, {6,2:D}, {7,4:F1}, Y,
(8,2:D))      .",
                nc, sname, uplo, n, alpha, lda, incx, beta, incy);
        }
        else if (banded)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1},
Y, {9,2:D})    .",
                nc, sname, uplo, n, k, alpha, lda, incx, beta, incy);
        }
        else if (packed)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, AP, X, {5,2:D}, {6,4:F1}, Y, {7,2:D})
            .",
                nc, sname, uplo, n, alpha, incx, beta, incy);
        }
        return;
    }
    else
    {
        // Avoid repeating tests with N.le.0
        goto Loop110;
    }
} //50
} //60
} //70
} //80
} //90
} //100
Loop110: ;
} //110

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^2");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
}

static void DCHK3(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
    int nkb, int[] kb, int ninc, int[] inc, int nmax, int incmax, ref double[,] a, ref double[] aa, double[] _as, ref double[] x, ref double[] xx,
double[] xs, double[] xt, double[] g, double[] z)
{
    // Tests DTRMV, DTBMV, DTPMV, DTRSV, DTBSV and DTPSV

    // Parameters
    const double zero = 0.0;
    const double half = 0.5;
    const double one = 1.0;

    // Scalar Arguments
    /*double eps, thresh;
    int incmax, nidim, ninc, nkb, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    double[,] a = new double[nmax,nmax];
    double[] aa = new double[nmax*nmax];
    double[] _as = new double[nmax*nmax];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax*incmax];
    double[] xt = new double[nmax];
    double[] xx = new double[nmax*incmax];
    double[] z = new double[nmax];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc];
    int[] kb = new int[nkb]; */

    // Local Scalars
    double err, errmax, transl;
    //int i, icd, ict, icu, ik, _in, incx, incxs, ix, k, ks, laa, lda, ldas, lx, n, nargs, nc, nk, ns;
    int incx, incxs, k, ks, laa, lda, ldas, lx, n, nargs, nc, nk, ns;
    bool banded, full, _null, packed, reset, same;
    string diag, diags, trans, transs, uplo, uplos, ichd, ichu, icht;
    /* CHARACTER*1          DIAG, DIAGS, TRANS, TRANSS, UPLO, UPLOS
       CHARACTER*2          ICHD, ICHU
       CHARACTER*3          ICHT
    */
    // Local Arrays
    bool[] isame = new bool[13];

    double[,] tmp2d_x;

    err = 0.0; //added
    ichu = "UL";
    icht = "NTC";
    ichd = "UN";

    // Executable Statements
    full = (sname.Substring(2, 1) == "R");
    banded = (sname.Substring(2, 1) == "B");
    packed = (sname.Substring(2, 1) == "P");

    // Define the number of arguments
    if (full) {
        nargs = 8;
    }
    else if (banded) {
        nargs = 9;
    }
    else if (packed) {
        nargs = 7;
    }
}

```

```

} else {
    nargs = 0; // added
}

nc = 0;
reset = true;
errmax = zero;

// Set up zero vector for DMVCH
for (int i = 1; i <= nmax; i = i + 1)
{
    z[i - 1] = zero;
}

//110
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    if (banded)
    {
        nk = nkb;
    }
    else
    {
        nk = 1;
    }

    //100
    for (int ik = 1; ik <= nk; ik = ik + 1)
    {
        if (banded)
        {
            k = kb[ik - 1];
        }
        else
        {
            k = n - 1;
        }

        // Set LDA to 1 more than minimum value if room
        if (banded)
        {
            lda = k + 1;
        }
        else
        {
            lda = n;
        }
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            //GO TO 100
            goto Loop100;
        }
        if (packed)
        {
            laa = (n * (n + 1)) / 2;
        }
        else
        {
            laa = lda * n;
        }
        _null = (n <= 0);
        //90
        for (int icu = 1; icu <= 2; icu = icu + 1)
        {
            uplo = ichu.Substring(icu - 1, 1);
            //80
            for (int ict = 1; ict <= 3; ict = ict + 1)
            {
                trans = icht.Substring(ict - 1, 1);
                //70
                for (int icd = 1; icd <= 2; icd = icd + 1)
                {
                    diag = ichd.Substring(icd - 1, 1);

                    // Generate the matrix A
                    transl = zero;
                    DMAKE(sname.Substring(1, 2), uplo, diag, n, n, ref a, nmax, ref aa, lda, k, k, ref reset, transl);
                    //60
                    for (int ix = 1; ix <= ninc; ix = ix + 1)
                    {
                        incx = inc[ix - 1];
                        lx = Math.Abs(incx) * n;

                        // Generate the vector X
                        transl = half;
                        tmp2d_x = get_2dfrom1d(nmax, 0, 1, x);
                        DMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
                        x = get_1dfrom2d(nmax, 0, 1, tmp2d_x);
                        if (n > 1)
                        {
                            x[(n / 2) - 1] = zero;
                            xx[1 + Math.Abs(incx) * (n / 2 - 1) - 1] = zero;
                        }
                        nc = nc + 1;

                        // Save every datum before calling the subroutine
                        uplos = uplo;
                        transs = trans;
                        diags = diag;
                        ns = n;
                        ks = k;
                        for (int i = 1; i <= laa; i = i + 1)
                        {
                            _as[i - 1] = aa[i - 1];
                        }
                        ldas = lda;
                        for (int i = 1; i <= lx; i = i + 1)

```

```

{
    xs[i - 1] = xx[i - 1];
}
incxs = incx;

// Call the subroutine
if (sname.Substring(3, 2) == "MV")
{
    if (full)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})
            nc, sname, uplo, trans, diag, n, lda, incx);
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        DTRMV(uplo, trans, diag, n, get_2dfromld(nmax*nmax,0,lda, aa), lda, ref xx, incx);
    }
    else if (banded)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})
            nc, sname, uplo, trans, diag, n, lda, incx);
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        DTBMV(uplo, trans, diag, n, k, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref xx, incx);
    }
    else if (packed)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1}, {5,3:d}), AP, X, {6,2:D})
            nc, sname, uplo, trans, diag, n, incx);
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        DTPMV(uplo, trans, diag, n, aa, ref xx, incx);
    }
}
else if (sname.Substring(3, 2) == "SV")
{
    if (full)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})
            nc, sname, uplo, trans, diag, n, lda, incx);
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        DTRSV(uplo, trans, diag, n, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref xx, incx);
    }
    else if (banded)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1}, {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})
            nc, sname, uplo, trans, diag, n, k, lda, incx);
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        /*nout.WriteLine("NC:{0}",nc);
        nout.WriteLine("{0,4:D} {1,4:D}",nmax*nmax,lda*n);
        nout.WriteLine("UPLO, TRANS, DIAG, N, K, LDA, INCX");
        nout.WriteLine("{0,1}{1,1}{2,1}{3,4:D}{4,4:D}{5,4:D}{6,4:D}",uplo,trans,diag,n,k,lda,incx);
        nout.WriteLine("^^^AA");
        //tmp2d_x = get_2dfromld(nmax*nmax,0,lda,aa);
        for (int i = 1; i <= nmax*nmax; i = i + 1) {
            nout.WriteLine("{0,18:F5}",aa[i-1]);

        }
        nout.WriteLine("^^^XX");
        for (int i = 1; i <= nmax; i = i + 1) {
            nout.WriteLine("{0,18:F5}",xx[i-1]);
        }
        */

        //misc_d.myflag = true;

        //BSV(string uplo, string trans, string diag, int n, int k, double[,] a, int lda, ref double[] x, int incx)
        DTBSV(uplo, trans, diag, n, k, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref xx, incx);
        /*
        nout.WriteLine("^^^OUT XX");
        for (int i = 1; i <= nmax; i = i + 1)
        {
            nout.WriteLine("{0,18:F5}", xx[i - 1]);
        }
        */
    }
    else if (packed)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1}, {5,3:d}, AP, X, {6,2:D})
            nc, sname, uplo, trans, diag, n, incx);
        }
    }
}

```

```

        if (rewi)
        {
            //REWIND NTRA
        }
        DTSPV(uplo, trans, diag, n, aa, ref xx, incx);
    }

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})          .",
            nc, sname, uplo, trans, diag, n, lda, incx);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})
            nc, sname, uplo, trans, diag, n, k, lda, incx);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}), {5,3:d}), AP, X, {6,2:D})          .",
            nc, sname, uplo, trans, diag, n, incx);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (trans == transs);
isame[2] = (diag == diags);
isame[3] = (ns == n);
if (full)
{
    isame[4] = LDE(_as, aa, laa);
    isame[5] = (ldas == lda);
    if (_null)
    {
        isame[6] = LDE(xs, xx, lx);
    }
    else
    {
        isame[6] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incx), xs), get_2dfrold(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
    }
    isame[7] = (incxs == incx);
}
else if (banded)
{
    isame[4] = (ks == k);
    isame[5] = LDE(_as, aa, laa);
    isame[6] = (ldas == lda);
    if (_null)
    {
        isame[7] = LDE(xs, xx, lx);
    }
    else
    {
        isame[7] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incx), xs), get_2dfrold(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
    }
    isame[8] = (incxs == incx);
}
else if (packed)
{
    isame[4] = LDE(_as, aa, laa);
    if (_null)
    {
        isame[5] = LDE(xs, xx, lx);
    }
    else
    {
        isame[5] = LDERES("GE", " ", 1, n, get_2dfrold(nmax * incmax, 0, Math.Abs(incx), xs), get_2dfrold(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
    }
    isame[6] = (incxs == incx);
}

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})          .",
            nc, sname, uplo, trans, diag, n, lda, incx);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})
            nc, sname, uplo, trans, diag, n, k, lda, incx);
    }
    else if (packed)

```



```

int[] idim = new int[nidim];
int[] inc = new int[ninc];*/

// Local Scalars
double alpha, als, err, errmax, transl;
//int i, ia, im, _in, incx, incxs, incy, incys, ix, iy, j, laa, lda, ldas, lx, ly, m, ms, n, nargs, nc, nd, ns;
int incx, incxs, incy, incys, laa, lda, ldas, lx, ly, m, ms, n, nargs, nc, nd, ns;
bool _null, reset, same;

// Local Arrays
double[] w = new double[1];
bool[] isame = new bool[13];

double[, ] tmp2d_x, tmp2d_y, tmp2d_aa;

// Executable Statements
err = 0.0; // added
// Define the number of arguments
nargs = 9;
nc = 0;
reset = true;
errmax = zero;
m = 0; // added

//l20
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    nd = n / 2 + 1;

    //l10
    for (int im = 1; im <= 2; im = im + 1)
    {
        if (im == 1)
        {
            m = Math.Max(n - nd, 0);
        }
        if (im == 2)
        {
            m = Math.Min(n + nd, nmax);
        }

        // Set LDA to 1 more than minimum value if room.
        lda = m;
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            goto Loop110;
        }
        laa = lda * n;
        _null = (n <= 0) || (m <= 0);

    //l00
    for (int ix = 1; ix <= ninc; ix = ix + 1)
    {
        incx = inc[ix - 1];
        lx = Math.Abs(incx) * m;

        // Generate the vector X
        transl = half;
        tmp2d_x = get_2dfromld(nmax, 0, 1, x);
        DMAKE("GE", "", " ", 1, m, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, m - 1, ref reset, transl);
        x = get_ldfrom2d(nmax, 0, 1, tmp2d_x);
        if (m > 1)
        {
            x[(m / 2) - 1] = zero;
            xx[(1 + Math.Abs(incx)) * (m / 2 - 1) - 1] = zero;
        }

    //90
    for (int iy = 1; iy <= ninc; iy = iy + 1)
    {
        incy = inc[iy - 1];
        ly = Math.Abs(incy) * n;

        // Generate the vector Y
        transl = zero;
        tmp2d_y = get_2dfromld(nmax, 0, 1, y);
        DMAKE("GE", "", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
        y = get_ldfrom2d(nmax, 0, 1, tmp2d_y);
        if (n > 1)
        {
            y[(n / 2) - 1] = zero;
            yy[(1 + Math.Abs(incy)) * (n / 2 - 1) - 1] = zero;
        }

    //80
    for (int ia = 1; ia <= nalf; ia = ia + 1)
    {
        alpha = alf[ia - 1];

        // Generate the matrix A

        transl = zero;
        DMAKE(sname.Substring(1, 2), "", " ", m, n, ref a, nmax, ref aa, lda, m - 1, n - 1, ref reset, transl);
        nc = nc + 1;

        // Save every datum before calling the subroutine
        ms = m;
        ns = n;
        als = alpha;

        for (int i = 1; i <= laa; i = i + 1)
        {
            _as[i - 1] = aa[i - 1];
        }
        ldas = lda;
        for (int i = 1; i <= lx; i = i + 1)
        {

```

```

        xs[i - 1] = xx[i - 1];
    }

    incxs = incx;
    for (int i = 1; i <= ly; i = i + 1)
    {
        ys[i - 1] = yy[i - 1];
    }
    incys = incy;

    // Call the subroutine
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
            nc, sname, m, n, alpha, incx, incy, lda);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    /* nout.WriteLine("NC:{0,4:D}{1,4:D}{2,4:D}{3,4:D}{4,4:D}{5,4:D}", nc, _in, im, ix, iy, ia);
    nout.WriteLine("M, N, ALPHA, INCX, INCY, LDA");
    nout.WriteLine("{0,4:D}{1,4:D}{2,10:F3}{3,4:D}{4,4:D}{5,4:D}", m, n, alpha, incx, incy, lda);
    nout.WriteLine("^^^XX");
    //tmp2d_x = get_2dfromld(nmax*nmax, 0, lda, aa);
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}", xx[i-1]);
    }
    nout.WriteLine("^^^YY");
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}", yy[i-1]);
    }
    nout.WriteLine("^^^AA");
    for (int i = 1; i <= nmax*nmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}", aa[i-1]);
    }
    */
    tmp2d_aa = get_2dfromld(nmax * nmax, 0, lda, aa);
    // (int m, int n, double alpha, double[] x, int incx, double[] y, int incy, ref double[,] a, int lda
    DGER(m, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
    aa = get_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);

    /*
    nout.WriteLine("^^^OUT AA");
    for (int i = 1; i <= nmax*nmax; i = i + 1)
    {
        nout.WriteLine("{0,18:F5}", aa[i - 1]);
    }
    */

    // Check if error-exit was taken incorrectly
    if (!ok)
    {
        nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");

        fatal = true;
        //GO TO 140
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D} {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
            nc, sname, m, n, alpha, incx, incy, lda);
        return;
    }

    // See what data changed inside subroutine
    isame[0] = (ms == m);
    isame[1] = (ns == n);
    isame[2] = (als == alpha);
    isame[3] = LDE(xs, xx, lx);
    isame[4] = (incxs == incx);
    isame[5] = LDE(ys, yy, ly);
    isame[6] = (incys == incy);
    if (_null)
    {
        isame[7] = LDE(_as, aa, laa);
    }
    else
    {
        isame[7] = LDERES("GE", " ", m, n, get_2dfromld(nmax * nmax, 0, lda, _as), get_2dfromld(nmax * nmax, 0, lda, aa), lda);
    }
    isame[8] = (ldas == lda);

    // If data was incorrectly changed, report and return
    same = true;

    //40
    for (int i = 1; i <= nargs; i = i + 1)
    {
        same = (same && isame[i - 1]);
        if (!isame[i - 1])
        {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }
    if (!same)
    {
        fatal = true;
        //GO TO 140
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D} {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
            nc, sname, m, n, alpha, incx, incy, lda);
        return;
    }

    if (!_null)
    {
        // Check the result column by column
        if (incx > 0)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                z[i - 1] = x[i - 1];
            }
        }
    }

```

```

        else
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                z[i - 1] = x[(m - i + 1) - 1];
            }
        }

        //70
        for (int j = 1; j <= n; j = j + 1)
        {
            if (incy > 0)
            {
                w[0] = y[j - 1];
            }
            else
            {
                w[0] = y[(n - j + 1) - 1];
            }
            //DMVCH("N", m, 1, alpha, z, nmax, w, 1, one, a[1-1,j-1], 1, yt, g, aa[1+(j-1)*lda -1], eps, err, fatal, nout,
            //DMVCH("N", m, 1, alpha, get_2dfrom1d(nmax, 0, nmax, z), nmax, w, 1, one, get_darr2d(0, nmax - 1, j - 1, a), 1, ref
            yt, ref g, get_darr(1 + (j - 1) * lda - 1, nmax * nmax - 1, aa), eps, ref err, ref fatal, nout, true);

            errmax = Math.Max(errmax, err);

            // If got really bad answer, report and return
            if (fatal)
            {
                //GO TO 130
                nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
                //140
                nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
                nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D} {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})
                nc, sname, m, n, alpha, incx, incy, lda);
                return;
            }
        } //70
    }
    else
    {
        // Avoid repeating tests with M.le.0 or N.le.0.
        goto Loop110;
    }
} //80
} //90
} //100
Loop110: ;
} //110
} //120

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^4");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
    sname, nc, errmax);
}

}
static void DCHK5(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
int nalf, double[] alf, int ninc, int[] inc, int nmax, int incmax, ref double[,] a, ref double[] aa, double[] _as, ref double[] x, ref double[]
xx, double[] xs, ref double[] y,
ref double[] yy, double[] ys, double[] yt, double[] g, double[] z)
{
    // Tests DSYR and DSPR

    // Parameters
    const double zero = 0.0;
    const double half = 0.5;
    const double one = 1.0;

    /**/ Scalar Arguments
    double eps, thresh;
    int incmax, nalf, nidim, ninc, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    double[,] a = new double[nmax,nmax];
    double[] aa = new double[nmax*nmax];
    double[] alf = new double[nalf];
    double[] _as = new double[nmax*nmax];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax*incmax];
    double[] xx = new double[nmax*incmax];
    double[] y = new double[nmax];
    double[] ys = new double[nmax*incmax];
    double[] yt = new double[nmax];
    double[] yy = new double[nmax*incmax];
    double[] z = new double[nmax];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc]; /*

    // Local Scalars
    double alpha, als, err, errmax, transl;
    int i, ia, ic, _in, incx, incxs, ix, j, ja, jj, laa, lda, ldas, lj, lx, n, nargs, nc, nd, ns;
    //int incx, incxs, ja, jj, laa, lda, ldas, lj, lx, n, nargs, nc, nd, ns;
    bool full, _null, packed, reset, same, upper;
    string uplo, uplos, ich;

    // Local Arrays
    double[] w = new double[1];
    bool[] isame = new bool[13];

    double[,] tmp2d_x, tmp2d_aa,tmp2d;

```

```

double[] tmp1d;
err = 0.0; // added
ich = "UL";

// Executable Statements
full = (sname.ToUpper().Substring(2, 1) == "Y");
packed = (sname.ToUpper().Substring(2, 1) == "P");

// Define the number of arguments
if (full)
{
    nargs = 7;
}
else if (packed)
{
    nargs = 6;
}
else {
    nargs = 0; // added
}

nc = 0;
reset = true;
errmax = zero;

//100
for (_in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];

    // Set LDA to 1 more than minimum value if room
    lda = n;
    if (lda < nmax)
    {
        lda = lda + 1;
    }

    // Skip tests if not enough room
    if (lda > nmax)
    {
        //GO TO 100
        goto Loop100; //continue?
    }
    if (packed)
    {
        laa = (n * (n + 1)) / 2;
    }
    else
    {
        laa = lda * n;
    }

    //90
    for (ic = 1; ic <= 2; ic = ic + 1)
    {
        uplo = ich.Substring(ic - 1, 1);
        upper = (uplo == "U");

        //80
        for (ix = 1; ix <= ninc; ix = ix + 1)
        {
            incx = inc[ix - 1];
            lx = Math.Abs(incx) * n;

            // Generate the vector X
            transl = half;
            tmp2d x = get_2dfrom1d(nmax, 0, 1, x);
            DMAKE("GE", "", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
            x = get_ldfrom2d(nmax, 0, 1, tmp2d_x);
            if (n > 1)
            {
                x[(n / 2) - 1] = zero;
                xx[(1 + Math.Abs(incx) * (n / 2 - 1)) - 1] = zero;
            }

            //70
            for (ia = 1; ia <= nalf; ia = ia + 1)
            {
                alpha = alf[ia - 1];
                _null = (n <= 0) || (alpha == zero);

                // Generate the matrix A
                transl = zero;

                /*nout.WriteLine("^^^DMAKE OUT AA BEFORE");

                for (i = 1; i <= nmax * nmax; i = i + 1)
                {
                    nout.WriteLine("{0,4:D}{1,18:F5}", i, aa[i - 1]);
                }
                DMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, n - 1, n - 1, ref reset, transl);
                /*nout.WriteLine("NC:{0,4:D}", nc);
                nout.WriteLine("{0,2}{1,1}{2,4:D}{3,4:D},{4,4:D},{5,4:D}{6,4:D}{7,4:D}{8,4}{9,18:F5}", sname.Substring(1, 2), uplo, n, n, nmax,
lda, n - 1, n - 1, reset,
                transl);
                nout.WriteLine("^^^DMAKE OUT AA");

                for (i = 1; i <= nmax * nmax; i = i + 1)
                {
                    nout.WriteLine("{0,4:D}{1,18:F5}", i, aa[i - 1]);
                }
                /*nout.WriteLine("{0}]", sname.Substring(0, 4));
                //nout.Flush();
                //nout.Close();

                //Environment.Exit(1);
                //if (sname.Substring(0, 4) == "DSYR")
                //{
                    //nout.Flush();
                    //nout.Close();
                    //Environment.Exit(1);
                //}
                nc = nc + 1;

```

```

// Save every datum before calling the subroutine
uplos = uplo;
ns = n;
als = alpha;
for ( i = 1; i <= laa; i = i + 1)
{
    _as[i - 1] = aa[i - 1];
}

ldas = lda;
for ( i = 1; i <= lx; i = i + 1)
{
    xs[i - 1] = xx[i - 1];
}
incxs = incx;

// Call the subroutine
if (full)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .", nc,
sname, uplo, n, alpha, incx, lda);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    // nout.WriteLine("NC:{0,4:D}{1,4:D}{2,4:D}{3,4:D}{4,4:D}", nc, _in, ic, ix, ia);
    //nout.WriteLine("UPLO, N, ALPHA, INCX, LDA");
    nout.WriteLine("{0,1}{1,4:D}{2,10:F3}{3,4:D}{4,4:D}", uplo, n, alpha, incx, lda);
    nout.WriteLine("^^^XX");

    for ( i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}", xx[i-1]);
    }

    nout.WriteLine("^^^AA");
    for ( i = 1; i <= nmax*nmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}", aa[i-1]);
    }

    /*
    tmp2d aa = get_2dfrom1d(nmax * nmax, 0, lda, aa);
    DSYR(uplo, n, alpha, xx, incx, ref tmp2d aa, lda);
    aa = get_1dfrom2d(nmax * nmax, 0, lda, tmp2d aa);
    */
    nout.WriteLine("^^^OUT AA");
    for ( i = 1; i <= nmax*nmax; i = i + 1)
    {
        nout.WriteLine("{0,18:F5}", aa[i - 1]);
    }
    /*
}
else if (packed)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc, sname,
uplo, n, alpha, incx);
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    DSPR(uplo, n, alpha, xx, incx, ref aa);
}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .", nc,
sname, uplo, n, alpha, incx, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc, sname,
uplo, n, alpha, incx);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (ns == n);
isame[2] = (als == alpha);
isame[3] = LDE(xs, xx, lx);
isame[4] = (incxs == incx);

if (_null)
{
    isame[5] = LDE(_as, aa, laa);
}
else
{
    isame[5] = LDERES(sname.Substring(1, 2), uplo, n, n, get_2dfrom1d(nmax * nmax, 0, lda, _as), get_2dfrom1d(nmax * nmax, 0,
lda, aa), lda);
}

if (!packed)
{
    isame[6] = (ldas == lda);
}

// If data was incorrectly changed, report and return
same = true;
for ( i = 1; i <= nargs; i = i + 1)

```

```

    {
        same = (same && isame[i - 1]);
        if (!isame[i - 1])
        {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }

    if (!same)
    {
        fatal = true;
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full)
        {
            nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D}}", sname, uplo, n, alpha, incx, lda);
        }
        else if (packed)
        {
            nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)", uplo, n, alpha, incx);
        }
        return;
    }

    if (!_null)
    {
        // Check the result column by column
        if (incx > 0)
        {
            for (i = 1; i <= n; i = i + 1)
            {
                z[i - 1] = x[i - 1];
            }
        }
        else
        {
            for (i = 1; i <= n; i = i + 1)
            {
                z[i - 1] = x[(n - i + 1) - 1];
            }
        }

        ja = 1;
        //60
        for (j = 1; j <= n; j = j + 1)
        {
            w[0] = z[j - 1];
            if (upper)
            {
                jj = 1;
                lj = j;
            }
            else
            {
                jj = j;
                lj = n - j + 1;
            }
            //A(NMAX,*) A,NMAX...
            //DMVCH("N", lj, 1, alpha, z[jj-1], lj, w, 1, one, a[jj-1, j-1], 1, yt, g, aa[ja-1], eps, err, fatal, nout, true);
            //---nout.WriteLine("NC:{0,4:D}{1,4:D}{2,4:D}{3,4:D}{4,4:D}{5,4:D}{6,4:D}{7,4:D} BEF DMVCH", nc, _in, ic, ix,
            //DMVCH(string trans, int m, int n, double alpha, double[,] a, int nmax, double[] x, int incx, double beta, double[]
            //ref double[] g, double[] yy, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv)
            // SUBROUTINE DMVCH( TRANS, M, N, ALPHA, A, NMAX, X, INCX, BETA, Y,
            // INCY, YT, G, YY, EPS, ERR, FATAL, NOUT, MV )
            //A=get_2dfromld Z, X=W, Y=get_darr2d A, YT, G, YY=get_darr AA
            //---nout.WriteLine("M,ALPHA,NMAX,ERR/A,X,Y,YT,G,YY BEFORE");
            //---nout.WriteLine("{0,4:D}{1,8:F3}{2,4:D}{3,8:F3}", lj, alpha, lj, err);

            //a=Z(jj) tmp2d = get_2dfromld(nmax, jj - 1, lj, z);
            /*nout.WriteLine("Z: JJ={0,4:D}", jj);
            for (int ii = 0; ii < nmax; ii = ii + 1)
            {
                nout.WriteLine("{0,4}{1,8:F3}", ii + 1, z[ii]);
            }
            nout.WriteLine("W:");
            nout.WriteLine("{0,8:F3}", w[0]);
            nout.WriteLine("A: JJ,J={0,4:D}{0,4:D}", jj, j);
            for (int ii = 0; ii < nmax; ii = ii + 1)
            {
                nout.WriteLine("{0,4}{1,8:F3}", ii + 1, a[ii, j - 1]);
            }
            nout.WriteLine("YT:");
            for (int ii = 0; ii < nmax; ii = ii + 1)
            {
                nout.WriteLine("{0,4}{1,8:F3}", ii + 1, yt[ii]);
            }
            nout.WriteLine("G:");
            for (int ii = 0; ii < nmax; ii = ii + 1)
            {
                nout.WriteLine("{0,4}{1,8:F3}", ii + 1, g[ii]);
            }
            nout.WriteLine("AA: JA={0,4:D}", ja);
            for (int ii = 0; ii < nmax * nmax; ii = ii + 1)
            {
                nout.WriteLine("{0,4}{1,8:F3}", ii + 1, aa[ii]);
            }
            nout.WriteLine("IN DMVCH:");
            // SUBROUTINE DMVCH( TRANS, M, N, ALPHA, A, NMAX, X, INCX, BETA, Y,
            // INCY, YT, G, YY, EPS, ERR, FATAL, NOUT, MV )
            //Console.WriteLine("nc:{0,4:D} lj:{0,4:D}", nc, lj);
            //print_2dfromld(nmax, jj - 1, lj, z, get_2dfromld(nmax, jj - 1, lj, z));

            DMVCH("N", lj, 1, alpha, get_2dfromld(nmax, jj - 1, lj, z), lj, w, 1, one, get_darr2d(jj - 1, nmax - 1, j - 1, a),
            1, ref yt, ref g, get_darr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);

            //DMVCH("N", lj, 1, alpha, new double[,]{(misc_d.get_arrrd(jj-1,nmax-1,z)),{0}}, lj, w, 1, one, get_darr2d(jj-1,nmax-
            1,j-1,a), 1, yt, g, get_darr(ja-1,(nmax-1)*nmax-1,aa), eps, err, fatal, nout, true);
            //A(NMAX,*) G,X,Y,YT,YY
            /* double[,] a = new double[nmax,nmax];
            double[] aa = new double[nmax*nmax];

```

```

double[] g = new double[nmax];
double[] x = new double[nmax];
double[] y = new double[nmax];
double[] yt = new double[nmax];
double[] yy = new double[nmax*incmax];
double[] z = new double[nmax];*/

/* nout.WriteLine("M,ALPHA,NMAX,ERR/A,X,Y,YT,G,YY");
nout.WriteLine("{0,4:D}{1,18:F3}{2,4:D}{3,18:F3}",lj,alpha, lj,err);

//a=2(jj) tmp2d = get_2dfrom1d(nmax, jj - 1, lj, z);
nout.WriteLine("Z: JJ={0,4:D}",jj);
for (int ii = 0; ii < nmax; ii = ii + 1) {
    nout.WriteLine("{0,4}{1,18:F3}",ii+1,z[ii]);
}
nout.WriteLine("W:");
nout.WriteLine("{0,18:F3}",w[0]);
nout.WriteLine("A: JJ,J={0,4:D}{0,4:D}",jj,j);
for (int ii = 0; ii < nmax; ii = ii + 1) {
    nout.WriteLine("{0,4}{1,18:F3}",ii+1,a[ii,j-1]);
}
nout.WriteLine("YT:");
for (int ii = 0; ii < nmax; ii = ii + 1) {
    nout.WriteLine("{0,4}{1,18:F3}",ii+1,yt[ii]);
}
nout.WriteLine("G:");
for (int ii = 0; ii < nmax; ii = ii + 1) {
    nout.WriteLine("{0,4}{1,18:F3}",ii+1,g[ii]);
}
nout.WriteLine("AA: JA={0,4:D}",ja);
for (int ii = 0; ii < nmax*nmax; ii = ii + 1) {
    nout.WriteLine("{0,4}{1,18:F3}",ii+1,aa[ii]);
}*/
//get_darr(ja - 1, (nmax * nmax) - 1, aa)

if (full)
{
    if (upper)
    {
        ja = ja + lda;
    }
    else
    {
        ja = ja + lda + 1;
    }
}
else
{
    ja = ja + lj;
}
errmax = Math.Max(errmax, err);
//nout.WriteLine("{0}{1,18:F5}", fatal, errmax);
// If got really bad answer, report and return
if (fatal)
{
    //GO TO 110
    nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .",
nc, sname, uplo, n, alpha, incx, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc,
sname, uplo, n, alpha, incx);
    }
    return;
}
} //60
}
else
{
    // Avoid repeating tests if N.le.0.
    if (n <= 0)
    {
        //GO TO 100
        goto Loop100;
        /*if (errmax < thresh)
        {
            //nout.WriteLine("^^^5");
            nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
        }
        else
        {
            nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO
(2,8:F2) - SUSPECT *****", sname, nc, errmax);
        }
        return;*/
    }
}
} //70
} //80
} //90
Loop100: ;
} //100

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^6");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
}
}

static void DCHK6(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,

```

```

    int nalf, double[] alf, int ninc, int[] inc, int nmax, int incmax, ref double[,] a, ref double[] aa, double[] _as, ref double[] x, ref double[]
xx, double[] xs, ref double[] y,
    ref double[] yy, double[] ys, double[] yt, double[] g, double[,] z)
{
    // Tests DSYR2 and DSPR2

    // Parameters
    const double zero = 0.0;
    const double half = 0.5;
    const double one = 1.0;

    // Scalar Arguments
    /*double eps, thresh;
    int incmax, nalf, nidim, ninc, nmax; // nout, ntra;
    StringWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;*/

    // Array Arguments
    /*double[,] a = new double[nmax,nmax];
    double[] aa = new double[nmax*nmax];
    double[] alf = new double[nalf];
    double[] _as = new double[nmax*nmax];
    double[] g = new double[nmax];
    double[] x = new double[nmax];
    double[] xs = new double[nmax*incmax];
    double[] xx = new double[nmax*incmax];
    double[] y = new double[nmax];
    double[] ys = new double[nmax*incmax];
    double[] yt = new double[nmax];
    double[] yy = new double[nmax*incmax];
    double[,] z = new double[nmax,2];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc];*/

    // Local Scalars
    double alpha, als, err, errmax, transl;
    //int i, ia, ic, _in, incx, incxs, incy, incys, ix, iy, j, ja, jj, laa, lda, ldas, lj, lx, ly, n, nargs, nc, ns;
    int incx, incxs, incy, incys, ja, jj, laa, lda, ldas, lj, lx, ly, n, nargs, nc, ns;
    bool full, _null, packed, reset, same, upper;
    string uplo, uplos, ich;

    // Local Arrays
    double[] w = new double[2];
    bool[] isame = new bool[13];

    double[,] tmp2d_x, tmp2d_y, tmp2d_aa;

    err = 0.0; // added

    ich = "UL";

    // Executable Statements
    full = (sname.Substring(2, 1) == "Y");
    packed = (sname.Substring(2, 1) == "P");

    // Define the number of arguments
    if (full)
    {
        nargs = 9;
    }
    else if (packed)
    {
        nargs = 8;
    }
    else {
        nargs = 0; // added
    }

    nc = 0;
    reset = true;
    errmax = zero;

    //140
    for (int _in = 1; _in <= nidim; _in = _in + 1)
    {
        n = idim[_in - 1];
        // Set LDA to 1 more than minimum value if room
        lda = n;
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            //GO TO 140
            // Report result
            if (errmax < thresh)
            {
                //nout.WriteLine("^^^7");
                nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
            }
            else
            {
                nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT
*****", sname, nc, errmax);
            }
        }
        return;
    }
    if (packed)
    {
        laa = (n * (n + 1)) / 2;
    }
    else
    {
        laa = lda * n;
    }

    //130
    for (int ic = 1; ic <= 2; ic = ic + 1)
    {
        uplo = ich.ToUpper().Substring(ic - 1, 1);

```

```

upper = (uplo == "U");

//120
for (int ix = 1; ix <= ninc; ix = ix + 1)
{
    incx = inc[ix - 1];
    lx = Math.Abs(incx) * n;

    // Generate the vector X
    transl = half;
    tmp2d_x = get_2dfromld(nmax, 0, 1, x);
    DMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
    x = get_ldfrom2d(nmax, 0, 1, tmp2d_x);
    if (n > 1)
    {
        x[(n / 2) - 1] = zero;
        xx[(1 + Math.Abs(incx) * (n / 2 - 1)) - 1] = zero;
    }

//110
for (int iy = 1; iy <= ninc; iy = iy + 1)
{
    incy = inc[iy - 1];
    ly = Math.Abs(incy) * n;

    // Generate the vector Y
    transl = zero;
    tmp2d_y = get_2dfromld(nmax, 0, 1, y);
    DMAKE("GE", " ", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
    y = get_ldfrom2d(nmax, 0, 1, tmp2d_y);
    if (n > 1)
    {
        y[(n / 2) - 1] = zero;
        yy[(1 + Math.Abs(incy) * (n / 2 - 1)) - 1] = zero;
    }

//100
for (int ia = 1; ia <= nalf; ia = ia + 1)
{
    alpha = alf[ia - 1];
    _null = ((n <= 0) || (alpha == zero));

    // Generate the matrix A
    transl = zero;
    DMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, n - 1, n - 1, ref reset, transl);
    //nout.WriteLine("NC:{0}", nc);
    nc = nc + 1;

    // Save every datum before calling the subroutine
    uplos = uplo;
    ns = n;
    als = alpha;
    for (int i = 1; i <= laa; i = i + 1)
    {
        _as[i - 1] = aa[i - 1];
    }
    ldas = lda;
    for (int i = 1; i <= lxx; i = i + 1)
    {
        xs[i - 1] = xx[i - 1];
    }
    incxs = incx;
    for (int i = 1; i <= lyy; i = i + 1)
    {
        ys[i - 1] = yy[i - 1];
    }
    incys = incy;

    // Call the subroutine
    if (full)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})",
nc, sname, uplo, n, alpha, incx, incy, lda);
        }
        if (rewi)
        {
            //REWIND NTRA
        }

        // nout.WriteLine("BEF DSYR2");
        //
        //nout.WriteLine("NC:{0,4:D}{1,4:D}{2,4:D}{3,4:D}{4,4:D}", nc, _in, ic, ix, ia);
        /*nout.WriteLine("UPLO, N, ALPHA, INCY, INCYX, LDA");
        nout.WriteLine("{0,1}{1,4:D}{2,10:F3}{3,4:D}{4,4:D}", uplo, n, alpha, incx, lda);
        nout.WriteLine("^^^XX");

        for ( i = 1; i <= nmax*incmax; i = i + 1) {
            nout.WriteLine("{0,18:F5}", xx[i-1]);
        }

        nout.WriteLine("^^^AA");
        for ( i = 1; i <= nmax*nmax; i = i + 1) {
            nout.WriteLine("{0,18:F5}", aa[i-1]);
        }

        */
        tmp2d_aa = get_2dfromld(nmax * nmax, 0, lda, aa);
        DSYR2(uplo, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
        aa = get_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
        //nout.WriteLine("OUT DSYR2");
        /* nout.WriteLine("^^^OUT AA");
        for ( i = 1; i <= nmax*nmax; i = i + 1)
        {
            nout.WriteLine("{0,18:F5}", aa[i - 1]);
        }
        */
    }
    else if (packed)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, AP)",
sname, uplo, n, alpha, incx, incy);
        }
    }
}

```

```

    }
    if (rewi)
    {
        //REWIND NTRA
    }
    //nout.WriteLine("BEF DSPR2");
    DSPR2(uplo, n, alpha, xx, incx, yy, incy, ref aa);
    //nout.WriteLine("OUT DSPR2");
}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 160
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})          .",
nc, sname, uplo, n, alpha, incx, incy, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, AP)          .", nc,
sname, uplo, n, alpha, incx, incy);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (ns == n);
isame[2] = (als == alpha);
isame[3] = LDE(xs, xx, lx);
isame[4] = (incxs == incx);
isame[5] = LDE(ys, yy, ly);
isame[6] = (incys == incy);

if (_null)
{
    isame[7] = LDE(_as, aa, laa);
}
else
{
    isame[7] = LDERES(sname.Substring(1, 2), uplo, n, n, get_2dfromld(nmax * nmax, 0, lda, _as), get_2dfromld(nmax * nmax,
0, lda, aa), lda);
}
if (!packed)
{
    isame[8] = (ldas == lda);
}

// If data was incorrectly changed, report and return
same = true;
//40
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}

if (!same)
{
    fatal = true;
    //GO TO 160
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})          .",
nc, sname, uplo, n, alpha, incx, incy, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, AP)          .", nc,
sname, uplo, n, alpha, incx, incy);
    }
    return;
}

if (!_null)
{
    // Check the result column by column
    if (incx > 0)
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 0] = x[i - 1];
        }
    }
    else
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 0] = x[(n - i + 1) - 1];
        }
    }
    if (incy > 0)
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 1] = y[i - 1];
        }
    }
    else
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 1] = y[(n - i + 1) - 1];
        }
    }
}

```

```

    }

    ja = 1;
    //90
    for (int j = 1; j <= n; j = j + 1)
    {
        w[0] = z[j - 1, 1];
        w[1] = z[j - 1, 0];
        if (upper)
        {
            jj = 1;
            lj = j;
        }
        else
        {
            jj = j;
            lj = n - j + 1;
        }
        //DMVCH("N", lj, 2, alpha, z[jj-1,0], nmax, w, 1, one, a[jj-1,j-1], 1, yt, g, aa[ja-1], eps, err, fatal, nout,
true);

        //from DCHK5
        // DMVCH("N", lj, 1, alpha, get_2dfrom1d(nmax, jj - 1, lj, z), lj, w, 1, one, get_darr2d(jj - 1, nmax - 1, j - 1,
a),

        //1, ref yt, ref g, get_darr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);
        /* CALL DMVCH( 'N', LJ, 2, ALPHA, Z( JJ, 1 ),
$          NMAX, W, 1, ONE, A( JJ, J ), 1,
$          YT, G, AA( JA ), EPS, ERR, FATAL,
$          NOUT, .TRUE. ) */

        // static double[] get_ldfrom2d(int l, int start, int lda, double[,] arr)
        /*DMVCH("N", lj, 2, alpha, get_2dfrom1d(nmax*2, 0, nmax, get_ldfrom2d(nmax*2,jj-1,nmax,z)), nmax, w, 1, one,
get_darr2d(jj - 1, nmax - 1, j - 1, a), 1, ref yt, ref g, get_darr(ja - 1, (nmax * nmax) - 1, aa), eps, ref
err, ref fatal, nout, true);*/

        //static double[,] get_2dfrom1d(int l, int start, int lda, double[,] arr)
        //static double[,] get_ldfrom2d(int l, int start, int lda, double[,] arr)
        DMVCH("N", lj, 2, alpha, get_2dfrom1d(nmax*2, 0, nmax, get_ldfrom2d(nmax*2,jj-1,nmax, z)), nmax,
w, 1, one, get_darr2d(jj - 1, nmax - 1, j - 1, a), 1,
ref yt, ref g, get_darr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);

        //DMVCH("N", lj, 2, alpha, z[jj - 1, 0], nmax, w, 1, one, misc_d.get_arr2d(jj-1,nmax-1,j-1,a), 1, yt, g,
misc_d.get_arrd(ja - 1,nmax*nmax-1,aa), eps, err, fatal, nout, true);
        if (full)
        {
            if (upper)
            {
                ja = ja + lda;
            }
            else
            {
                ja = ja + lda + 1;
            }
        }
        else
        {
            ja = ja + lj;
        }
        errmax = Math.Max(errmax, err);

        // If got really bad answer, report and return
        if (fatal)
        {
            //GO TO 150
            nout.WriteLine("          THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            if (full)
            {
                nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})
.", nc, sname, uplo, n, alpha, incx, incy, lda);
            }
            else if (packed)
            {
                nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, AP)
.", nc, sname, uplo, n, alpha, incx, incy);
            }
            return;
        }
    } //90
}
else
{
    // Avoid repeating tests with N.le.0
    if (n <= 0)
    {
        //GO TO 140
        goto Loop140;
    }
    /* // Report result
    if (errmax < thresh)
    {
        //nout.WriteLine("^^^8");
        nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
    }
    else
    {
        nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO
(2,8:F2) - SUSPECT *****", sname, nc, errmax);
    }
    return;*/
}
}
} //100
} //110
} //120
} //130
Loop140: ;
} //140

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^9");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
}

```

```

        else
        {
            nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ((1,6:D) CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
        }
        return;
    }
    static void DCHKE(int isnum, string srnamt, StreamWriter nout)
    {
        // Tests the error exits from the Level 2 Blas. Requires a special version of the error-handling routine XERBLA. ALPHA, BETA, A, X and Y
        // should not need to be defined.
        // Auxiliary routine for test program for Level 2 Blas.

        // Local Scalars
        double alpha, beta;

        // Local Arrays
        double[,] a = new double[1, 1];
        double[] x = new double[1];
        double[] y = new double[1];

        double[] tmp1d_a;

        alpha = 0.0; //added
        beta = 0.0; // added

        // Executable Statements
        // OK is set to .FALSE. by the special version of XERBLA or by CHKXER if anything is wrong.
        ok = true;

        // LERR is set to .TRUE. by the special version of XERBLA each time it is called, and is then tested and re-set by CHKXER
        lerr = false;
        //nout.WriteLine("DCHKE: isnum={0}", isnum);
        switch (isnum)
        {
            case 1:
                //nout.WriteLine("^^^infot=1");
                infot = 1;
                DGEMV("/", 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //nout.WriteLine("^^^infot=2");
                infot = 2;
                DGEMV("N", -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //nout.WriteLine("^^^infot=3");
                infot = 3;
                DGEMV("N", 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //nout.WriteLine("^^^infot=6");
                infot = 6;
                DGEMV("N", 2, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //nout.WriteLine("^^^infot=8");
                infot = 8;
                DGEMV("N", 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //nout.WriteLine("^^^infot=11");
                infot = 11;
                DGEMV("N", 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //GO TO 170
                break;

            case 2:
                infot = 1;
                DGBMV("/", 0, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);
                // nout.Flush();
                //nout.Close();
                //Environment.Exit(1);
                infot = 2;
                DGBMV("N", -1, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 3;
                DGBMV("N", 0, -1, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 4;
                DGBMV("N", 0, 0, -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 5;
                DGBMV("N", 2, 0, 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 8;
                DGBMV("N", 0, 0, 1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 10;
                DGBMV("N", 0, 0, 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 13;
                DGBMV("N", 0, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
                CHKXER(srnamt, infot, nout, lerr, ok);
                //GO TO 170
                break;

            case 3:
                infot = 1;
                DSYMV("/", 0, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 2;
                DSYMV("U", -1, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 5;
                DSYMV("U", 2, alpha, a, 1, x, 1, beta, ref y, 1);
                CHKXER(srnamt, infot, nout, lerr, ok);

                infot = 7;
                DSYMV("U", 0, alpha, a, 1, x, 0, beta, ref y, 1);

```

```

CHKXER(srnamt, infot, nout, lerr, ok);

infot = 10;
DSYMV("U", 0, alpha, a, 1, x, 1, beta, ref y, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 4:
infot = 1;
DSBMV("/I", 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DSBMV("U", -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DSBMV("U", 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
DSBMV("U", 0, 1, alpha, a, 1, x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 8;
DSBMV("U", 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 11;
DSBMV("U", 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 5:
infot = 1;
DSPMV("/I", 0, alpha, get_darr2d(0,0,0,a), x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DSPMV("U", -1, alpha, get_darr2d(0,0,0,a), x, 1, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
DSPMV("U", 0, alpha, get_darr2d(0,0,0,a), x, 0, beta, ref y, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
DSPMV("U", 0, alpha, get_darr2d(0,0,0,a), x, 1, beta, ref y, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 6:
infot = 1;
DTRMV("/I", "N", "N", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTRMV("U", "/I", "N", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTRMV("U", "N", "/I", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTRMV("U", "N", "N", -1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
DTRMV("U", "N", "N", 2, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 8;
DTRMV("U", "N", "N", 0, a, 1, ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 7:
infot = 1;
DTBMV("/I", "N", "N", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTBMV("U", "/I", "N", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTBMV("U", "N", "/I", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTBMV("U", "N", "N", -1, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
DTBMV("U", "N", "N", 0, -1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
DTBMV("U", "N", "N", 0, 1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
DTBMV("U", "N", "N", 0, 0, a, 1, ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 8:
infot = 1;
DTPMV("/I", "N", "N", 0, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTPMV("U", "/I", "N", 0, get_darr2d(0,0,0,a), ref x, 1);

```

```

CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTPMV("U", "N", "/", 0, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTPMV("U", "N", "N", -1, get_darr2d(0,0,0, a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
DTPMV("U", "N", "N", 0, get_darr2d(0,0,0,a), ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 9:
infot = 1;
DTRSV("/", "N", "N", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTRSV("U", "/", "N", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTRSV("U", "N", "/", 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTRSV("U", "N", "N", -1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
DTRSV("U", "N", "N", 2, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 8;
DTRSV("U", "N", "N", 0, a, 1, ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 10:
infot = 1;
DTBSV("/", "N", "N", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTBSV("U", "/", "N", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTBSV("U", "N", "/", 0, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTBSV("U", "N", "N", -1, 0, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
DTBSV("U", "N", "N", 0, -1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
DTBSV("U", "N", "N", 0, 1, a, 1, ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
DTBSV("U", "N", "N", 0, 0, a, 1, ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 11:
infot = 1;
DTPSV("/", "N", "N", 0, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DTPSV("U", "/", "N", 0, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
DTPSV("U", "N", "/", 0, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
DTPSV("U", "N", "N", -1, get_darr2d(0,0,0,a), ref x, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
DTPSV("U", "N", "N", 0, get_darr2d(0,0,0,a), ref x, 0);
CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 170
break;
case 12:
infot = 1;
DGER(-1, 0, alpha, x, 1, y, 1, ref a, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
DGER(0, -1, alpha, x, 1, y, 1, ref a, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
DGER(0, 0, alpha, x, 1, y, 1, ref a, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
DGER(0, 0, alpha, x, 1, y, 0, ref a, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
DGER(2, 0, alpha, x, 1, y, 1, ref a, 1);
CHKXER(srnamt, infot, nout, lerr, ok);

```

```

        //GO TO 170
        break;
    case 13:
        infot = 1;
        DSYR("/", 0, alpha, x, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        DSYR("U", -1, alpha, x, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        DSYR("U", 0, alpha, x, 0, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 7;
        DSYR("U", 2, alpha, x, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 170
        break;
    case 14:
        infot = 1;
        tmpld a = new double[] { a[0, 0] };
        DSPR("/", 0, alpha, x, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        tmpld a = new double[] { a[0, 0] };
        DSPR("U", -1, alpha, x, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        tmpld a = new double[] { a[0, 0] };
        DSPR("U", 0, alpha, x, 0, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 170
        break;
    case 15:
        infot = 1;
        DSYR2("/", 0, alpha, x, 1, y, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        DSYR2("U", -1, alpha, x, 1, y, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        DSYR2("U", 0, alpha, x, 0, y, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 7;
        DSYR2("U", 0, alpha, x, 1, y, 0, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 9;
        DSYR2("U", 2, alpha, x, 1, y, 1, ref a, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 170
        break;
    case 16:
        infot = 1;
        tmpld a = new double[] { a[0, 0] };
        DSPR2("/", 0, alpha, x, 1, y, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        tmpld a = new double[] { a[0, 0] };
        DSPR2("U", -1, alpha, x, 1, y, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        tmpld a = new double[] { a[0, 0] };
        DSPR2("U", 0, alpha, x, 0, y, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 7;
        tmpld a = new double[] { a[0, 0] };
        DSPR2("U", 0, alpha, x, 1, y, 0, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        CHKXER(srnamt, infot, nout, lerr, ok);
        break;
    default:
        break;
}
//170
if (ok)
{
    nout.WriteLine(" {0,6} PASSED THE TESTS OF ERROR-EXITS", srnamt);
}
else
{
    nout.WriteLine("***** {0,6} FAILED THE TESTS OF ERROR-EXITS *****", srnamt);
}
return;
}
static void DMAKE(string type, string uplo, string diag, int m, int n, ref double[,] a, int nmax, ref double[] aa, int lda, int kl, int ku, ref
bool reset, double transl)
{
    // Generates values for an M by N matrix A within the bandwidth defined by KL and KU. Stores the values in the array AA in the data structure
    required
    // by the routine, with unwanted elements set to rogue value.
    //
    // TYPE is 'GE', 'GB', 'SY', 'SB', 'SP', 'TR', 'TB' OR 'TP'.

    // Parameters
    const double zero = 0.0;
    const double one = 1.0;
    const double rogue = -1e10;

```

```

/*// Scalar Arguments
double transl;
int kl, ku, lda, m, n, nmax;
bool reset;
string diag, uplo, type;

// Array Arguments
//DOUBLE PRECISION  A( NMAX, * ), AA( * )
*/
// Local Scalars
//int i, il, i2, i3, ibeg, iend, ioff, j, kk;
int il, i2, i3, ibeg, iend, ioff, kk;

bool gen, lower, sym, tri, unit, upper;

// Executable Statements
gen = (type.ToUpper().Substring(0, 1) == "G");
sym = (type.ToUpper().Substring(0, 1) == "S");
tri = (type.ToUpper().Substring(0, 1) == "T");
upper = ((sym || tri) && (uplo.ToUpper().Substring(0,1) == "U"));
lower = ((sym || tri) && (uplo.ToUpper().Substring(0,1) == "L"));
unit = (tri && (diag == "U"));
//Console.WriteLine("TYPE,UPLO,DIAG/GEN,SYM,TRI,UPPER,LOWER,UNIT");
//Console.WriteLine("{0,2}{1,2}{2,2}",type,uplo,diag);
//Console.WriteLine("{0,6}{1,6}{2,6}{3,6}{4,6}{5,6}",gen,sym,tri,upper,lower,unit);

// Generate data in array A
//20
for (int j = 1; j <= n; j = j + 1) {
    //10
    //Console.WriteLine("j={0} < {1}", j, n);
    for (int i = 1; i <= m; i = i + 1) {
        //Console.WriteLine("i={0} < {1}", j, m);
        if (gen || (upper && (i <= j)) || (lower && (i >= j))) {
            if (((i <= j) && ((j - i) <= ku)) || ((i >= j) && ((i - j) <= kl))) {
                a[i - 1, j - 1] = DBEG(ref reset) + transl;
            } else {
                a[i - 1, j - 1] = zero;
            }
            if (i != j) {
                if (sym) {
                    a[j - 1, i - 1] = a[i - 1, j - 1];
                } else if (tri) {
                    a[j - 1, i - 1] = zero;
                }
            }
        }
        //Console.WriteLine("end i={0} < {1}", j, m);
    } //10
    if (tri) {
        a[j - 1, j - 1] = a[j - 1, j - 1] + one;
    }
    if (unit) {
        a[j - 1, j - 1] = one;
    }
    //Console.WriteLine("end j={0} < {1}", j, n);
} //20
//Console.WriteLine("done main j");
// Store elements in array AS in data structure required by routine
if (type == "GB") {
    //50
    for (int j = 1; j <= n; j = j + 1) {
        //Console.WriteLine("j2={0} < {1}", j, n);
        for (int i = 1; i <= m; i = i + 1) {
            aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
        }
        for (int i = m + 1; i <= lda; i = i + 1) {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
    }
}
else if (type == "GB") {
    //Console.WriteLine("nmax={0}", nmax);
    //90
    for (int j = 1; j <= n; j = j + 1) {
        //Console.WriteLine("j={0} < {1}", j, n);
        //60
        for (il = 1; il <= (ku + 1 - j); il = il + 1) {
            //Console.WriteLine("il={0} < {1}", il, ku+1-j);
            aa[(il + (j - 1) * lda) - 1] = rogue;
        }
        //70
        for (int i2 = il; i2 <= Math.Min(kl + ku + 1, ku + 1 + m - j); i2 = i2 + 1) {
            for (i2 = il; i2 <= Math.Min(kl + ku + 1, ku + 1 + m - j); i2 = i2 + 1) {
                //Console.WriteLine("i2={0} < {1}", i2, Math.Min(kl + ku + 1, ku + 1 + m - j));
                aa[(i2 + (j - 1) * lda) - 1] = a[(i2 + j - ku - 1) - 1, j - 1];
            }
        }
        //80
        for (i3 = i2; i3 <= lda; i3 = i3 + 1) {
            //Console.WriteLine("i3={0} < {1}", i3, lda);
            aa[(i3 + (j - 1) * lda) - 1] = rogue;
        }
    }
}
else if ((type == "SY") || (type == "TR")) {
    //130
    for (int j = 1; j <= n; j = j + 1) {
        if (upper) {
            ibeg = 1;
            if (unit) {
                iend = j - 1;
            } else {
                iend = j;
            }
        } else {
            if (unit) {
                ibeg = j + 1;
            } else {
                ibeg = j;
            }
            iend = n;
        }
        for (int i = 1; i <= (ibeg - 1); i = i + 1) {

```

```

        aa[(i + (j - 1) * lda) - 1] = rogue;
    }
    for (int i = ibeg; i <= iend; i = i + 1) {
        aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
    }
    for (int i = (iend + 1); i <= lda; i = i + 1) {
        aa[(i + (j - 1) * lda) - 1] = rogue;
    }
} //130
} else if ((type == "SB") || (type == "TB")) {
    //170
    for (int j = 1; j <= n; j = j + 1) {
        if (upper) {
            kk = kl + 1;
            ibeg = Math.Max(1, kl + 2 - j);
            if (unit) {
                iend = kl;
            } else {
                iend = kl + 1;
            }
        } else {
            kk = 1;
            if (unit) {
                ibeg = 2;
            } else {
                ibeg = 1;
            }
            iend = Math.Min(kl + 1, 1 + m - j);
        }
        for (int i = 1; i <= (ibeg - 1); i = i + 1) {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
        for (int i = ibeg; i <= iend; i = i + 1) {
            aa[(i + (j - 1) * lda) - 1] = a[(i + j - kk) - 1, j - 1];
        }
        for (int i = (iend + 1); i <= lda; i = i + 1) {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
    } //170
} else if ((type == "SP") || (type == "TP")) {
    ioff = 0;
    //190
    for (int j = 1; j <= n; j = j + 1) {
        if (upper) {
            ibeg = 1;
            iend = j;
        } else {
            ibeg = j;
            iend = n;
        }
        //180
        for (int i = ibeg; i <= iend; i = i + 1) {
            ioff = ioff + 1;
            aa[ioff - 1] = a[i - 1, j - 1];
            if (i == j) {
                if (unit) {
                    aa[ioff - 1] = rogue;
                }
            }
        }
    }
}
}
}
static void DMVCH(string trans, int m, int n, double alpha, double[,] a, int nmax, double[] x, int incx, double beta, double[] y, int incy, ref
double[] yt, ref double[] g, double[] yy, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv)
{
    // Checks the results of the computational tests

    // Parameters
    const double zero = 0.0;
    const double one = 1.0;

    // Array Arguments ..
    //DOUBLE PRECISION  A( NMAX, * ), G( * ), X( * ), Y( * ), YT( * ), YY( * )

    // Local Scalars
    double err1;
    //int i, incxl, incyl, iy, j, jx, kx, ky, ml, nl;
    int incxl, incyl, iy, jx, kx, ky, ml, nl;
    bool tran;

    // Executable Statements
    //nout.WriteLine("Rank:{0,4:D} D1:{1,4:D} D2:{2,4:D} NMAX:{3,4:D}", a.Rank, a.GetLength(0), a.GetLength(1), nmax);
    /*for (int i = 1; i <= nmax; i = i + 1)
    {
        nout.WriteLine("{0,4:D}{1,18:F5}", i, a[i - 1, 0]);
    }*/
    tran = (trans.ToUpper().Substring(0,1) == "T") || (trans.ToUpper().Substring(0,1) == "C");
    if (tran) {
        ml = n;
        nl = m;
    } else {
        ml = m;
        nl = n;
    }
    if (incx < 0)
    {
        kx = nl;
        incxl = -1;
    } else {
        kx = 1;
        incxl = 1;
    }
    if (incy < 0)
    {
        ky = ml;
        incyl = -1;
    } else {
        ky = 1;
        incyl = 1;
    }
}

```



```

        return lde;
    }
}
lde = true;
return lde;
}
static bool LDERES(string type, string uplo, int m, int n, double[,] aa, double[,] _as, int lda)
{
    //      DOUBLE PRECISION  AA( LDA, * ), AS( LDA, * )
    // Tests if selected elements in two arrays are equal
    // TYPE is 'GE', 'SY' or 'SP'

    // Local Scalars
    //int i, ibeg, iend, j;
    bool lderes;
    int ibeg, iend;
    bool upper;

    // Executable Statements
    upper = (uplo == "U");
    if (type == "GE")
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = m + 1; i <= lda; i = i + 1)
            {
                if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
                {
                    //GO TO 70
                    lderes = false;
                    return lderes;
                }
            }
        }
    }
    else if (type == "SY")
    {
        //50
        for (int j = 1; j <= n; j = j + 1)
        {
            if (upper)
            {
                {
                    ibeg = 1;
                    iend = j;
                }
                else
                {
                    {
                        ibeg = j;
                        iend = n;
                    }
                }
                //30
                for (int i = 1; i <= ibeg - 1; i = i + 1)
                {
                    if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
                    {
                        //GO TO 70
                        lderes = false;
                        return lderes;
                    }
                }
                //40
                for (int i = iend + 1; i <= lda; i = i + 1)
                {
                    if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
                    {
                        //GO TO 70
                        lderes = false;
                        return lderes;
                    }
                }
            } //50
        }

        lderes = true;
        return lderes;
    }
}
static double DBEG(ref bool reset)
{
    // Generates random numbers uniformly distributed between -0.5 and 0.5

    // Local Scalars
    //static int i, ic, mi; see above

    // Executable Statements
    if (reset)
    {
        // Initialize local variables
        mi = 891;
        i = 7;
        ic = 0;
        reset = false;
    }

    // The sequence of values of I is bounded between 1 and 999.
    // If initial I = 1,2,3,6,7 or 9, the period will be 50.
    // If initial I = 4 or 8, the period will be 25.
    // If initial I = 5, the period will be 10.
    // IC is used to break up the period by skipping 1 value of I in 6.

    ic = ic + 1;

    do
    {
        //Console.WriteLine("in do loop DBEG ic={0}",ic);
        i = i * mi;
        i = i - 1000 * (i / 1000);
        if (ic >= 5)
        {
            {
                ic = 0;
                continue;
            }
            else{
                break;
            }
        }
    } while (true);
}

```

```

        return (double)(i - 500) / 1001.0;
    }
    static double DDIFF(double x, double y)
    {
        // Auxiliary routine for test program for Level 2 Blas
        // Executable Statements
        return (x - y);
    }

    public static void CHKXER(string srnamt, int infot, StreamWriter nout, bool lerr, bool ok)
    {
        // Tests whether XERBLA has detected an error when it should

        if (!lerr)
        {
            nout.WriteLine("***** ILLEGAL VALUE OF PARAMETER NUMBER (0,2:D) NOT DETECTED BY (1,6) *****", infot, srnamt);
            ok = false;
        }
        lerr = false;
    } //used in misc_z

    public static void XERBLA(string sname, int info)
    {
        // This is a special version of XERBLA to be used only as part of the test program for testing error exits from the Level 2 BLAS routines.
        // XERBLA is an error handler for the Level 2 BLAS routines. It is called by the Level 2 BLAS routines if an input parameter is invalid.
        /*
         * Scalars in Common ..
         *   INTEGER      INFOT, NOUT
         *   LOGICAL      LERR, OK
         *   CHARACTER*6   SRNAMT*/

        // Executable Statements
        lerr = true;
        //nout.WriteLine("in:sname={0} info={1} :infot={2} srnamt={3}", sname, info, infot, srnamt);
        if (info != infot)
        {
            if (infot != 0)
            {
                nout.WriteLine("***** XERBLA WAS CALLED WITH INFO = (0,6:D) INSTEAD OF (1,2:D) *****", info, infot);
            }
            else
            {
                nout.WriteLine("***** XERBLA WAS CALLED WITH INFO = (0,6:D) *****", info);
            }
            ok = false;
        }

        if (sname != srnamt)
        {
            nout.WriteLine("***** XERBLA WAS CALLED WITH SRNAME = (0,6) INSTEAD OF (1,6) *****", sname, srnamt);
            ok = false;
        }
    } //separate for misc_z

    static void _LSAME()
    {
        /*// switch (cmach.ToUpper().Substring(0,1))
        {
            case "E":
                rmach = eps;
                break;
            case "S":
                rmach = sfmin;*/
    }

    public static void DGBMV(string trans, int m, int n, int kl, int ku, double alpha, double[,] a, int lda, double[] x, int incx, double beta, ref
double[] y, int incy)
    {
        // DGBMV performs one of the matrix-vector operations
        //
        // y := alpha*A*x + beta*y, or y := alpha*A**T*x + beta*y,
        // where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.
        //
        // On entry, TRANS specifies the operation to be performed as follows:
        // TRANS = 'N' or 'n' y := alpha*A*x + beta*y.
        // TRANS = 'T' or 't' y := alpha*A**T*x + beta*y.
        // TRANS = 'C' or 'c' y := alpha*A**T*x + beta*y.
        //
        // On entry, M specifies the number of rows of the matrix A. M must be at least zero.
        // On entry, N specifies the number of columns of the matrix A. N must be at least zero.
        // On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.
        // On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.
        // On entry, ALPHA specifies the scalar alpha.
        //
        // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
        // Before entry, the leading ( kl + ku + 1 ) by n part of the array A must contain the matrix of coefficients, supplied column by column, with
the leading diagonal of the matrix in
        // row ( ku + 1 ) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row
( ku + 2 ),
        // and so on.
        // Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced.
        // The following program segment will transfer a band matrix from conventional full matrix storage to band storage:
        //
        //       DO 20, J = 1, N
        //       K = KU + 1 - J
        //       DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
        //       A( K + I, J ) = matrix( I, J )
        //       10 CONTINUE
        //       20 CONTINUE
        //
        // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( kl + ku + 1 ).
        //
        // X is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( m - 1 ) * abs(
INCX ) ) otherwise.
        // Before entry, the incremented array X must contain the vector x.
        //
        // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
        // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.
        //
        // Y is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( n - 1 ) * abs(
INCY ) ) otherwise.
        // Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
        //
        // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.
        //
        // Level 2 Blas routine. The vector and matrix arguments are not referenced when N = 0, or M = 0
        //
        // Parameters
        const double one = 1.0;
        const double zero = 0.0;
    }

```

```

// Local Scalars
double temp;
//int i, info, ix, iy, j, jx, jy, k, kupl, kx, ky, lenx, leny;
int info, ix, iy, jx, jy, k, kupl, kx, ky, lenx, leny;

// Test the input parameters

info = 0;
if (!(trans.ToUpper().Substring(0, 1) == "N")) && (!(trans.ToUpper().Substring(0, 1) == "T")) && (!(trans.ToUpper().Substring(0, 1) == "C"))
{
    info = 1;
}
else if (m < 0)
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (kl < 0)
{
    info = 4;
}
else if (ku < 0)
{
    info = 5;
}
else if (lda < (kl + ku + 1))
{
    info = 8;
}
else if (incx == 0)
{
    info = 10;
}
else if (incy == 0)
{
    info = 13;
}
if (info != 0)
{
    XERBLA("DGBMV ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

// Set LENX and LENY, the lengths of the vectors x and y, and set up the start points in X and Y
if (trans.ToUpper().Substring(0, 1) == "N")
{
    lenx = n;
    leny = m;
}
else
{
    lenx = m;
    leny = n;
}
if (incx > 0)
{
    kx = 1;
}
else
{
    kx = 1 - (lenx - 1) * incx;
}
if (incy > 0)
{
    ky = 1;
}
else
{
    ky = 1 - (leny - 1) * incy;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the band part of A.

// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
    }
}

```

```

        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = beta * y[iy - 1];
                iy = iy + incy;
            }
        }
    }
    if (alpha == zero)
    {
        return;
    }
    kupl = ku + 1;
    if (trans.ToUpper().Substring(0, 1) == "N")
    {
        // Form y := alpha*A*x + y.
        jx = kx;
        if (incy == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[jx - 1] != zero)
                {
                    temp = alpha * x[jx - 1];
                    k = kupl - j;
                    for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                    {
                        y[i - 1] = y[i - 1] + temp * a[k + i - 1, j - 1];
                    }
                }
                jx = jx + incx;
            }
        }
        else {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[jx - 1] != zero)
                {
                    temp = alpha * x[jx - 1];
                    iy = ky;
                    k = kupl - j;
                    for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                    {
                        y[iy - 1] = y[iy - 1] + temp * a[k + i - 1, j - 1];
                        iy = iy + incy;
                    }
                }
                jx = jx + incx;
                if (j > ku)
                {
                    ky = ky + incy;
                }
            }
        }
    }
    else {
        // Form y := alpha*A**T*x + y.
        jy = ky;
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = zero;
                k = kupl - j;
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + a[k + i - 1, j - 1] * x[i - 1];
                }
                y[jy - 1] = y[jy - 1] + alpha * temp;
                jy = jy + incy;
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = zero;
                ix = kx;
                k = kupl - j;
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + a[k + i - 1, j - 1] * x[ix - 1];
                    ix = ix + incx;
                }
                y[jy - 1] = y[jy - 1] + alpha * temp;
                jy = jy + incy;
                if (j > ku)
                {
                    kx = kx + incx;
                }
            }
        }
    }
}

public static void DGEMV(string trans, int m, int n, double alpha, double[, ] a, int lda, double[] x, int incx, double beta, ref double[] y, int incy)
{
    // DGEMV performs one of the matrix-vector operations
    //
    // y := alpha*A*x + beta*y, or y := alpha*A**T*x + beta*y,
    //
    // where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
    //
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' y := alpha*A*x + beta*y.
    // TRANS = 'T' or 't' y := alpha*A**T*x + beta*y.
    // TRANS = 'C' or 'c' y := alpha*A**T*x + beta*y.
    // On entry, M specifies the number of rows of the matrix A. M must be at least zero.
    // On entry, N specifies the number of columns of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.
    //
    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry, the leading m by n part of the array A must contain the matrix of coefficients.
    //
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, m ).

```

```

// X is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( m - 1 ) * abs(
INCX ) ) otherwise.
// Before entry, the incremented array X must contain the vector x.

// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
// On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

// Y is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( n - 1 ) * abs(
INCY ) ) otherwise.
// Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

// On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

// Level 2 Blas routine.
// The vector and matrix arguments are not referenced when N = 0, or M = 0

// Parameters
const double one = 1.0;
const double zero = 0.0;

// Local Scalars
double temp;
//int i, info, ix, iy, j, jx, jy, kx, ky, lenx, leny;
int info, ix, iy, jx, jy, kx, ky, lenx, leny;

// Test the input parameters
info = 0;
if ( (! (trans.ToUpper().Substring(0,1) == "N")) && (! (trans.ToUpper().Substring(0,1) == "T")) &&
    (! (trans.ToUpper().Substring(0,1) == "C"))) {
    info = 1;
} else if (m < 0) {
    info = 2;
} else if (n < 0) {
    info = 3;
} else if (lda < Math.Max(1,m)) {
    info = 6;
} else if (incx == 0) {
    info = 8;
} else if (incy == 0) {
    info = 11;
}
if (info != 0) {
    XERBLA("DGEMV ",info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

// Set LENX and LENY, the lengths of the vectors x and y, and set up the start points in X and Y
if (trans.ToUpper().Substring(0, 1) == "N")
{
    lenx = n;
    leny = m;
}
else
{
    lenx = m;
    leny = n;
}
if (incx > 0)
{
    kx = 1;
}
else
{
    kx = 1 - (lenx - 1) * incx;
}
if (incy > 0)
{
    ky = 1;
}
else
{
    ky = 1 - (leny - 1) * incy;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the band part of A.

// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
    }
}

```

```

        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = beta * y[iy - 1];
                iy = iy + incy;
            }
        }
    }
}
if (alpha == zero)
{
    return;
}
if (trans.ToUpper().Substring(0, 1) == "N")
{
    // Form y := alpha*A*x + y.
    jx = kx;
    if (incy == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                for (int i = 1; i <= m; i = i + 1)
                {
                    y[i - 1] = y[i - 1] + temp * a[i - 1, j - 1];
                }
            }
            jx = jx + incx;
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                iy = ky;
                for (int i = 1; i <= m; i = i + 1)
                {
                    y[iy - 1] = y[iy - 1] + temp * a[i - 1, j - 1];
                    iy = iy + incy;
                }
            }
            jx = jx + incx;
        }
    }
}
else {
    // Form y := alpha*A**T*x + y.
    jy = ky;
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = zero;
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = temp + a[i - 1, j - 1] * x[i - 1];
            }
            y[jy - 1] = y[jy - 1] + alpha * temp;
            jy = jy + incy;
        }
    }
    else {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = zero;
            ix = kx;
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = temp + a[i - 1, j - 1] * x[ix - 1];
                ix = ix + incx;
            }
            y[jy - 1] = y[jy - 1] + alpha * temp;
            jy = jy + incy;
        }
    }
}
}
}
public static void DGER(int m, int n, double alpha, double[] x, int incx, double[] y, int incy, ref double[,] a, int lda)
{
    // DGER performs the rank 1 operation
    //
    // A := alpha*x*y**T + A,
    //
    // where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.
    //
    // On entry, M specifies the number of rows of the matrix A. M must be at least zero.
    // On entry, N specifies the number of columns of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.
    //
    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( m - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the m element vector x.
    //
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
    //
    // Y is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.
    //
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.
    //
    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated
matrix.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, m ).
    //
    // Parameters

```

```

const double zero = 0.0;

// Local Scalars
double temp;
int info, ix, jy, kx;

// Test the input parameters
info = 0;
if (m < 0) {
    info = 1;
} else if (n < 0) {
    info = 2;
} else if (incx == 0) {
    info = 5;
} else if (incy == 0) {
    info = 7;
} else if (lda < Math.Max(1,m)) {
    info = 9;
}
if (info != 0) {
    XERBLA("DGER ",info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || (alpha == zero)) {
    return;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if (incy > 0) {
    jy = 1;
} else {
    jy = 1 - (n-1)*incy;
}
if (incx == 1) {
    for (int j = 1; j <= n; j = j + 1) {
        if (y[jy-1] != zero) {
            temp = alpha*y[jy-1];
            for (int i = 1; i <= m; i = i + 1) {
                a[i-1,j-1] = a[i-1,j-1] + x[i-1]*temp;
            }
            jy = jy + incy;
        }
    }
} else {
    if (incx > 0) {
        kx = 1;
    } else {
        kx = 1 - (m-1)*incx;
    }
    for (int j = 1; j <= n; j = j + 1) {
        if (y[jy-1] != zero) {
            temp = alpha*y[jy-1];
            ix = kx;
            for (int i = 1; i <= m; i = i + 1) {
                a[i-1,j-1] = a[i-1,j-1] + x[ix-1]*temp;
                ix = ix + incx;
            }
            jy = jy + incy;
        }
    }
}
}

public static void DSBMV(string uplo, int n, int k, double alpha, double[,] a, int lda, double[] x, int incx, double beta, ref double[] y, int incy)
{
    // DSBMV performs the matrix-vector operation
    //
    // y := alpha*A*x + beta*y,
    //
    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric band matrix, with k super-diagonals.
    //
    // On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:
    //
    //         UPLO = 'U' or 'u'   The upper triangular part of A is being supplied.
    //         UPLO = 'L' or 'l'   The lower triangular part of A is being supplied.
    //
    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.
    // On entry, ALPHA specifies the scalar alpha.
    //
    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
    symmetric matrix, supplied column by
    // column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
    so on. The top left k by k triangle
    // of the array A is not referenced.
    // The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to
    band storage:
    //
    //         DO 20, J = 1, N
    //             M = K + 1 - J
    //             DO 10, I = MAX( 1, J - K ), J
    //                 A( M + I, J ) = matrix( I, J )
    //             10 CONTINUE
    //         20 CONTINUE
    //
    // Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
    symmetric matrix, supplied column by
    // column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
    The bottom right k by k triangle of the
    // array A is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from
    conventional full matrix storage to band storage:
    //
    //         DO 20, J = 1, N
    //             M = 1 - J
    //             DO 10, I = J, MIN( N, J + K )
    //                 A( M + I, J ) = matrix( I, J )
    //             10 CONTINUE
    //         20 CONTINUE
    //
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).

```

```

// X is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the vector x.

// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
// On entry, BETA specifies the scalar beta.

// Y is DOUBLE PRECISION array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCY ) ).
// Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

// On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

// The vector and matrix arguments are not referenced when N = 0, or M = 0

// Parameters
const double one = 1.0;
const double zero = 0.0;

// Local Scalars
double temp1, temp2;
int info, ix, iy, jx, jy, kplus1, kx, ky, l;

// Test the input parameters
info = 0;
if ( !(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
    info = 1;
} else if ( n < 0 ) {
    info = 2;
} else if ( k < 0 ) {
    info = 3;
} else if ( lda < (k+1) ) {
    info = 6;
} else if ( incx == 0 ) {
    info = 8;
} else if ( incy == 0 ) {
    info = 11;
}
if ( info != 0 ) {
    XERBLA("DSBMV ",info);
    return;
}

// Quick return if possible
if ( (n == 0) || ((alpha == zero) && (beta == one)) ) {
    return;
}

// Set up the start points in X and Y
if ( incx > 0 ) {
    kx = 1;
} else {
    kx = 1 - (n-1)*incx;
}
if ( incy > 0 ) {
    ky = 1;
} else {
    ky = 1 - (n-1)*incy;
}

// Start the operations. In this version the elements of the array A are accessed sequentially with one pass through A.

// First form y := beta*y.
if (beta != one) {
    if (incy == 1) {
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = zero;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = beta*y[i-1];
            }
        }
    } else {
        iy = ky;
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = zero;
                iy = iy + incy;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = beta*y[iy-1];
                iy = iy + incy;
            }
        }
    }
}
if (alpha == zero) {
    return;
}
if (uplo.ToUpper().Substring(0,1) == "U") {
    // Form y when upper triangle of A is stored.
    kplus1 = k + 1;
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            l = kplus1 - j;
            for (int i = Math.Max(1,j-k); i <= j-1; i = i + 1) {
                y[i-1] = y[i-1] + temp1*a[l+i-1,j-1];
                temp2 = temp2 + a[l+i-1,j-1]*x[i-1];
            }
            y[j-1] = y[j-1] + temp1*a[kplus1-1,j-1] + alpha*temp2;
        }
    } else {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            ix = kx;
            iy = ky;

```

```

        l = kplus1-j;
        for (int i = Math.Max(1,j-k); i <= j-1; i = i + 1) {
            y[iy-1] = y[iy-1] + temp1*a[l+i-1,j-1];
            temp2 = temp2 + a[l+i-1,j-1]*x[ix-1];
            ix = ix + incx;
            iy = iy + incy;
        }
        y[jy-1] = y[jy-1] + temp1*a[kplus1-1,j-1] + alpha*temp2;
        jx = jx + incx;
        jy = jy + incy;
        if (j > k) {
            kx = kx + incx;
            ky = ky + incy;
        }
    }
} else {
    // Form y when lower triangle of A is stored.
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            y[j-1] = y[j-1] + temp1*a[0,j-1];
            l = 1-j;
            for (int i = j+1; i <= Math.Min(n,j+k); i = i + 1) {
                y[i-1] = y[i-1] + temp1*a[l+i-1,j-1];
                temp2 = temp2 + a[l+i-1,j-1]*x[i-1];
            }
            y[j-1] = y[j-1] + alpha*temp2;
        }
    } else {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            y[jy-1] = y[jy-1] + temp1*a[0,j-1];
            l = 1-j;
            ix = jx;
            iy = jy;
            for (int i = j+1; i <= Math.Min(n,j+k); i = i + 1) {
                ix = ix + incx;
                iy = iy + incy;
                y[iy-1] = y[iy-1] + temp1*a[l+i-1,j-1];
                temp2 = temp2 + a[l+i-1,j-1]*x[ix-1];
            }
            y[jy-1] = y[jy-1] + alpha*temp2;
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}
}

public static void DSPMV(string uplo, int n, double alpha, double[] ap, double[] x, int incx, double beta, ref double[] y, int incy)
{
    // DSPMV performs the matrix-vector operation
    // y := alpha*A*x + beta*y,
    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.
    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:
    //
    //         UPLO = 'U' or 'u'   The upper triangular part of A is supplied in AP.
    //         UPLO = 'L' or 'l'   The lower triangular part of A is supplied in AP.
    //
    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.
    //
    // AP is DOUBLE PRECISION array of DIMENSION at least ( ( n*( n + 1 ) )/2 ).
    // Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column
    by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. Before entry with UPLO = 'L' or 'l', the
    array AP must
    // contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2 )
    and AP( 3 ) contain a( 2, 1 )
    // and a( 3, 1 ) respectively, and so on.
    //
    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.
    //
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.
    //
    // Y is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.
    //
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.
    //
    // The vector and matrix arguments are not referenced when N = 0, or M = 0
    //
    // Parameters
    const double one = 1.0;
    const double zero = 0.0;

    // Local Scalars ..
    double temp1, temp2;
    int info, ix, iy, jx, jy, k, kk, kx, ky;

    // Test the input parameters
    info = 0;
    if (!(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L")) {
        info = 1;
    } else if (n < 0) {
        info = 2;
    } else if (incx == 0) {
        info = 6;
    } else if (incy == 0) {
        info = 9;
    }
    if (info != 0) {
        XERBLA("DSPMV ",info);
        return;
    }
}

```

```

}

// Quick return if possible.
if ((n == 0) || ((alpha == 0) && (beta == one))) {
    return;
}

// Set up the start points in X and Y
if (incx > 0) {
    kx = 1;
} else {
    kx = 1-(n-1)*incx;
}
if (incy > 0) {
    ky = 1;
} else {
    ky = 1-(n-1)*incy;
}

// Start the operations. In this version the elements of the array AP are accessed sequentially with one pass through AP.

// First form y := beta*y.
if (beta != one) {
    if (incy == 1) {
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = 0;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = beta*y[i-1];
            }
        }
    } else {
        iy = ky;
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = zero;
                iy = iy + incy;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = beta*y[iy-1];
                iy = iy + incy;
            }
        }
    }
}

if (alpha == zero) {
    return;
}

kk = 1;
if (uplo.ToUpper().Substring(0,1) == "U") {
    // Form y when AP contains the upper triangle.
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            k = kk;
            for (int i = 1; i <= (j-1); i = i + 1) {
                y[i-1] = y[i-1] + temp1*ap[k-1];
                temp2 = temp2 + ap[k-1]*x[i-1];
                k = k + 1;
            }
            y[j-1] = y[j-1] + temp1*ap[(kk+j-1)-1] + alpha*temp2;
            kk = kk + j;
        }
    } else {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            ix = kx;
            iy = ky;
            for (k = kk; k <= (kk+j-2); k = k + 1) {
                y[iy-1] = y[iy-1] + temp1*ap[k-1];
                temp2 = temp2 + ap[k-1]*x[ix-1];
                ix = ix + incx;
                iy = iy + incy;
            }
            y[jy-1] = y[jy-1] + temp1*ap[(kk+j-1)-1] + alpha*temp2;
            jx = jx + incx;
            jy = jy + incy;
            kk = kk + j;
        }
    }
} else {
    // Form y when AP contains the lower triangle.
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            y[j-1] = y[j-1] + temp1*ap[kk-1];
            k = kk + 1;
            for (int i = (j+1); i <= n; i = i + 1) {
                y[i-1] = y[i-1] + temp1*ap[k-1];
                temp2 = temp2 + ap[k-1]*x[i-1];
                k = k + 1;
            }
            y[j-1] = y[j-1] + alpha*temp2;
            kk = kk + (n-j+1);
        }
    } else {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            y[jy-1] = y[jy-1] + temp1*ap[kk-1];
            ix = jx;
            iy = jy;
            for (k = (kk + 1); k <= (kk + n - j); k = k + 1) {
                ix = ix + incx;
            }
        }
    }
}

```

```

        iy = iy + incy;
        y[iy-1] = y[iy-1] + temp1*ap[k-1];
        temp2 = temp2 + ap[k-1]*x[ix-1];
    }
    y[jy-1] = y[jy-1] + alpha*temp2;
    jx = jx + incx;
    jy = jy + incy;
    kk = kk + (n-j+1);
}
}

}

public static void DSPR(string uplo, int n, double alpha, double[] x, int incx, ref double[] ap)
{
    // DSPR    performs the symmetric rank 1 operation

    // A := alpha*x*x**T + A,

    // where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

    //          UPLO = 'U' or 'u'   The upper triangular part of A is supplied in AP.
    //          UPLO = 'L' or 'l'   The lower triangular part of A is supplied in AP.

    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.

    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // AP is DOUBLE PRECISION array of DIMENSION at least ( ( n * ( n + 1 ) ) / 2 ).
    // Before entry with  UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially,
    // column by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. On exit, the array AP is overwritten by the
    // upper triangular part of the
    // updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed
    // sequentially, column by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respectively, and so on. On exit, the array AP is overwritten by the
    // lower triangular part of the
    // updated matrix.

    // Parameters
    const double zero = 0.0;

    // Local Scalars
    double temp;
    int info, ix, jx, k, kk, kx;
    kx = 0; // added
    // Test the input parameters
    info = 0;
    if ( !(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
        info = 1;
    } else if ( n < 0 ) {
        info = 2;
    } else if ( incx == 0 ) {
        info = 5;
    }
    if ( info != 0 ) {
        XERBLA("DSPR ", info);
        return;
    }

    // Quick return if possible
    if ( (n == 0) || (alpha == zero) ) {
        return;
    }

    // Set the start point in X if the increment is not unity
    if ( incx <= 0 ) {
        kx = 1 - (n-1) * incx;
    } else if ( incx != 1 ) {
        kx = 1;
    }

    // Start the operations. In this version the elements of the array AP are accessed sequentially with one pass through AP.
    kk = 1;
    if (uplo.ToUpper().Substring(0,1) == "U") {
        // Form A when upper triangle is stored in AP.
        if ( incx == 1 ) {
            for ( int j = 1; j <= n; j = j + 1 ) {
                if ( x[j-1] != zero ) {
                    temp = alpha*x[j-1];
                    k = kk;
                    for ( int i = 1; i <= j; i = i + 1 ) {
                        ap[k-1] = ap[k-1] + x[i-1]*temp;
                        k = k + 1;
                    }
                    kk = kk + j;
                }
            }
        } else {
            jx = kx;
            for ( int j = 1; j <= n; j = j + 1 ) {
                if ( x[jx-1] != zero ) {
                    temp = alpha*x[jx-1];
                    ix = kx;
                    for ( k = kk; k <= (kk + j - 1); k = k + 1 ) {
                        ap[k-1] = ap[k-1] + x[ix-1]*temp;
                        ix = ix + incx;
                    }
                    jx = jx + incx;
                    kk = kk + j;
                }
            }
        }
    } else {
        // Form A when lower triangle is stored in AP.
        if ( incx == 1 ) {
            for ( int j = 1; j <= n; j = j + 1 ) {

```

```

        if (x[j-1] != zero) {
            temp = alpha*x[j-1];
            k = kk;
            for (int i = j; i <= n; i = i + 1) {
                ap[k-1] = ap[k-1] + x[i-1]*temp;
                k = k + 1;
            }
            kk = kk + n-j+1;
        }
    } else {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1) {
            if (x[jx-1] != zero) {
                temp = alpha*x[jx-1];
                ix = jx;
                for (k = kk; k <= (kk + n - j); k = k + 1) {
                    ap[k-1] = ap[k-1] + x[ix-1]*temp;
                    ix = ix + incx;
                }
            }
            jx = jx + incx;
            kk = kk + n-j+1;
        }
    }
}

}

public static void DSPR2(string uplo, int n, double alpha, double[] x, int incx, double[] y, int incy, ref double[] ap)
{
    // DOUBLE PRECISION AP(*),X(*),Y(*)

    // DSPR2 performs the symmetric rank 2 operation

    // A := alpha*x*y**T + alpha*y*x**T + A,

    // where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

    //          UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.
    //          UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.
    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.

    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Y is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.

    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // AP is DOUBLE PRECISION array of DIMENSION at least ( ( n * ( n + 1 ) ) / 2 ).
    // Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially,
    column by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. On exit, the array AP is overwritten by the
    upper triangular part of the
    // updated matrix.
    // Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column
    by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respectively, and so on. On exit, the array AP is overwritten by the
    lower triangular part of the
    // updated matrix.

    // Parameters
    const double zero = 0.0;

    // Local Scalars
    double temp1, temp2;
    int info, ix, iy, jx, jy, k, kk, kx, ky;
    kx = 0; // added
    ky = 0; // added
    jx = 0; // added
    jy = 0; // added
    // Test the input parameters
    info = 0;
    if ( (!uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
        info = 1;
    } else if ( n < 0 ) {
        info = 2;
    } else if ( incx == 0 ) {
        info = 5;
    } else if ( incy == 0 ) {
        info = 7;
    }
    if ( info != 0 ) {
        XERBLA("DSPR2 ",info);
        return;
    }

    // Quick return if possible
    if ( (n == 0) || (alpha == zero) ) {
        return;
    }

    // Set up the start points in X and Y if the increments are not both unity
    if ( (incx != 1) || (incy != 1) ) {
        if (incx > 0) {
            kx = 1;
        } else {
            kx = 1 - (n-1)*incx;
        }

        if (incy > 0) {
            ky = 1;
        } else {
            ky = 1 - (n-1)*incy;
        }
        jx = kx;
        jy = ky;
    }
}

```

```

    }

    // Start the operations. In this version the elements of the array AP are accessed sequentially with one pass through AP.
    kk = 1;
    if (uplo.ToUpper().Substring(0,1) == "U") {
        // Form A when upper triangle is stored in AP
        if ((incx == 1) && (incy == 1)) {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[j-1] != zero) || (y[j-1] != zero)) {
                    temp1 = alpha*y[j-1];
                    temp2 = alpha*x[j-1];
                    k = kk;
                    for (int i = 1; i <= j; i = i + 1) {
                        ap[k-1] = ap[k-1] + x[i-1]*temp1+y[i-1]*temp2;
                        k = k + 1;
                    }
                    kk = kk + j;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[jx-1] != zero) || (y[jy-1] != zero)) {
                    temp1 = alpha*y[jy-1];
                    temp2 = alpha*x[jx-1];
                    ix = kx;
                    iy = ky;
                    for (k = kk; k <= (kk+j-1); k = k + 1) {
                        ap[k-1] = ap[k-1] + x[ix-1]*temp1+y[iy-1]*temp2;
                        ix = ix + incx;
                        iy = iy + incy;
                    }
                    jx = jx + incx;
                    jy = jy + incy;
                    kk = kk + j;
                }
            }
        }
    } else {
        // Form A when lower triangle is stored in AP
        if ((incx == 1) && (incy == 1)) {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[j-1] != zero) || (y[j-1] != zero)) {
                    temp1 = alpha*y[j-1];
                    temp2 = alpha*x[j-1];
                    k = kk;
                    for (int i = j; i <= n; i = i + 1) {
                        ap[k-1] = ap[k-1] + x[i-1]*temp1+y[i-1]*temp2;
                        k = k + 1;
                    }
                    kk = kk + n - j + 1;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[jx-1] != zero) || (y[jy-1] != zero)) {
                    temp1 = alpha*y[jy-1];
                    temp2 = alpha*x[jx-1];
                    ix = jx;
                    iy = jy;
                    for (k = kk; k <= (kk + n-j); k = k + 1) {
                        ap[k-1] = ap[k-1] + x[ix-1]*temp1+y[iy-1]*temp2;
                        ix = ix + incx;
                        iy = iy + incy;
                    }
                    jx = jx + incx;
                    jy = jy + incy;
                    kk = kk + n - j + 1;
                }
            }
        }
    }
}

public static void DSYMV(string uplo, int n, double alpha, double[,] a, int lda, double[] x, int incx, double beta, ref double[] y, int incy)
{
    // DOUBLE PRECISION A(LDA,*),X(*),Y(*)
    // DSYMV performs the matrix-vector operation

    // y := alpha*A*x + beta*y,

    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

    // On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.
    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry, ALPHA specifies the scalar alpha.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the
    symmetric matrix and the strictly
    // lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A
    must contain the lower
    // triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

    // Y is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // Parameters
    const double one = 1.0;
    const double zero = 0.0;

    // Local Scalars
    double temp1, temp2;
    int info, ix, iy, jx, jy, kx, ky;

```

```

// Test the input parameters
info = 0;
if (!(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L")) {
    info = 1;
} else if (n < 0) {
    info = 2;
} else if (lda < Math.Max(1,n)) {
    info = 5;
} else if (incx == 0) {
    info = 7;
} else if (incy == 0) {
    info = 10;
}
if (info != 0) {
    XERBLA("DSYMV ",info);
    return;
}

// Quick return if possible
if ((n == 0) || ((alpha == zero) && (beta == one))) {
    return;
}

// Set up the start points in X and Y
if (incx > 0) {
    kx = 1;
} else {
    kx = 1-(n-1)*incx;
}
if (incy > 0) {
    ky = 1;
} else {
    ky = 1-(n-1)*incy;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A
//
// First form y := beta*y
if (beta != one) {
    if (incy == 1) {
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = zero;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[i-1] = beta*y[i-1];
            }
        }
    } else {
        iy = ky;
        if (beta == zero) {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = zero;
                iy = iy + incy;
            }
        } else {
            for (int i = 1; i <= n; i = i + 1) {
                y[iy-1] = beta*y[iy-1];
                iy = iy + incy;
            }
        }
    }
}
if (alpha == zero) {
    return;
}
if (uplo.ToUpper().Substring(0,1) == "U") {
    // Form y when A is stored in upper triangle
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            for (int i = 1; i <= j-1; i = i + 1) {
                y[i-1] = y[i-1] + temp1*a[i-1,j-1];
                temp2 = temp2+a[i-1,j-1]*x[i-1];
            }
            y[j-1] = y[j-1] + temp1*a[j-1,j-1]+alpha*temp2;
        }
    } else {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            ix = kx;
            iy = ky;
            for (int i = 1; i <= j-1; i = i + 1) {
                y[iy-1] = y[iy-1] + temp1*a[i-1,j-1];
                temp2 = temp2 + a[i-1,j-1]*x[ix-1];
                ix = ix + incx;
                iy = iy + incy;
            }
            y[jy-1] = y[jy-1] + temp1*a[j-1,j-1] + alpha*temp2;
            jx = jx + incx;
            jy = jy + incy;
        }
    }
} else {
    // Form y when A is stored in lower triangle.
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[j-1];
            temp2 = zero;
            y[j-1] = y[j-1] + temp1*a[j-1,j-1];
            for (int i = j+1; i <= n; i = i + 1) {
                y[i-1] = y[i-1] + temp1*a[i-1,j-1];
                temp2 = temp2 + a[i-1,j-1]*x[i-1];
            }
            y[j-1] = y[j-1] + alpha*temp2;
        }
    } else {
        jx = kx;
        jy = ky;

```

```

        for (int j = 1; j <= n; j = j + 1) {
            temp1 = alpha*x[jx-1];
            temp2 = zero;
            y[jy-1] = y[jy-1] + temp1*a[j-1,j-1];
            ix = jx;
            iy = jy;
            for (int i = j+1; i <= n; i = i + 1) {
                ix = ix + incx;
                iy = iy + incy;
                y[iy-1] = y[iy-1] + temp1*a[i-1,j-1];
                temp2 = temp2 + a[i-1,j-1]*x[ix-1];
            }
            y[jy-1] = y[jy-1] + alpha*temp2;
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}

public static void DTRSV(string uplo, string trans, string diag, int n, double[,] a, int lda, ref double[] x, int incx)
{
    // DOUBLE PRECISION A(LDA,*),X(*)

    // DTRSV solves one of the systems of equations

    // A*x = b, or A**T*x = b,

    // where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.
    // No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

    // On entry, UPLO specifies the matrix is an upper or lower triangular matrix as follows:
    //
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // On entry, TRANS specifies the equations to be solved as follows:
    //
    // TRANS = 'N' or 'n' A*x = b.
    // TRANS = 'T' or 't' A**T*x = b.
    // TRANS = 'C' or 'c' A**T*x = b.

    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    //
    // DIAG = 'U' or 'u' A is assumed to be unit triangular.
    // DIAG = 'N' or 'n' A is not assumed to be unit triangular.

    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and
    the strictly lower triangular part of
    // A is not referenced.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and
    the strictly upper triangular part of
    // A is not referenced.
    // Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 )*abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution
    vector x.

    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Level 2 Blas routine.

    //Parameters
    const double zero = 0.0;

    // Local Scalars
    double temp;
    int info, ix, jx, kx;
    bool nount;

    kx = 0; //added
    // Test the input parameters
    info = 0;
    if ( !(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
        info = 1;
    } else if ( !(trans.ToUpper().Substring(0,1) == "N") && !(trans.ToUpper().Substring(0,1) == "T") && !(trans.ToUpper().Substring(0,1) == "C") ) {
        info = 2;
    } else if ( !(diag.ToUpper().Substring(0,1) == "U") && !(diag.ToUpper().Substring(0,1) == "N") ) {
        info = 3;
    } else if ( n < 0 ) {
        info = 4;
    } else if ( lda < Math.Max(1,n) ) {
        info = 6;
    } else if ( incx == 0 ) {
        info = 8;
    }
    if (info != 0) {
        XERBLA("DTRSV ",info);
        return;
    }

    // Quick return if possible.
    if (n == 0) {
        return;
    }

    nount = (diag.ToUpper().Substring(0,1) == "N");

    // Set up the start point in X if the increment is not unity. This will be ( N - 1 )*INCX too small for descending loops.
    if (incx <= 0) {
        kx = 1-(n-1)*incx;
    } else if (incx != 1) {
        kx = 1;
    }

    // Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
    if (trans.ToUpper().Substring(0,1) == "N") {
        // Form x := inv( A ) * x.

```

```

if (uplo.ToUpper().Substring(0,1) == "U") {
  if (incx == 1) {
    for (int j = n; j >= 1; j = j - 1) { //DO 20 J = N,1,-1
      if (x[j-1] != zero) {
        if (nounit) {
          x[j-1] = x[j-1]/a[j-1,j-1];
        }
        temp = x[j-1];
        for (int i = j-1; i >= 1; i = i - 1) { //DO 10 I = J - 1,1,-1
          x[i-1] = x[i-1] - temp*a[i-1,j-1];
        }
      }
    }
  } else {
    jx = kx + (n-1)*incx;
    for (int j = n; j >= 1; j = j - 1) { //DO 40 J = N,1,-1
      if (x[jx-1] != zero) {
        if (nounit) {
          x[jx-1] = x[jx-1]/a[j-1,j-1];
        }
        temp = x[jx-1];
        ix = jx;
        for (int i = j-1; i >= 1; i = i - 1) { //DO 30 I = J - 1,1,-1
          ix = ix - incx;
          x[ix-1] = x[ix-1] - temp*a[i-1,j-1];
        }
      }
      jx = jx - incx;
    }
  }
} else {
  if (incx == 1) {
    for (int j = 1; j <= n; j = j + 1) {
      if (x[j-1] != zero) {
        if (nounit) {
          x[j-1] = x[j-1]/a[j-1,j-1];
        }
        temp = x[j-1];
        for (int i = j+1; i <= n; i = i + 1) {
          x[i-1] = x[i-1] - temp*a[i-1,j-1];
        }
      }
    }
  } else {
    jx = kx;
    for (int j = 1; j <= n; j = j + 1) {
      if (x[jx-1] != zero) {
        if (nounit) {
          x[jx-1] = x[jx-1]/a[j-1,j-1];
        }
        temp = x[jx-1];
        ix = jx;
        for (int i = j+1; i <= n; i = i + 1) {
          ix = ix + incx;
          x[ix-1] = x[ix-1] - temp*a[i-1,j-1];
        }
      }
      jx = jx + incx;
    }
  }
}
} else {
  Form x := inv( A**T ) * x.
  if (uplo.ToUpper().Substring(0,1) == "U") {
    if (incx == 1) {
      for (int j = 1; j <= n; j = j + 1) {
        temp = x[j-1];
        for (int i = 1; i <= j-1; i = i + 1) {
          temp = temp - a[i-1,j-1]*x[i-1];
        }
        if (nounit) {
          temp = temp/a[j-1,j-1];
        }
        x[j-1] = temp;
      }
    } else {
      jx = kx;
      for (int j = 1; j <= n; j = j + 1) {
        temp = x[jx-1];
        ix = kx;
        for (int i = 1; i <= j-1; i = i + 1) {
          temp = temp - a[i-1,j-1]*x[ix-1];
          ix = ix + incx;
        }
        if (nounit) {
          temp = temp/a[j-1,j-1];
        }
        x[jx-1] = temp;
        jx = jx + incx;
      }
    }
  }
} else {
  if (incx == 1) {
    for (int j = n; j >= 1; j = j - 1) { //DO 140 J = N,1,-1
      temp = x[j-1];
      for (int i = n; i >= j+1; i = i - 1) { //DO 130 I = N,J + 1,-1
        temp = temp - a[i-1,j-1]*x[i-1];
      }
      if (nounit) {
        temp = temp/a[j-1,j-1];
      }
      x[j-1] = temp;
    }
  } else {
    kx = kx + (n-1)*incx;
    jx = kx;
    for (int j = n; j >= 1; j = j-1) { //DO 160 J = N,1,-1
      temp = x[jx-1];
      ix = kx;
      for (int i = n; i >= j+1; i = i - 1) { //DO 150 I = N,J + 1,-1
        temp = temp - a[i-1,j-1]*x[ix-1];
        ix = ix - incx;
      }
    }
  }
}

```



```

// DSYR performs the symmetric rank 1 operation

// A := alpha*x*x**T + A,
// where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix.

// On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
// UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.
// UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

// On entry, N specifies the order of the matrix A. N must be at least zero.

// On entry, ALPHA specifies the scalar alpha.

// X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the n element vector x.

// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

// A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
// Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the
symmetric matrix and the strictly
// lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part
of the updated matrix.
// Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the
symmetric matrix and the strictly
// upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part
of the updated matrix.

// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

// Parameters
const double zero = 0.0;

// Local Scalars
double temp;
int info, ix, jx, kx; //i,j local

kx = 0; // added
// Test the input parameters
info = 0;
if (!(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
    info = 1;
} else if (n < 0) {
    info = 2;
} else if (incx == 0) {
    info = 5;
} else if (lda < Math.Max(1,n)) {
    info = 7;
}
if (info != 0) {
    XERBLA("DSYR ",info);
    return;
}

// Quick return if possible
if ((n == 0) || (alpha == zero)) {
    return;
}

// Set the start point in X if the increment is not unity
if (incx <= 0) {
    kx = 1 - (n-1) * incx;
} else if (incx != 1) {
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.
if (uplo.ToUpper().Substring(0,1) == "U") {
    // Form A when A is stored in upper triangle
    if (incx == 1) {
        for (int j = 1; j <= n; j = j + 1) {
            if (x[j-1] != zero) {
                temp = alpha * x[j-1];
                for (int i = 1; i <= j; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[i-1] * temp;
                }
            }
        }
    } else {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1) {
            if (x[jx-1] != zero) {
                temp = alpha * x[jx-1];
                ix = kx;
                for (int i = 1; i <= j; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[ix-1] * temp;
                    ix = ix + incx;
                }
                jx = jx + incx;
            }
        }
    }
} else {
    // Form A when A is stored in lower triangle.
    if (incx == 1) {
        for (int j = 1; j <= n; j = j + 1) {
            if (x[j-1] != zero) {
                temp = alpha * x[j-1];
                for (int i = j; i <= n; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[i-1] * temp;
                }
            }
        }
    } else {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1) {
            if (x[jx-1] != zero) {
                temp = alpha * x[jx-1];
                ix = jx;
                for (int i = j; i <= n; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[ix-1] * temp;
                    ix = ix + incx;
                }
            }
        }
    }
}

```

```

        }
        jx = jx + incx;
    }
}
}
public static void DSYR2(string uplo, int n, double alpha, double[] x, int incx, double[] y, int incy, ref double[,] a, int lda)
{
    // DSYR2 performs the symmetric rank 2 operation

    // A := alpha*x*y**T + alpha*y*x**T + A,

    // where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix.

    // On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // On entry, ALPHA specifies the scalar alpha.

    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Y is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.

    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the
    symmetric matrix and the strictly
    // lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part
    of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the
    symmetric matrix and the strictly
    // upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part
    of the updated matrix.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // DOUBLE PRECISION A(LDA,*),X(*),Y(*)

    // Parameters
    const double zero = 0.0;

    // Local Scalars
    double temp1, temp2;
    int info, ix, iy, jx, jy, kx, ky;

    jx = jy = kx = ky = 0; // added
    // Test the input parameters
    info = 0;
    if (!(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L")) {
        info = 1;
    } else if (n < 0) {
        info = 2;
    } else if (incx == 0) {
        info = 5;
    } else if (incy == 0) {
        info = 7;
    } else if (lda < Math.Max(1,n)) {
        info = 9;
    }
    if (info != 0) {
        XERBLA("DSYR2 ",info);
        return;
    }

    // Quick return if possible
    if ((n == 0) || (alpha == zero)) {
        return;
    }

    // Set up the start points in X and Y if the increments are not both unity.
    if ((incx != 1) || (incy != 1)) {
        if (incx > 0) {
            kx = 1;
        } else {
            kx = 1-(n-1)*incx;
        }
        if (incy > 0) {
            ky = 1;
        } else {
            ky = 1-(n-1)*incy;
        }
        jx = kx;
        jy = ky;
    }

    // Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.
    if (uplo.ToUpper().Substring(0,1) == "U") {
        // Form A when A is stored in the upper triangle
        if ((incx == 1) && (incy == 1)) {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[j-1] != zero) || (y[j-1] != zero)) {
                    temp1 = alpha*y[j-1];
                    temp2 = alpha*x[j-1];
                    for (int i = 1; i <= j; i = i + 1) {
                        a[i-1,j-1] = a[i-1,j-1] + x[i-1]*temp1+y[i-1]*temp2;
                    }
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                if ((x[jx-1] != zero) || (y[jy-1] != zero)) {
                    temp1 = alpha*y[jy-1];
                    temp2 = alpha*x[jx-1];
                    ix = kx;
                    iy = ky;
                    for (int i = 1; i <= j; i = i + 1) {

```

```

        a[i-1,j-1] = a[i-1,j-1]+x[ix-1]*templ+y[iy-1]*temp2;
        ix = ix + incx;
        iy = iy + incy;
    }
    }
    jx = jx + incx;
    jy = jy + incy;
}
} else {
    // Form A when A is stored in the lower triangle
    if ((incx == 1) && (incy == 1)) {
        for (int j = 1; j <= n; j = j + 1) {
            if ((x[j-1] != zero) || (y[j-1] != zero)) {
                templ = alpha*y[j-1];
                temp2 = alpha*x[j-1];
                for (int i = j; i <= n; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[i-1]*templ+y[i-1]*temp2;
                }
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            if ((x[jx-1] != zero) || (y[jy-1] != zero)) {
                templ = alpha*y[jy-1];
                temp2 = alpha*x[jx-1];
                ix = jx;
                iy = jy;
                for (int i = j; i <= n; i = i + 1) {
                    a[i-1,j-1] = a[i-1,j-1] + x[ix-1]*templ+y[iy-1]*temp2;
                    ix = ix + incx;
                    iy = iy + incy;
                }
            }
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}
}

public static void DTBMV(string uplo, string trans, string diag, int n, int k, double[,] a, int lda, ref double[] x, int incx)
{
    // DTBMV performs one of the matrix-vector operations
    // x := A*x, or x := A**T*x,
    // where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with ( k + 1 ) diagonals.
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' x := A*x.
    // TRANS = 'T' or 't' x := A**T*x.
    // TRANS = 'C' or 'c' x := A**T*x.
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    // DIAG = 'U' or 'u' A is assumed to be unit triangular.
    // DIAG = 'N' or 'n' A is not assumed to be unit triangular.
    // On entry, N specifies the order of the matrix A. N must be at least zero.
    // On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.
    // On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.
    // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
    matrix of coefficients, supplied column by
    // column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
    so on. The top left k by k triangle
    // of the array A is not referenced.
    // The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:
    //
    // DO 20, J = 1, N
    // M = K + 1 - J
    // DO 10, I = MAX( 1, J - K ), J
    // A( M + I, J ) = matrix( I, J )
    // 10 CONTINUE
    // 20 CONTINUE
    //
    // Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
    matrix of coefficients, supplied column by
    // column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
    The bottom right k by k triangle of the
    // array A is not referenced.
    // The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:
    //
    // DO 20, J = 1, N
    // M = 1 - J
    // DO 10, I = J, MIN( N, J + K )
    // A( M + I, J ) = matrix( I, J )
    // 10 CONTINUE
    // 20 CONTINUE
    //
    // Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but
    are assumed to be unity.
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).
    // X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the tranformed vector x.
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
    // Level 2 Blas routine.
    // The vector and matrix arguments are not referenced when N = 0, or M = 0
    // DOUBLE PRECISION A(LDA,*),X(*)
    // Parameters
    const double zero = 0.0;

```

```

// Local Scalars
double temp;
int info, ix, jx, kplus1, kx, l;
bool nunit;

kx = 0; // added
// Test the input parameters
info = 0;
if (! (uplo.ToUpper().Substring(0,1) == "U") && ! (uplo.ToUpper().Substring(0,1) == "L")) {
    info = 1;
} else if (! (trans.ToUpper().Substring(0,1) == "N") && ! (trans.ToUpper().Substring(0,1) == "T") && ! (trans.ToUpper().Substring(0,1) == "C")) {
    info = 2;
} else if (! (diag.ToUpper().Substring(0,1) == "U") && ! (diag.ToUpper().Substring(0,1) == "N")) {
    info = 3;
} else if (n < 0) {
    info = 4;
} else if (k < 0) {
    info = 5;
} else if (lda < (k+1)) {
    info = 7;
} else if (incx == 0) {
    info = 9;
}
if (info != 0) {
    XERBLA("DTBMV ",info);
    return;
}

// Quick return if possible
if (n == 0) {
    return;
}

nunit = (diag.ToUpper().Substring(0,1) == "N");

// Set up the start point in X if the increment is not unity. This will be (N-1)*INCX too small for descending loops.
if (incx <= 0) {
    kx = 1-(n-1)*incx;
} else if (incx != 1) {
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if (trans.ToUpper().Substring(0,1) == "N") {
    // Form x := A*x.
    if (uplo.ToUpper().Substring(0,1) == "U") {
        kplus1 = k + 1;
        if (incx == 1) {
            for (int j = 1; j <= n; j = j + 1) {
                if (x[j-1] != zero) {
                    temp = x[j-1];
                    l = kplus1 - j;
                    for (int i = Math.Max(1,j-k); i <= j-1; i = i + 1) {
                        x[i-1] = x[i-1] + temp*a[l+i-1,j-1];
                    }
                    if (nunit) {
                        x[j-1] = x[j-1]*a[kplus1-1,j-1];
                    }
                }
            }
        } else {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1) {
                if (x[jx-1] != zero) {
                    temp = x[jx-1];
                    ix = kx;
                    l = kplus1 - j;
                    for (int i = Math.Max(1,j-k); i <= j-1; i = i + 1) {
                        x[ix-1] = x[ix-1] + temp*a[l+i-1,j-1];
                        ix = ix + incx;
                    }
                    if (nunit) {
                        x[jx-1] = x[jx-1] * a[kplus1-1,j-1];
                    }
                }
                jx = jx + incx;
                if (j > k) {
                    kx = kx + incx;
                }
            }
        }
    } else {
        if (incx == 1) {
            for (int j = n; j >= 1; j = j - 1) { //DO 60 J = N,1,-1
                if (x[j-1] != zero) {
                    temp = x[j-1];
                    l = 1-j;
                    for (int i = Math.Min(n,j+k); i >= j+1; i = i - 1) { //DO 50 I = MIN(N,J+K),J + 1,-1
                        x[i-1] = x[i-1] + temp*a[l+i-1,j-1];
                    }
                    if (nunit) {
                        x[j-1] = x[j-1]*a[0,j-1];
                    }
                }
            }
        } else {
            kx = kx + (n-1)*incx;
            jx = kx;
            for (int j = n; j >= 1; j = j - 1) { //DO 80 J = N,1,-1
                if (x[jx-1] != zero) {
                    temp = x[jx-1];
                    ix = kx;
                    l = 1-j;
                    for (int i = Math.Min(n,j+k); i >= j+1; i = i - 1) { //DO 70 I = MIN(N,J+K),J + 1,-1
                        x[ix-1] = x[ix-1] + temp*a[l+i-1,j-1];
                        ix = ix - incx;
                    }
                    if (nunit) {
                        x[jx-1] = x[jx-1]*a[0,j-1];
                    }
                }
                jx = jx - incx;
                if ((n-j) >= k) {
                    kx = kx - incx;
                }
            }
        }
    }
}

```

```

    }
  }
} else {
  // Form x := A**T*x.
  if (uplo.ToUpper().Substring(0,1) == "U") {
    kplus1 = k + 1;
    if (incx == 1) {
      for (int j = n; j >= 1; j = j - 1) { //DO 100 J = N,1,-1
        temp = x[j-1];
        l = kplus1 - j;
        if (nunit) {
          temp = temp*a[kplus1-1,j-1];
        }
        for (int i = j-1; i >= Math.Max(1,j-k); i = i - 1) { //DO 90 I = J - 1,MAX(1,J-K),-1
          temp = temp + a[l+i-1,j-1]*x[i-1];
        }
        x[j-1] = temp;
      }
    } else {
      kx = kx + (n-1)*incx;
      jx = kx;
      for (int j = n; j >= 1; j = j - 1) { //DO 120 J = N,1,-1
        temp = x[jx-1];
        kx = kx - incx;
        ix = kx;
        l = kplus1 - j;
        if (nunit) {
          temp = temp*a[kplus1-1,j-1];
        }
        for (int i = j-1; i >= Math.Max(1,j-k); i = i - 1) { //DO 110 I = J - 1,MAX(1,J-K),-1
          temp = temp + a[l+i-1,j-1]*x[ix-1];
          ix = ix - incx;
        }
        x[jx-1] = temp;
        jx = jx - incx;
      }
    }
  } else {
    if (incx == 1) {
      for (int j = 1; j <= n; j = j + 1) {
        temp = x[j-1];
        l = 1 - j;
        if (nunit) {
          temp = temp*a[0,j-1];
        }
        for (int i = j+1; i <= Math.Min(n,j+k); i = i + 1) {
          temp = temp + a[l+i-1,j-1]*x[i-1];
        }
        x[j-1] = temp;
      }
    } else {
      jx = kx;
      for (int j = 1; j <= n; j = j + 1) {
        temp = x[jx-1];
        kx = kx + incx;
        ix = kx;
        l = 1 - j;
        if (nunit) {
          temp = temp*a[0,j-1];
        }
        for (int i = j+1; i <= Math.Min(n,j+k); i = i + 1) {
          temp = temp + a[l+i-1,j-1]*x[ix-1];
          ix = ix + incx;
        }
        x[jx-1] = temp;
        jx = jx + incx;
      }
    }
  }
}

}

public static void DTBSV(string uplo, string trans, string diag, int n, int k, double[, ] a, int lda, ref double[] x, int incx)
{
  // DTBSV solves one of the systems of equations
  //
  // A*x = b, or A**T*x = b,
  //
  // where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with ( k + 1 ) diagonals.
  //
  // No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.
  //
  // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
  //
  // UPLO = 'U' or 'u' A is an upper triangular matrix.
  //
  // UPLO = 'L' or 'l' A is a lower triangular matrix.
  //
  // On entry, TRANS specifies the equations to be solved as follows:
  //
  // TRANS = 'N' or 'n' A*x = b.
  //
  // TRANS = 'T' or 't' A**T*x = b.
  //
  // TRANS = 'C' or 'c' A**T*x = b.
  //
  // On entry, DIAG specifies whether or not A is unit triangular as follows:
  //
  // DIAG = 'U' or 'u' A is assumed to be unit triangular.
  //
  // DIAG = 'N' or 'n' A is not assumed to be unit triangular.
  //
  // On entry, N specifies the order of the matrix A. N must be at least zero.
  //
  // On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.
  //
  // On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.
  //
  // A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
  //
  // Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
  //
  // matrix of coefficients, supplied column by
  //
  // column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
  //
  // so on. The top left k by k triangle
  //
  // of the array A is not referenced.
  //
  // The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:
  //
  //
  //
  //      DO 20, J = 1, N
  //        M = K + 1 - J
  //        DO 10, I = MAX( 1, J - K ), J
  //          A( M + I, J ) = matrix( I, J )
  //

```

```

//          10    CONTINUE
//          20 CONTINUE
//
// Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
matrix of coefficients, supplied column by
// column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
The bottom right k by k triangle of the
// array A is not referenced.
// The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:
//          DO 20, J = 1, N
//              M = 1 - J
//              DO 10, I = J, MIN( N, J + K )
//                  A( M + I, J ) = matrix( I, J )
//          10    CONTINUE
//          20 CONTINUE
//
// Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but
are assumed to be unity.

// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).

// X is DOUBLE PRECISION array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution
vector x.

// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

// Parameters
const double zero = 0.0;

// Local Scalars
double temp;
int info, ix, jx, kplus1, kx, l;
bool nunit;

kx = 0; //added
// Test the input parameters
info = 0;
if ( !(uplo.ToUpper().Substring(0,1) == "U") && !(uplo.ToUpper().Substring(0,1) == "L") ) {
    info = 1;
} else if ( !(trans.ToUpper().Substring(0,1) == "N") && !(trans.ToUpper().Substring(0,1) == "T") && !(trans.ToUpper().Substring(0,1) == "C") ) {
    info = 2;
} else if ( !(diag.ToUpper().Substring(0,1) == "U") && !(diag.ToUpper().Substring(0,1) == "N") ) {
    info = 3;
} else if ( n < 0 ) {
    info = 4;
} else if ( k < 0 ) {
    info = 5;
} else if ( lda < (k+1) ) {
    info = 7;
} else if ( incx == 0 ) {
    info = 9;
}
if (info != 0) {
    XERBLA("DTBSV ", info);
    return;
}

// Quick return if possible
if (n == 0) {
    return;
}

nunit = (diag.ToUpper().Substring(0,1) == "N");

// Set up the start point in X if the increment is not unity. This will be ( N - 1 ) * INCX too small for descending loops.
if (incx <= 0) {
    kx = 1 - (n-1) * incx;
} else if (incx != 1) {
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed by sequentially with one pass through A.
if (trans.ToUpper().Substring(0,1) == "N") {
    // Form x := inv( A ) * x.
    if (uplo.ToUpper().Substring(0, 1) == "U")
    {
        kplus1 = k + 1;
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 20 J = N,1,-1
                if (x[j - 1] != zero)
                {
                    l = kplus1 - j;
                    if (nunit)
                    {
                        x[j - 1] = x[j - 1] / a[kplus1 - 1, j - 1];
                    }
                    temp = x[j - 1];
                    for (int i = (j - 1); i >= Math.Max(1, j - k); i = i - 1)
                    {
                        x[i - 1] = x[i - 1] - temp * a[(1 + i) - 1, j - 1];
                    }
                }
            }
        }
        else
        {
            kx = kx + (n - 1) * incx;
            jx = kx;
            for (int j = n; j >= 1; j = j - 1) { //DO 40 J = N,1,-1
                kx = kx - incx;
                if (x[jx - 1] != zero)
                {
                    ix = kx;
                    l = kplus1 - j;
                    if (nunit)
                    {
                        x[jx - 1] = x[jx - 1] / a[kplus1 - 1, j - 1];
                    }
                    temp = x[jx - 1];
                    for (int i = (j - 1); i >= Math.Max(1, j - k); i = i - 1)    { //DO 30 I = J - 1, MAX(1, J-K), -1

```

```

        x[ix - 1] = x[ix - 1] - temp * a[(l + i) - 1, j - 1];
        ix = ix - incx;
    }
    }
    jx = jx - incx;
}
}
} else
{
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[j - 1] != zero)
            {
                l = l - j;
                if (nounit)
                {
                    x[j - 1] = x[j - 1] / a[0, j - 1];
                }
                temp = x[j - 1];
                for (int i = (j + 1); i <= Math.Min(n, j + k); i = i + 1)
                {
                    x[i - 1] = x[i - 1] - temp * a[(l + i) - 1, j - 1];
                }
            }
        }
    }
    else
    {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1)
        {
            kx = kx + incx;
            if (x[jx - 1] != zero)
            {
                ix = kx;
                l = l - j;
                if (nounit)
                {
                    x[jx - 1] = x[jx - 1] / a[0, j - 1];
                }
                temp = x[jx - 1];
                for (int i = (j + 1); i <= Math.Min(n, j + k); i = i + 1)
                {
                    x[ix - 1] = x[ix - 1] - temp * a[(l + i) - 1, j - 1];
                    ix = ix + incx;
                }
            }
            jx = jx + incx;
        }
    }
}
}
} else {
    // Form x := inv( A**T)*x.
    if (uplo.ToUpper().Substring(0,1) == "U") {
        kplus1 = k + 1;
        if (incx == 1) {
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[j-1];
                l = kplus1 - j;
                for (int i = Math.Max(1,j-k); i <= (j-1); i = i + 1) {
                    temp = temp - a[(l+i)-1,j-1]*x[i-1];
                }
                if (nounit) {
                    temp = temp/a[kplus1-1,j-1];
                }
                x[j-1] = temp;
            }
        }
        else {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[jx-1];
                ix = kx;
                l = kplus1 - j;
                for (int i = Math.Max(1,j-k); i <= (j-1); i = i + 1) {
                    temp = temp - a[(l+i)-1,j-1]*x[ix-1];
                    ix = ix + incx;
                }
                if (nounit) {
                    temp = temp/a[kplus1-1,j-1];
                }
                x[jx-1] = temp;
                jx = jx + incx;
                if (j > k) {
                    kx = kx + incx;
                }
            }
        }
    }
    else {
        if (incx == 1) {
            for (int j = n; j >= 1; j = j - 1) { //DO 140 J = N,1,-1
                temp = x[j-1];
                l = 1-j;
                for (int i = Math.Min(n,j+k); i >= (j+1); i = i - 1) { //DO 130 I = MIN(N,J+K),J + 1,-1
                    temp = temp - a[(l+i)-1,j-1]*x[i-1];
                }
                if (nounit) {
                    temp = temp/a[0,j-1];
                }
                x[j-1] = temp;
            }
        }
        else {
            kx = kx + (n-1)*incx;
            jx = kx;
            for (int j = n; j >= 1; j = j - 1) { //DO 160 J = N,1,-1
                temp = x[jx-1];
                ix = kx;
                l = 1-j;
                for (int i = Math.Min(n,j+k); i >= (j+1); i = i - 1) { //DO 150 I = MIN(N,J+K),J + 1,-1
                    temp = temp - a[(l+i)-1,j-1]*x[ix-1];
                    ix = ix - incx;
                }
            }
        }
    }
}

```



```

    }
    kk = kk + j;
}
} else {
    jx = kx;
    for (int j = 1; j <= n; j = j + 1) {
        if (x[jx-1] != zero) {
            temp = x[jx-1];
            ix = kx;
            for (k = kk; k <= (kk + j - 2); k = k + 1) {
                x[ix-1] = x[ix-1] + temp*ap[k-1];
                ix = ix + incx;
            }
            if (nounit) {
                x[jx-1] = x[jx-1]*ap[(kk+j-1)-1];
            }
        }
        jx = jx + incx;
        kk = kk + j;
    }
}
} else {
    kk = (n*(n+1))/2;
    if (incx == 1) {
        for (int j = n; j >= 1; j = j - 1) { //DO 60 J = N,1,-1
            if (x[j-1] != zero) {
                temp = x[j-1];
                k = kk;
                for (int i = n; i >= (j+1); i = i - 1) { //DO 50 I = N,J + 1,-1
                    x[i-1] = x[i-1] + temp*ap[k-1];
                    k = k - 1;
                }
                if (nounit) {
                    x[j-1] = x[j-1]*ap[(kk-n+j)-1];
                }
            }
            kk = kk - (n-j+1);
        }
    } else {
        kx = kx + (n-1)*incx;
        jx = kx;
        for (int j = n; j >= 1; j = j - 1) {
            if (x[jx-1] != zero) {
                temp = x[jx-1];
                ix = kx;
                for (k = kk; k >= (kk - (n - (j+1)) ); k = k - 1) { //DO 70 K = KK, KK - (N - (J+1)), -1
                    x[ix-1] = x[ix-1] + temp*ap[k-1];
                    ix = ix - incx;
                }
                if (nounit) {
                    x[jx-1] = x[jx-1] * ap[(kk-n+j)-1];
                }
            }
            jx = jx - incx;
            kk = kk - (n-j+1);
        }
    }
}
} else {
    // Form x := A**T*x.
    if (uplo.ToUpper().Substring(0,1) == "U") {
        kk = (n*(n+1))/2;
        if (incx == 1) {
            for (int j = n; j >= 1; j = j - 1) { //DO 100 J = N,1,-1
                temp = x[j-1];
                if (nounit) {
                    temp = temp*ap[kk-1];
                }
                k = kk - 1;
                for (int i = (j-1); i >= 1; i = i - 1) { //DO 90 I = J - 1,1,-1
                    temp = temp + ap[k-1]*x[i-1];
                    k = k - 1;
                }
                x[j-1] = temp;
                kk = kk - j;
            }
        } else {
            jx = kx + (n-1)*incx;
            for (int j = n; j >= 1; j = j - 1) { //DO 120 J = N,1,-1
                temp = x[jx-1];
                ix = jx;
                if (nounit) {
                    temp = temp*ap[kk-1];
                }
                for (k = (kk-1); k >= (kk-j+1); k = k - 1) { //DO 110 K = KK - 1, KK - J + 1, -1
                    ix = ix - incx;
                    temp = temp + ap[k-1]*x[ix-1];
                }
                x[jx-1] = temp;
                jx = jx - incx;
                kk = kk - j;
            }
        }
    } else {
        kk = 1;
        if (incx == 1) {
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[j-1];
                if (nounit) {
                    temp = temp * ap[kk-1];
                }
                k = kk + 1;
                for (int i = (j+1); i <= n; i = i + 1) {
                    temp = temp + ap[k-1]*x[i-1];
                    k = k + 1;
                }
                x[j-1] = temp;
                kk = kk + (n-j+1);
            }
        } else {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[jx-1];
                ix = jx;

```



```

    }
    }
    kk = kk - j;
}
} else {
    jx = kx + (n-1)*incx;
    for (int j = n; j >= 1; j = j - 1) { //DO 40 J = N,1,-1
        if (x[jx-1] != zero) {
            if (nounit) {
                x[jx-1] = x[jx-1]/ap[kk-1];
            }
            temp = x[jx-1];
            ix = jx;
            for (k = kk-1; k >= kk - j + 1; k = k - 1) { //DO 30 K = KK - 1, KK - J + 1, -1
                ix = ix - incx;
                x[ix-1] = x[ix-1] - temp*ap[k-1];
            }
            jx = jx - incx;
            kk = kk - j;
        }
    }
} else {
    kk = 1;
    if (incx == 1) {
        for (int j = 1; j <= n; j = j + 1) {
            if (x[j-1] != zero) {
                if (nounit) {
                    x[j-1] = x[j-1]/ap[kk-1];
                }
                temp = x[j-1];
                k = kk + 1;
                for (int i = j+1; i <= n; i = i + 1) {
                    x[i-1] = x[i-1] - temp*ap[k-1];
                    k = k + 1;
                }
            }
            kk = kk + (n-j+1);
        }
    } else {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1) {
            if (x[jx-1] != zero) {
                if (nounit) {
                    x[jx-1] = x[jx-1]/ap[kk-1];
                }
                temp = x[jx-1];
                ix = jx;
                for (k = kk+1; k <= kk+n-j; k = k + 1) {
                    ix = ix + incx;
                    x[ix-1] = x[ix-1] - temp*ap[k-1];
                }
            }
            jx = jx + incx;
            kk = kk + (n-j+1);
        }
    }
}
} else {
    // Form x := inv( A**T ) * x.
    if (uplo.ToUpper().Substring(0,1) == "U") {
        kk = 1;
        if (incx == 1) {
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[j-1];
                k = kk;
                for (int i = 1; i <= j - 1; i = i + 1) {
                    temp = temp - ap[k-1]*x[i-1];
                    k = k + 1;
                }
                if (nounit) {
                    temp = temp/ap[kk+j-1-1];
                }
                x[j-1] = temp;
                kk = kk + j;
            }
        } else {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1) {
                temp = x[jx-1];
                ix = kx;
                for (k = kk; k <= kk+j-2; k = k + 1) {
                    temp = temp - ap[k-1]*x[ix-1];
                    ix = ix + incx;
                }
                if (nounit) {
                    temp = temp/ap[kk+j-1-1];
                }
                x[jx-1] = temp;
                jx = jx + incx;
                kk = kk + j;
            }
        }
    }
} else {
    kk = (n*(n+1))/2;
    if (incx == 1) {
        for (int j = n; j >= 1; j = j - 1) { //DO 140 J = N,1,-1
            temp = x[j-1];
            k = kk;
            for (int i = n; i >= j+1; i = i - 1) {
                temp = temp - ap[k-1]*x[i-1];
                k = k - 1;
            }
            if (nounit) {
                temp = temp/ap[kk-n+j-1];
            }
            x[j-1] = temp;
            kk = kk - (n-j+1);
        }
    } else {
        kx = kx + (n-1)*incx;
        jx = kx;
        for (int j = n; j >= 1; j = j - 1) { //DO 160 J = N,1,-1
            temp = x[jx-1];

```



```

tmp.Clear();
//using (StreamReader infile = new StreamReader(fin))
using (StreamReader infile = new StreamReader(Console.OpenStandardInput()))
{
    string line;
    int i = 0;
    while ((line = infile.ReadLine()) != null)
    {
        tmp.Add(line.Trim());
        Console.WriteLine("Read:{0}", line.Trim());
    }
}
findata = (String[])tmp.ToArray(typeof(string));
}
catch (Exception e)
{
    Console.WriteLine("Error Reading {0}", fin);
    Environment.Exit(1);
}

//Read name and unit number for summary output file and open file.
summary = findata[0].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
nout = int.Parse(findata[1].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

fout1 = new StreamWriter(summary, false); // true = append
misc d.nout = fout1; // for XERBLA
fout2 = null;
noutc = nout;

// Read name and unit number for snapshot output file and open file.
snaps = findata[2].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
ntra = int.Parse(findata[3].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

trace = (ntra >= 0);
//Console.WriteLine("ntra: {0}, trace: {1}", ntra, trace);
if (trace)
{
    fout2 = new StreamWriter(snaps, true); // true = append
}

// Read the flag that directs rewinding of the snapshot file.
//Console.WriteLine("4052: {0}", findata[4]);
//Console.WriteLine("====");
//Console.WriteLine("4052: {0}", findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
//rewi = bool.Parse(findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    rewi = true;
} else {
    rewi = false;
}
//Console.WriteLine("4052: {0}", rewi);
rewi = rewi & trace;

// Read the flag that directs stopping on any failure.
//sfatal = bool.Parse(findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    sfatal = true;
} else {
    sfatal = false;
}
//Console.WriteLine("4052a:");
// Read the flag that indicates whether error exists are to be tested.
//tsterr = bool.Parse(findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T") {
    tsterr = true;
} else {
    tsterr = false;
}

// Read the threshold value of the test ratio
thresh = double.Parse(findata[7].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

// Read and check the parameter values for the tests.
// Values of N
nidim = int.Parse(findata[8].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
//Console.WriteLine("H0");
if ((nidim < 1) || nidim > nidmax)
{
    Console.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "N", nidmax);
    Console.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); //
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "N", nidmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4095:");
    Environment.Exit(1);
}
//Console.WriteLine("H1");
for (int i = 1; i <= nidim; i = i + 1)
{
    idim[i - 1] = int.Parse(findata[9].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((idim[i - 1] < 0) || (idim[i - 1] > nmax))
    {
        fout1.WriteLine("VALUE OF N IS LESS THAN 0 OR GREATER THAN {0,2:D}", nmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4111:");
        Environment.Exit(1);
    }
}
//Console.WriteLine("H2");
// Values of ALPHA
nalf = int.Parse(findata[10].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nalf < 1) || (nalf > nalfmax))
{

```

```

foutl.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "ALPHA", nalmax);
foutl.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
if (trace)
{
    foutl.Flush();
    foutl.Close();
}
Console.WriteLine("4191:");
Environment.Exit(1);
}
for (int i = 1; i <= nalf; i = i + 1)
{
    alf[i - 1] = double.Parse(findata[11].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    //foutl.WriteLine("{0}. {1}", i, alf[i - 1]);
}

// Values of BETA
nbet = int.Parse(findata[12].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nbet < 1) || (nbet > nbemax))
{
    foutl.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "BETA", nbemax);
    foutl.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        foutl.Flush();
        foutl.Close();
    }
    Console.WriteLine("4210:");
    Environment.Exit(1);
}
for (int i = 1; i <= nbet; i = i + 1)
{
    bet[i - 1] = double.Parse(findata[13].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
}

// Report values of parameters
foutl.WriteLine("TESTS OF THE DOUBLE PRECISION LEVEL 3 BLAS \r\n\r\n THE FOLLOWING PARAMETER VALUES WILL BE USED:");
foutl.Write(" FOR N");
for (int i = 1; i <= nidim; i = i + 1)
{
    foutl.Write("{0,6:D}", idim[i - 1]);
}
foutl.WriteLine();
foutl.Write(" FOR ALPHA");
for (int i = 1; i <= nalf; i = i + 1)
{
    foutl.Write("{0,6:F1}", alf[i - 1]);
}
foutl.WriteLine();
foutl.Write(" FOR BETA");
for (int i = 1; i <= nbet; i = i + 1)
{
    foutl.Write("{0,6:F1}", bet[i - 1]);
}
foutl.WriteLine();

if (!tsterr)
{
    foutl.WriteLine();
    foutl.WriteLine("ERROR-EXITS WILL NOT BE TESTED");
}
foutl.WriteLine();
foutl.WriteLine("ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN {0,8:F2}", thresh);
foutl.WriteLine();

// Read names of subroutines and flags which indicate whether they are to be tested.
for (int i = 1; i <= nsubs; i = i + 1)
{
    ltest[i - 1] = false;
}

for (int i = 1; i <= nsubs; i = i + 1)
{
    sname = findata[14 + i - 1].Substring(0, 6);
    //ltestt = bool.Parse(findata[18 + i - 1].Substring(6, 2).Trim());
    if (findata[14 + i - 1].Substring(6, 2).Trim().ToUpper() == "T") {
        ltestt = true;
    } else {
        ltestt = false;
    }
    if (sname == snames[i - 1])
    {
        ltest[i - 1] = ltestt;
    }
    else
    {
        foutl.WriteLine("SUBPROGRAM NAME {0} NOT RECOGNIZED\n***** TESTS ABANDONED *****", sname);
        foutl.Flush();
        foutl.Close();
        Console.WriteLine("4276:");
        Environment.Exit(1);
    }
}

//Console.WriteLine("H6");
// Compute EPS (the machine precision).
// uses misc_d.eps, call DLAMCH first
//eps = misc_d.eps;
eps = misc_d.prec;
foutl.WriteLine("RELATIVE MACHINE PRECISION IS TAKEN TO BE {0,9:E1}", eps);
// Check the reliability of DMMCH using exact data.
n = Math.Min(32, nmax);
for (int j = 1; j <= n; j = j + 1)
{
    for (int i = 1; i <= n; i = i + 1)
    {
        ab[i - 1, j - 1] = Math.Max(i - j + 1, 0);
    }
    ab[j - 1, (nmax+1)-1] = j;
    ab[0, (nmax+j) - 1] = j;
    c[j-1,0] = zero;
}
for (int j = 1; j <= n; j = j + 1)

```

```

    cc[j - 1] = j * ((j + 1) * j) / 2 - ((j + 1) * j * (j - 1)) / 3;
}

// CC holds the exact result. On exit from DMMCH CT holds the result computed by DMMCH.
transa = "N";
transb = "N";
//DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX,      AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC,      NMAX, EPS, ERR, FATAL, NOUT, .TRUE.
)

DMMCH(transa,transb,n,1,n,one,ab,nmax, get_2dfromld(nmax*nmax, 0, nmax, get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-
1,ab)),nmax,zero,c,nmax,ref ct,ref g, get_2dfromld(nmax*nmax,0,nmax,cc),nmax,eps,ref err,ref fatal, foutl,true);
//DMMCH( TRANSA, TRANSB, M, N, KK, ALPHA, A, LDA, B, LDB, BETA, C, LDC, CT, G, CC, LDCC, EPS, ERR, FATAL, NOUT, MV )
//A( LDA, * ), B( LDB, * ), C( LDC, * ), CC( LDCC, * ), CT( * ), G( * )

same = LDE(cc,ct,n );
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}",transa,transb);
    foutl.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
transb = "T";
//DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX,      AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC,      NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
DMMCH(transa,transb,n,1,n,one,ab,nmax, get_2dfromld(nmax*nmax, 0, nmax, get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-
1,ab)),nmax,zero,c,nmax,ref ct,ref g, get_2dfromld(nmax*nmax,0,nmax,cc),nmax,eps,ref err, ref fatal,foutl,true);
same = LDE( cc,ct,n );
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}",transa,transb);
    foutl.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
for (int j = 1; j <= n; j = j + 1) {
    ab[j-1,(nmax+1)-1] = n-j+1;
    ab[0,(nmax+j)-1] = n-j+1;
}
for (int j = 1; j <= n; j = j + 1) {
    cc[(n-j+1)-1] = j*((j+1)*j)/(2-((j+1)*j*(j-1))/3);
}
transa = "T";
transb = "N";
//DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX,      AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC,      NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
DMMCH(transa,transb,n,1,n,one,ab,nmax, get_2dfromld(nmax*nmax, 0, nmax, get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-
1,ab)),nmax,zero,c,nmax,ref ct,ref g, get_2dfromld(nmax*nmax,0,nmax,cc),nmax,eps,ref err, ref fatal,foutl,true );
same = LDE(cc,ct,n);
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}",transa,transb);
    foutl.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
transb = "T";
//DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX,      AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC,      NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
DMMCH(transa,transb,n,1,n,one,ab,nmax, get_2dfromld(nmax*nmax, 0, nmax, get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-
1,ab)),nmax,zero,c,nmax,ref ct,ref g, get_2dfromld(nmax*nmax,0,nmax,cc),nmax,eps,ref err, ref fatal,foutl,true);

same = LDE( cc,ct,n );
if ((!same) || (err != zero))
{
    foutl.WriteLine("ERROR IN DMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    foutl.WriteLine("DMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}",transa,transb);
    foutl.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    foutl.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    foutl.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}

// Test each subroutine in turn
for (int isnum = 1; isnum <= nsubs; isnum = isnum + 1)
{
    //foutl.WriteLine("****SNAME={0} {1}", snames[isnum - 1],isnum);
    foutl.WriteLine();
    if (!ttest[isnum - 1])
    {
        // Subprogram is not to be tested
        foutl.WriteLine(" {0,6} WAS NOT TESTED", snames[isnum - 1]);
    }
    else {
        srnamt = snames[isnum - 1];
        // Test error exits
        if (tsterr)
        {
            DBLAT3_DCHKE(isnum, snames[isnum - 1], foutl);
            //foutl.WriteLine("end DCHKE");
            foutl.WriteLine();
        }
        //Console.WriteLine("H777");
        // Test computations
        infot = 0;
        ok = true;
        fatal = false;
        //foutl.Flush();
        //foutl.WriteLine("^^^isnum={0}", isnum);

        switch (isnum)
        //switch (isnum+50)
        {
            // case 1-5 ab in as [nmax,nmax]
            case 1:
                // Test DGEMM, 01

```

```

AB( 1, NMAX + 1 ), BB, BS, C, CC, CS, CT, G )
//DCHK1( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, AB, AA, AS,
//tmp_ab1 =
DBLAT3_DCHK1(snames[isnum-1],eps,thresh,fout1, fout2,trace,rewi,ref fatal,nidim,idim,nalf, alf,nbet,bet,nmax,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d(nmax*2*nmax,0,nmax,ab)),
aa,_as,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-1,ab)),bb,bs,c, cc,cs,ct,g);
// static void DBLAT3_DCHK1(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool
rewi, ref bool fatal, int nidim, int[] idim, int nalf, double[] alf, int nbet, double[] bet, int nmax,
// ref double[,] a,
// ref double[,] aa, double[] _as,
// ref double[,] b, ref double[] bb, double[] bs, ref double[,] c, ref double[] cc, double[]
// DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
// $ AS( NMAX*NMAX ), B( NMAX, NMAX ),
// $ BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
// $ C( NMAX, NMAX ), CC( NMAX*NMAX ),
// $ CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
break;
case 2:
// Test DSYMM, 02
//DCHK2( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, AB, AA, AS,
AB( 1, NMAX + 1 ), BB, BS, C,CC, CS, CT, G )

DBLAT3_DCHK2(snames[isnum-1],eps,thresh,fout1, fout2,trace,rewi,ref fatal,nidim,idim,nalf, alf,nbet,bet,nmax,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d(nmax*2*nmax,0,nmax,ab)),
aa,_as,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-1,ab)), bb,bs,c,cc,cs,ct,g); // in as b

break;
case 3:
case 4:
// Test DTRMM, 03, DTRSM, 04
//DCHK3( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NMAX, AB,AA, AS, AB( 1, NMAX +
1 ), BB, BS, CT, G, C )
//ab,aa,ab,bb

tmp2d_ab1 = get_2dfromld(nmax * nmax, 0, nmax, get_ldfrom2d(nmax * 2 * nmax, 0, nmax, ab));
tmp2d_ab2 = get_2dfromld(nmax * nmax, 0, nmax, get_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1, ab));
DBLAT3_DCHK3(snames[isnum-1],eps,thresh,fout1, fout2,trace,rewi,ref fatal,nidim,idim,nalf, alf,nmax,
ref tmp2d_ab1,
ref aa, ref as,
ref tmp2d_ab2,ref bb,ref bs,ct,g, ref c);
//may not be needed
for (int i = 1; i <= nmax; i = i + 1)
{
for (int j = 1; j <= nmax; j = j + 1)
{
ab[i-1,j-1] = tmp2d_ab1[i-1,j-1];
ab[i-1,nmax+j-1] = tmp2d_ab2[i-1,j-1];
}
}

break;
// Local Arrays
//double[] aa = new double[nmax * nmax];
//double[,] ab = new double[nmax,2*nmax];
//double[] alf = new double[nalfmax];
//double[] _as = new double[nmax * nmax];
//double[] Bb = new double[nmax*nmax];
//double[] bet = new double[nbemax];
//double[] bs = new double[nmax*nmax];
//double[,] c = new double[nmax,nmax];
//double[] cc = new double[nmax*nmax];
//double[] cs = new double[nmax*nmax];
//double[] ct = new double[nmax];
//double[] g = new double[nmax];
//double[] w = new double[2 * nmax];
//ab nmax,2*nmax
case 5:
// Test DSYRK, 05
//DCHK4( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, AB, AA, AS, AB(
1, NMAX + 1 ), BB, BS, C,CC, CS, CT, G )
//static double[] get_ldfrom2d_starti(int l1, int start1, int l2, int start2, double[,] arr)
//Environment.Exit(1);
//DCHK4( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM,
// IDIM, NALF, ALF, NBET, BET, NMAX,
// AB,
// AA, AS,
//AB( 1, NMAX + 1 ), BB, BS, C, CC, CS, CT, G )
DBLAT3_DCHK4(snames[isnum-1],eps,thresh,fout1, fout2,trace,rewi,ref fatal,nidim,
idim, nalf, alf, nbet,bet,nmax,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d(nmax*2*nmax,0,nmax,ab)),
aa,_as,
get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d_starti_full(nmax,0,2*nmax,nmax+1-1,ab)),bb,bs,c,cc,cs,ct,g);
break;

// DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
// $ AS( NMAX*NMAX ), B( NMAX, NMAX ),
// $ BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
// $ C( NMAX, NMAX ), CC( NMAX*NMAX ),
// $ CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )

//DBLAT3_DCHK4(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool
fatal, int nidim,
//int[] idim,int nalf, double[] alf, int nbet, double[] bet, int nmax,
// double[,] a,
//double[] aa, double[] _as,
//double[,] b, double[] Bb, double[] bs, double[,] c, double[] cc, double[] cs, double[] ct, double[] g)
//DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), AS( NMAX*NMAX ), B( NMAX, NMAX ), BB( NMAX*NMAX ), BET(
NBET ), BS( NMAX*NMAX ),
//C( NMAX, NMAX ), CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )

// case 6 ab in as [nmax*2*nmax]
case 6:
// Test DSYR2K, 06
// DCHK5( SNames( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, AB, AA, AS, BB,
BS, C, CC, CS, CT, G, W )
DBLAT3_DCHK5(snames[isnum-1],eps,thresh,fout1, fout2,trace,rewi,ref fatal,nidim,idim,nalf, alf,nbet,bet,nmax,

```

```

        get_1dfrom2d_starti_full(nmax,0,2*nmax,0,ab), aa,_as,bb,bs,c,cc,cs,ct,g,w);
    // DCHK5( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,
    // AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )

// DOUBLE PRECISION AA( NMAX*NMAX ), !!!!!AB( 2*NMAX*NMAX ),
// $
// $ ALF( NALF ),
// $ AS( NMAX*NMAX ), BB( NMAX*NMAX ),
// $ BET( NBET ), BS( NMAX*NMAX ), C( NMAX, NMAX ),
// $ CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ),
// $ G( NMAX ), !!!!!W( 2*NMAX )

// Local Arrays
//double[] aa = new double[nmax * nmax];
//double[,] ab = new double[nmax,2*nmax];
//double[] alf = new double[nalmax];
//double[] _as = new double[nmax * nmax];
//double[] bb = new double[nmax*nmax];
//double[] bet = new double[nbemax];
//double[] bs = new double[nmax*nmax];
//double[,] c = new double[nmax,nmax];
//double[] cc = new double[nmax*nmax];
//double[] cs = new double[nmax*nmax];
//double[] ct = new double[nmax];
//double[] g = new double[nmax];
//double[] w = new double[2 * nmax];
//ab nmax,2*nmax

        break;
    default:
        break;
    }
    fout1.Flush();
    if (fatal || sfatal)
    {
        fout1.WriteLine("\n***** FATAL ERROR - TESTS ABANDONED *****");
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        fout1.Flush();
        fout1.Close();

        Environment.Exit(1);
    }
}

}
fout1.WriteLine("\nEND OF TESTS");
if (trace)
{
    fout2.Flush();
    fout2.Close();
}

Console.WriteLine("end.");
fout1.Flush();
fout1.Close();
}

static void DBLAT3_DCHK1(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int nalf, double[] alf, int nbet, double[] bet, int nmax, double[,] a, double[] aa, double[] _as, double[,] b, double[] bb, double[] bs,
double[,] c, double[] cc, double[] cs, double[] ct, double[] g)
{
    //SUBROUTINE DCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, A, AA, AS, B, BB, BS, C, CC,
CS, CT, G )

    // Tests DGEMM

    // Parameters
    const double zero = 0.0;
    /*
    * .. Scalar Arguments ..
    DOUBLE PRECISION EPS, THRESH
    INTEGER NALF, NBET, NIDIM, NMAX, NOUT, NTRA
    LOGICAL FATAL, REWI, TRACE
    CHARACTER*6 SNAME
    * .. Array Arguments ..
    DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
    $ AS( NMAX*NMAX ), B( NMAX, NMAX ),
    $ BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
    $ C( NMAX, NMAX ), CC( NMAX*NMAX ),
    $ CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
    INTEGER IDIM( NIDIM )*/

    // Local Scalars
    double alpha, als, beta, bls, err, errmax;
    //int i, ia, ib, , m, n
    int m,n,k,ks,laa,lbb,lcc,lda,ldas,ldb,lbs,ldc,ldcs,ma,mb,ms,na,nargs,nb,nc,ns;
    bool _null,reset,same,trana,tranb;
    string tranas, tranbs, tranca, transb, ich;

    // Local Arrays
    bool[] isame = new bool[13];

    double[,] tmp_bet, tmp2d_aa, tmp2d_bb, tmp2d_cc;
    err = 0.0; // added
    ich = "NTC";

    // Executable Statements
    nargs = 13;
    nc = 0;
    reset = true;
    errmax = zero;
    //added
    ma = 0;

    //l10
    for (int im = 1; im <= nidim; im = im + 1) {
        m = idim[im-1];
        //l100
        for (int _in = 1; _in <= nidim; _in = _in + 1) {
            n = idim[_in-1];
            // Set LDC to 1 more than minimum value if room

```

```

ldc = m;
if (ldc < nmax) {
    ldc = ldc + 1;
}

// Skip tests if not enough room
if (ldc > nmax) {
    goto Loop100;
}
lcc = ldc*n;
_null = (n <= 0) || (m <= 0);

//90
for (int ik = 1; ik <= nidim; ik = ik + 1) {
    k = idim[ik-1];

//80
    for (int ica = 1; ica <= 3; ica = ica + 1) {
        transa = ich.Substring(ica-1,1).ToUpper();
        trana = (transa == "T") || (transa == "C");

        if (trana) {
            ma = k;
            na = m;
        } else {
            ma = m;
            na = k;
        }

        // Set LDA to 1 more than minimum value if room
        lda = ma;
        if (lda < nmax) {
            lda = lda + 1;
        }
        // Skip tests if not enough room
        if (lda > nmax) {
            //GO TO 80
            goto Loop80;
        }
        laa = lda*na;

        // Generate the matrix A
        //DMAKE( 'GE', ' ', ' ', ' ', MA, NA, A, NMAX, AA, LDA, RESET, ZERO )
        DBLAT3_DMAKE("GE", " ", " ", ma, na, ref a, nmax, ref aa, lda, ref reset, zero);

//70
        for (int icb = 1; icb <= 3; icb = icb + 1) {
            transb = ich.Substring(icb-1,1).ToUpper();
            tranb = (transb == "T") || (transb == "C");
            if (tranb) {
                mb = n;
                nb = k;
            } else {
                mb = k;
                nb = n;
            }

            // Set LDB to 1 more than minimum value if room
            ldb = mb;
            if (ldb < nmax) {
                ldb = ldb + 1;
            }

            // Skip tests if not enough room
            if (ldb > nmax) {
                //GO TO 70
                goto Loop70;
            }
            lbb = ldb*nb;

            // Generate the matrix B
            //DMAKE( 'GE', ' ', ' ', ' ', MB, NB, B, NMAX, BB, LDB, RESET, ZERO )
            //tmp_bet = get_2dfrom1d(nbet, 0, nmax, bet);
            DBLAT3_DMAKE("GE", " ", " ", mb, nb, ref b, nmax, ref bb, ldb, ref reset, zero);
            //bet = get_1dfrom2d(nbet, 0, nmax, tmp_bet);
            //static void DBLAT3_DMAKE(string type, string uplo, string diag, int m, int n, ref double[,] a, int nmax, ref double[] aa,
int lda, ref bool reset, double trans1)
            //DBLAT3_DMAKE("GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);

//60
            for (int ia = 1; ia <= nalf; ia = ia + 1) {
                alpha = alf[ia-1];
                //50
                for (int ib = 1; ib <= nbet; ib = ib + 1) {
                    beta = bet[ib-1];
                    // Generate the matrix C
                    //DMAKE( 'GE', ' ', ' ', ' ', M, N, C, NMAX, CC, LDC, RESET, ZERO )

                    DBLAT3_DMAKE( "GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);
                    nc = nc + 1;

                    // Save every datum before calling the subroutine
                    tranas = transa;
                    tranbs = transb;
                    ms = m;
                    ns = n;
                    ks = k;
                    als = alpha;
                    for (int i = 1; i <= laa; i = i + 1) {
                        _as[i-1] = aa[i-1];
                    }
                    ldas = lda;
                    for (int i = 1; i <= lbb; i = i + 1) {
                        bs[i-1] = bb[i-1];
                    }
                    ldbs = ldb;
                    bls = beta;
                    for (int i = 1; i <= lcc; i = i + 1) {
                        cs[i-1] = cc[i-1];
                    }
                    ldcs = ldc;

                    // Call the subroutine
                    if (trace) {

```

```

        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A {8,3:D}, B {9,3:D},
(10,4:F1}, C, {11,3:D}).",nc,sname,transb,m,n,k,alpha,lda,ldb,beta,ldc);
    }
    if (rewi) {
        //REWIND NTRA
    }
    //DGEMM( TRANSA, TRANSB, M, N, K, ALPHA, AA, LDA, BB, LDB, BETA, CC, LDC )
    tmp2d_cc= get_2dfromld(nmax*nmax, 0, ldc, cc);
    DGEMM(transa, transb, m, n, k, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb,
beta, ref tmp2d_cc, ldc);

    cc = get_ldfrom2d(nmax*nmax, 0, ldc, tmp2d_cc);

    // Check if error-exit was taken incorrectly
    if (!ok) {
        nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
        fatal = true;
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A {8,3:D}, B {9,3:D},
(10,4:F1}, C, {11,3:D}).",nc,sname,transa,transb,m,n,k,
        alpha, lda, ldb, beta, ldc);
        nout.Flush();
        return;
    }

    // See what data changed inside subroutines
    isame[0] = (transa == tranas);
    isame[1] = (transb == tranbs);
    isame[2] = (ms == m);
    isame[3] = (ns == n);
    isame[4] = (ks == k);
    isame[5] = (als == alpha);
    isame[6] = LDE( as,aa,laa);
    isame[7] = (ldas == lda);
    isame[8] = LDE( bs,bb,ldb );
    isame[9] = (ldbs == ldb);
    isame[10] = (bls == beta);
    if (_null) {
        isame[11] = LDE(cs,cc,lcc);
    } else {
        isame[11] = LDERES("GE", " ",m,n,get_2dfromld(nmax*nmax,0,ldc,cs),get_2dfromld(nmax*nmax,0,ldc,cc),ldc);
    }
    isame[12] = (ldcs == ldc);

    // If data was incorrectly changed, report and return
    same = true;
    for (int i = 1; i <= nargs; i = i + 1) {
        same = (same && isame[i-1]);
        if (!isame[i-1]) {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }
    if (!same) {
        fatal = true;
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A {8,3:D}, B {9,3:D},
(10,4:F1}, C, {11,3:D}).",nc,sname,transa,transb,m,n,k,
        alpha, lda, ldb, beta, ldc);
        nout.Flush();
        return;
    }
    if (!_null) {
        // Check the result
        //DMMCH( TRANSA, TRANSB, M, N, K,ALPHA, A, NMAX, B, NMAX, BETA,C, NMAX, CT, G, CC, LDC, EPS,ERR, FATAL, NOUT,
.TRUE. )
        DMMCH(transa, transb, m, n, k, alpha, a, nmax, b, nmax, beta, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldc,cc), ldc, eps, ref err, ref fatal, nout, true);

        errmax =Math.Max(errmax,err);

        // If got really bad answer, report and return
        if (fatal) {
            //GO TO 120
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A {8,3:D}, B {9,3:D},
(10,4:F1}, C, {11,3:D}).",nc,sname,transa,transb,m,n,k,
            alpha, lda, ldb, beta, ldc);
            nout.Flush();
            return;
        }
    }
}
} //50
} //60
Loop70: ;
} //70
Loop80:;
} //80
} //90
Loop100:;
} //100
} //110

// Report result
if (errmax < thresh) {
    nout.WriteLine("{0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)",sname,nc);
} else {
    nout.WriteLine("{0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT
*****",sname,nc,errmax);
}
}
static void DBLAT3_DCHK2(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int nalf, double[] alf, int nbet, double[] bet, int nmax, double[,] a, double[] aa, double[] _as, double[,] b, double[] bb, double[] bs,
double[,] c, double[] cc, double[] cs, double[] ct, double[] g)
{
    //DCHK2( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, A, AA, AS, B, BB, BS, C, CC, CS, CT, G )

```

```

// Tests DSYMM
// Auxiliary routine for test program for Level 3 Blas

// Parameters
const double zero = 0.0;
/*
* .. Scalar Arguments ..
DOUBLE PRECISION   EPS, THRESH
INTEGER            NALF, NBET, NIDIM, NMAX, NOUT, NTRA
LOGICAL            FATAL, REWI, TRACE
CHARACTER*6        SNAME
* .. Array Arguments ..
DOUBLE PRECISION   A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
$                 AS( NMAX*NMAX ), B( NMAX, NMAX ),
$                 BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
$                 C( NMAX, NMAX ), CC( NMAX*NMAX ),
$                 CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
INTEGER            IDIM( NIDIM )*/

// Local Scalars
double alpha, als, beta, bls, err, errmax;
//int i, ia, ib, k, n
int m, laa, lbb, lcc, lda, ldas, ldb, ldfs, ldc, ldcs, ms, na, nargs, nc, ns;
bool left, _null, reset, same;
string side, sides, uplo, uplos, ichs, ichu;

// Local Arrays
bool[] isame = new bool[13];

double[,] tmp2d_cc;

err = 0.0; // added
ichs = "LR";
ichu = "UL";

//COMMON          /INFOC/INFOT, NOUTC, OK, LERR

// Executable Statements
nargs = 12;
nc = 0;
reset = true;
errmax = zero;

//100
for (int im = 1; im <= nidim; im = im + 1) {
    m = idim[im-1];
    //90
    for (int _in = 1; _in <= nidim; _in = _in + 1) {
        n = idim[_in-1];
        // Set LDC to 1 more than minimum value if room
        ldc = m;
        if (ldc < nmax) {
            ldc = ldc + 1;
        }
        // Skip tests if not enough room
        if (ldc > nmax) {
            goto Loop90;
        }
        lcc = ldc*n;
        _null = (n <= 0) || (m <= 0);
        // Set LDB to 1 more than minimum value if room
        ldb = m;
        if (ldb < nmax) {
            ldb = ldb + 1;
        }
        // Skip tests if not enough room
        if (ldb > nmax) {
            goto Loop90;
        }
        lbb = ldb*n;

        // Generate the matrix B
        //DMAKE( 'GE', ' ', ' ', M, N, B, NMAX, BB, LDB, RESET, ZERO )
        DBLAT3_DMAKE("GE", " ", " ", m, n, ref b, nmax, ref bb, ldb, ref reset, zero);

        //80
        for (int ics = 1; ics <= 2; ics = ics + 1) {
            side = ichs.Substring(ics-1,1).ToUpper();
            left = (side == "L");
            if (left) {
                na = m;
            } else {
                na = n;
            }
            // Set LDA to 1 more than minimum value if room
            lda = na;
            if (lda < nmax) {
                lda = lda + 1;
            }
            // Skip tests if not enough room
            if (lda > nmax) {
                goto Loop80;
            }
            laa = lda*na;
            //70
            for (int icu = 1; icu <= 2; icu = icu + 1) {
                uplo = ichu.Substring(icu-1,1).ToUpper();

                // Generate the symmetric matrix A
                //DMAKE( 'SY', UPLO, ' ', NA, NA, A, NMAX, AA, LDA, RESET, ZERO )
                DBLAT3_DMAKE("SY", uplo, " ", na, na, ref a, nmax, ref aa, lda, ref reset, zero);

                //60
                for (int ia = 1; ia <= nalf; ia = ia + 1) {
                    alpha = alf[ia-1];
                    //50
                    for (int ib = 1; ib <= nbet; ib = ib + 1) {
                        beta = bet[ib-1];
                        // Generate the matrix C.
                        //DMAKE( 'GE', ' ', ' ', M, N, C, NMAX, CC, LDC, RESET, ZERO )
                        DBLAT3_DMAKE("GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);
                        nc = nc + 1;
                        // Save every datum before calling the subroutine.
                        sides = side;

```

```

        uplos = uplo;
        ms = m;
        ns = n;
        als = alpha;
        for (int i = 1; i <= laa; i = i + 1) {
            _as[i-1] = aa[i-1];
        }
        ldas = lda;
        for (int i = 1; i <= lbb; i = i + 1) {
            bs[i-1] = bb[i-1];
        }
        ldbs = ldb;
        bls = beta;
        for (int i = 1; i <= lcc; i = i + 1) {
            cs[i-1] = cc[i-1];
        }
        ldc = ldc;

// Call the subroutine
if (trace) {
    //ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, {8,4:F1}, C {9,3:D}).",
    ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
(10,3:D)).",
        nc,sname,side,uplo,m,n,
        alpha,lda,ldb,beta,ldc);
}
if (rewi) {
    //REWIND NTRA
}
tmp2d_cc = get_2dfromld(nmax * nmax, 0, ldc, cc);
DSYMM(side, uplo, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb, beta, ref
tmp2d_cc, ldc);
cc = get_ldfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);

// Check if error-exit was taken incorrectly
if (!ok) {
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO l10
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
(10,3:D)).",
        nc,sname,side,uplo,m,n,alpha,lda,ldb,beta,ldc);
}

// See what data changed inside subroutines
isame[0] = (sides == side);
isame[1] = (uplos == uplo);
isame[2] = (ms == m);
isame[3] = (ns == n);
isame[4] = (als == alpha);
isame[5] = LDE(_as,aa,laa);
isame[6] = (ldas == lda);
isame[7] = LDE(bs,bb,lbb);
isame[8] = (ldbs == ldb);
isame[9] = (bls == beta);

if (_null) {
    isame[10] = LDE(cs,cc,lcc);
} else {
    isame[10] = LDERES("GE", " ", m, n, get_2dfromld(nmax*nmax,0,ldc,cs), get_2dfromld(nmax*nmax,0,ldc,cc), ldc);
}
isame[11] = (ldcs == ldc);

// If data was incorrectly changed, report and return.
same = true;
for (int i = 1; i <= nargs; i = i + 1) {
    same = (same && isame[i-1]);
    if (!isame[i-1]) {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****",i);
    }
}
if (!same) {
    fatal = true;
    //GO TO l10
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
(10,3:D)).",
        nc,sname,side,uplo,m,n,alpha,lda,ldb,beta,ldc);
    //nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, {8,4:F1}, C {9,3:D}).",
    //nc,sname,side,uplo,m,n,alpha,lda,ldb,beta,ldc);
}
if (!_null) {
    // Check the result
    if (left) {
        //CALL DMMCH( 'N', 'N', M, N, M, ALPHA, A,NMAX, B, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR, FATAL, NOUT,
.TRUE. )
        DMMCH("N", "N", m, n, m, alpha, a, nmax, b, nmax, beta, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldc,cc), ldc, eps, ref err, ref fatal, nout, true);
    } else {
        // DMMCH( 'N', 'N', M, N, N, ALPHA, B, NMAX, A, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR, FATAL, NOUT,
.TRUE. )
        DMMCH("N", "N", m, n, n, alpha, b, nmax, a, nmax, beta, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldc,cc), ldc, eps, ref err, ref fatal, nout, true);
    }
}

errmax = Math.Max(errmax,err);
// If got really bad answer, report and return
if (fatal) {
    //GO TO l10
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
(10,3:D)).",
        nc,sname,side,uplo,m,n,alpha,lda,ldb,beta,ldc);
    //nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, {8,4:F1}, C {9,3:D}).",
    //nc,sname,side,uplo,m,n,alpha,lda,ldb,beta,ldc);
}
}
} //50
} //60
} //70

```

```

        Loop80;;
    } //80
    Loop90;;
} //90
} //100

// Report result
if (errmax < thresh) {
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)",sname,nc);
} else {
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT
*****",sname,nc,errmax);
}
}

static void DBLAT3_DCHK3(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int nalf, double[] alf, int nmax, ref double[,] a, ref double[] aa, ref double[] _as, ref double[,] b, ref double[] bb, ref double[] bs,
double[] ct, double[] g, ref double[,] c)
{
    //SUBROUTINE DCHK3( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NMAX, A, AA, AS, B, BB, BS, CT, G, C )
    // Tests DTRMM and DTRSM
    // Auxiliary routine for test program for Level 3 Blas

    // Parameters
    const double zero = 0.0;
    const double one = 1.0;

    /*
    DOUBLE PRECISION  A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
    $                  AS( NMAX*NMAX ), B( NMAX, NMAX ),
    $                  BB( NMAX*NMAX ), BS( NMAX*NMAX ),
    $                  C( NMAX, NMAX ), CT( NMAX ), G( NMAX )*/

    // Local Scalars
    double alpha,als,err,errmax;
    //int i,ia,icd,ics,ict,icu,im,_in,j,laa,lbb,lda,ldas,ldb,lbs,m,ms,n,na,nargs,nc,ns;
    int laa, lbb, lda, ldas, ldb, lbs, m, ms, n, na, nargs, nc, ns;
    bool left, _null, reset, same;
    string diag, diags, side, sides, tranas, transa, uplo, uplos, ichd, ichs, ichu, icht;

    // Local Arrays
    bool[] isame = new bool[13];

    // COMMON          /INFOC/INFOT, NOUTC, OK, LERR

    ichu = "UL";
    icht = "NTC";
    ichd = "UN";
    ichs = "LR";

    double[,] tmp2d_bb;

    // Executable Statements
    nargs = 11;
    nc = 0;
    reset = true;
    errmax = zero;
    err = 0; //added

    // Set up zero matrix for DMMCH.
    for (int j = 1; j <= nmax; j = j + 1) {
        for (int i = 1; i <= nmax; i = i + 1) {
            c[i-1,j-1] = zero;
        }
    }
    //140

    for (int im = 1; im <= nidim; im = im + 1) {
        m = idim[im-1];
        //130
        for (int _in = 1; _in <= nidim; _in = _in + 1) {
            n = idim[_in-1];
            // Set LDB to 1 more than minimum value if room
            ldb = m;
            if (ldb < nmax) {
                ldb = ldb + 1;
            }
            // Skip tests if not enough room
            if (ldb > nmax) {
                //GO TO 130
                goto Loop130;
            }
            lbb = ldb*n;
            _null = (m <= 0) || (n <= 0);
            //120
            //ichs="LR"
            for (int ics = 1; ics <= 2; ics = ics + 1) {
                side = ichs.Substring(ics-1,1).ToUpper();
                left = (side == "L");
                if (left) {
                    na = m;
                } else {
                    na = n;
                }
                // Set LDA to 1 more than minimum value if room
                lda = na;
                if (lda < nmax) {
                    lda = lda + 1;
                }
                // Skip tests if not enough room
                if (lda > nmax) {
                    //GO TO 130
                    goto Loop130;
                }
                laa = lda*na;
                //110
                //ichu="UL"
                for (int icu = 1; icu <= 2; icu = icu + 1) {
                    uplo = ichu.Substring(icu-1,1).ToUpper();
                    //100
                    //icht = "NTC";
                    for (int ict = 1; ict <= 3; ict = ict + 1) {
                        transa = icht.Substring(ict-1,1).ToUpper();

```

```

//90
//ichd = "UN";
for (int icd = 1; icd <= 2; icd = icd + 1) {
    diag = ichd.Substring(icd-1,1).ToUpper();
    //80
    for (int ia = 1; ia <= nalf; ia = ia + 1) {
        alpha = alf[ia-1];

        // Generate the matrix A.
        //DMAKE( 'TR', UPLO, DIAG, NA, NA, A, NMAX, AA, LDA, RESET, ZERO )

        //nout.WriteLine("b1 ia={0}", ia);
        DBLAT3_DMAKE("TR", uplo, diag, na, na, ref a, nmax, ref aa, lda, ref reset, zero);
        //nout.WriteLine("a1 ia={0}", ia);
        // Generate the matrix B.
        //DMAKE( 'GE', ' ', ' ', M, N, B, NMAX, BB, LDB, RESET, ZERO )
        //nout.WriteLine("b2 ia={0}", ia);
        //if (icd == 2 && ia == 2)
        //    print alpha,a,lda,b,ldb,beta,c,ldc - starts 6th argument
        //    //one, a,nmax,c,nmax,zero,b,nmax

        // nout.WriteLine("a=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, a[zz, zz1]);
        //     }
        // }
        // nout.WriteLine("aa=");
        // for (int zz = 0; zz < nmax*nmax; zz = zz + 1)
        // {
        //     nout.WriteLine("{0}: {1}", zz, aa[zz]);
        // }

        // nout.Flush();

        //}
        //if (true) //sname.Substring(0,5) == "DTRSM" && nc > 502)
        //    print alpha,a,lda,b,ldb,beta,c,ldc - starts 6th argument
        //    //one, a,nmax,c,nmax,zero,b,nmax

        // nout.WriteLine("a=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, a[zz, zz1]);
        //     }
        // }
        // nout.WriteLine("aa=");
        // for (int zz = 0; zz < nmax * nmax; zz = zz + 1)
        // {
        //     nout.WriteLine("{0}: {1}", zz, aa[zz]);
        // }

        //}

        DBLAT3_DMAKE("GE", " ", " ", m, n, ref b, nmax, ref bb, ldb, ref reset, zero);

        //nout.WriteLine("a2 ia={0}", ia);
        //nout.WriteLine("=im:{2} of {0}, in:{3} of {0}, ics:{4} of 2, icu:{5} of 2, ict:{6} of 3, icd:{7} of 2, ia:{8} of
        //    {1}",
        //        nidim,nalf, im, in,ics,icu,ict, icd,ia);
        //nout.Flush();
        //if (true) //sname.Substring(0,5) == "DTRSM" && nc > 502)
        //    print alpha,a,lda,b,ldb,beta,c,ldc - starts 6th argument
        //    //one, a,nmax,c,nmax,zero,b,nmax

        // nout.WriteLine("b=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, b[zz, zz1]);
        //     }
        // }
        // nout.WriteLine("bb=");
        // for (int zz = 0; zz < nmax * nmax; zz = zz + 1)
        // {
        //     nout.WriteLine("{0}: {1}", zz, bb[zz]);
        // }

        // nout.Flush();
        // nout.Close();
        // Environment.Exit(1);

        //}
        //nout.Flush();
        // nout.Close();
        //Environment.Exit(1);
        //nout.WriteLine("nc={0}", nc);
        nc = nc + 1;
        //im, in, ics, icu,ict,icd,ia
        //nidim,nidim,2,2,3,2,nalf(3)
        //left,uplo,transa,diag
        //lunn

        //Console.WriteLine("nc = {0} {1} im:{2} in:{3} ics:{4} icu:{5} ict:{6} icd:{7} ia:{8} {9} {10}",
        //    nc,sname,im,in,ics,icu,ict,icd,ia,nidim,left);

        // Save every datum before calling the subroutine
        sides = side;
        uplos = uplo;
        tranas = transa;
        diags = diag;
        ms = m;
        ns = n;

```

```

als = alpha;
for (int i = 1; i <= laa; i = i + 1) {
    _as[i-1] = aa[i-1];
}
ldas = lda;
for (int i = 1; i <= lbb; i = i + 1) {
    bs[i-1] = bb[i-1];
}
ldbs = ldb;

// Call the subroutine
if (sname.Substring(3,2).ToUpper() == "MM") {
    if (trace) {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8,4:F1}, A {9,3:D}, B
(10,3:D))      .",
            nc, sname, side, uplo, transa, diag, m, n, alpha, lda, ldb);
    }
    if (rewi) {
        //REWIND NTRA
    }
    //DTRMM( SIDE, UPLO, TRANSA, DIAG, M,      N, ALPHA, AA, LDA, BB, LDB )
    //Console.WriteLine("in DTRMM");
    tmp2d_bb = get_2dfromld(nmax * nmax, 0, ldb, bb);
    DTRMM(side,uplo, transa, diag, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
    bb = get_ldfrom2d(nmax * nmax, 0, ldb, tmp2d_bb);
    //Console.WriteLine("out DTRMM");
} else if (sname.Substring(3,2).ToUpper() == "SM") {
    if (trace) {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8,4:F1}, A {9,3:D}, B
(10,3:D))      .",
            nc, sname, side, uplo, transa, diag, m, n, alpha, lda, ldb);
    }
    if (rewi) {
        //REWIND NTRA
    }
    //DTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, AA, LDA, BB, LDB )
    //Console.WriteLine("in DTRSM");
    //Console.WriteLine("In DTRSM {0}", nc);
    //nout.WriteLine("im:{2} of {0}, in:{3} of {0}, ics:{4} of 2, icu:{5} of 2, ict:{6} of 3, icd:{7} of 2, ia:{8}
of {1}",
        //
        nidim,nalf, im,in,ics,icu,ict,icd,ia);
    //nout.WriteLine("m={0}, n={1}, side={2}, uplo={3}, transa={4}, diag={5}, alpha={6}", m, n, side, uplo, transa,
diag, alpha);
    // m = idim[im-1];
    // n = idim[in-1];
    // side = ics.Substring(ics-1,1).ToUpper();
    // uplo = ichu.Substring(icu-1,1).ToUpper();
    // transa = icht.Substring(ict-1,1).ToUpper();
    // diag = ichd.Substring(icd-1,1).ToUpper();
    // alpha = alf[ia-1];
    tmp2d_bb = get_2dfromld(nmax * nmax, 0, ldb, bb);
    DTRSM(side,uplo, transa, diag, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
    bb = get_ldfrom2d(nmax * nmax, 0, ldb, tmp2d_bb);
    //Console.WriteLine("Out DTRSM {0}", nc);
    //Console.WriteLine("out DTRSM");
}

// Check if error-exit was taken incorrectly.
if (!ok) {
    nout.WriteLine( "***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 150
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8,4:F1}, A {9,3:D}, B {10,3:D})
.",
        nc, sname, side, uplo, transa, diag, m, n, alpha, lda, ldb);
}

// See what data changed inside subroutines
isame[0] = (sides == side);
isame[1] = (uplos == uplo);
isame[2] = (transa == transa);
isame[3] = (diags == diag);
isame[4] = (ms == m);
isame[5] = (ns == n);
isame[6] = (als == alpha);
isame[7] = LDE(_as, aa, laa);
isame[8] = (ldas == lda);

if (_null) {
    isame[9] = LDE(bs,bb,lbb);
} else {
    isame[9] = LDERES("GE", " ", m, n, get_2dfromld(nmax*nmax,0,ldb,bs), get_2dfromld(nmax*nmax,0,ldb,bb), ldb);
}
isame[10] = (ldbs == ldb);

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1) {
    same = (same && isame[i-1]);
    if (!isame[i-1]) {
        if (left) {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****",i);
        }
    }
}
if (!same) {
    fatal = true;
    //GO TO 150
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8,4:F1}, A {9,3:D}, B {10,3:D})
.",
        nc, sname, side, uplo, transa, diag, m, n, alpha, lda, ldb);
}

if (! null) {
    if (sname.Substring(3,2).ToUpper() == "MM") {
        // Check the result
        if (left) {
            //DMMCH( TRANSA, 'N', M, N, M, ALPHA, A, NMAX, B, NMAX, ZERO, C, NMAX, CT, G, BB, LDB, EPS, ERR, FATAL,
NOUT, .TRUE. )

```

```

        DMMCH(transa, "N", m, n, m, alpha, a, nmax, b, nmax, zero, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout,true);
    } else {
        //DMMCH( 'N', TRANSA, M, N, N, ALPHA, B, NMAX, A, NMAX, ZERO, C, NMAX, CT, G, BB, LDB, EPS, ERR, FATAL,
NOUT, .TRUE. )
        DMMCH("N", transa, m, n, n, alpha, b, nmax, a, nmax, zero, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout, true);
    }
} else if (sname.Substring(3,2).ToUpper() == "SM") {
    // Compute approximation to original matrix.
    for (int j = 1; j <= n; j = j + 1) {
        for (int i = 1; i <= m; i = i + 1) {
            c[i-1,j-1] = bb[(i+(j-1)*ldb) -1];
            bb[(i+(j-1)*ldb) -1] = alpha*b[i-1,j-1];
        }
    }
    if (left)
    {
        #region comment
        //nout.WriteLine("ia={0} of {1}, icd={2} of 2", ia, nalf,icd);
        //nout.WriteLine("Left SM ");
        //DMMCH( TRANSA, 'N', M, N, M, ONE, A, NMAX, C, NMAX, ZERO, B, NMAX, CT, G, BB, LDB, EPS, ERR, FATAL,
NOUT, .FALSE. )
        //if (icd == 2 && ia == 2)
        //{// print alpha,a,lda,b,ldb,beta,c,ldc - starts 6th argument
        // //one, a,nmax,c,nmax,zero,b,nmax
        // nout.WriteLine("alpha,beta={0}, {1}",one,zero);
        // nout.WriteLine("a=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, a[zz, zz1]);
        //     }
        // }
        // nout.WriteLine("b=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, c[zz, zz1]);
        //     }
        // }
        // nout.WriteLine("c=");
        // for (int zz = 0; zz < nmax; zz = zz + 1)
        // {
        //     for (int zz1 = 0; zz1 < nmax; zz1 = zz1 + 1)
        //     {
        //         nout.WriteLine("{0},{1}: {2}", zz, zz1, b[zz, zz1]);
        //     }
        // }
        // nout.Flush();
        // nout.Close();
        // Environment.Exit(1);
        //}
        #endregion
        //---DMMCH(transa, "N", m, n, m, alpha, a, nmax, b, nmax, zero, c, nmax, ref ct, ref g, get_2dfromld(nmax
* nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
        DMMCH(transa, "N", m, n, m, one, a, nmax, c, nmax, zero, b, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb),ldb, eps, ref err, ref fatal, nout, false);
        //nout.WriteLine("Out SM ");
    } else {
        //nout.WriteLine("Right SM ");
        //DMMCH( 'N', TRANSA, M, N, N, ONE, C, NMAX, A, NMAX, ZERO, B, NMAX, CT, G, BB, LDB, EPS, ERR, FATAL,
NOUT, .FALSE. )
        //---DMMCH("N", transa, m, n, n, alpha, b, nmax, a, nmax, zero, c, nmax, ref ct, ref g, get_2dfromld(nmax
* nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
        DMMCH("N", transa, m, n, n, one, c, nmax, a, nmax, zero, b, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout, false);
        //nout.WriteLine("Out SM ");
    }
}
errmax = Math.Max(errmax,err);
// If got really bad answer, report and return
if (fatal) {
    // GO TO 150
    // Report result
    if (errmax < thresh)
    {
        nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
    }
    else
    {
        nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST
RATIO {2,8:F2} - SUSPECT *****", sname, nc, errmax);
    }
    return;
}
}
} //80 ia
} //90
} //100
} //110
} //120
Loop130: ;
} //130
} //140
// Report result
if (errmax < thresh) {
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname,nc);
} else {
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST
RATIO {2,8:F2} - SUSPECT *****",
sname,nc, errmax);
}
}

static void DBLAT3_DCHK4(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int nalf, double[] alf, int nbet, double[] bet, int nmax, double[] a, double[] aa, double[] _as, double[] b, double[] bb, double[] bs,
double[], c, double[] cc, double[] cs, double[] ct, double[] g)
{

```

```

//SUBROUTINE DCHK4( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, A, AA, AS, B, BB, BS, C, CC,
CS, CT, G )
// Tests DSYSRK
// Auxiliary routine for test program for Level 3 Blas
// Parameters
const double zero = 0.0;
/*
* .. Array Arguments ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
$ AS( NMAX*NMAX ), B( NMAX, NMAX ),
$ BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
$ C( NMAX, NMAX ), CC( NMAX*NMAX ),
$ CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
* */

// Local Scalars
double alpha, als, beta, bets, err, errmax;
//int i, ia, ib, ict, icu, ik, _in, j, jc, jj, k, ks, laa, lcc, lda, ldas, ldc, ldc, ldc, ljd, ma, n, na, nargs, nc, ns;
int k, _in, jc, jj, ks, laa, lcc, lda, ldas, ldc, ldc, ljd, ma, n, na, nargs, nc, ns;
bool _null, reset, same, tran, upper;
string trans, transs, uplo, uplos, ichu, icht;

bool[] isame = new bool[13];
double[,] tmp2d_cc;
//COMMON /INFOC/INFOT, NOUTC, OK, LERR

icht = "NTC";
ichu = "UL";

// Executable Statements
nargs = 10;
nc = 0;
reset = true;
errmax = zero;
err = 0; //added

//100
for ( _in = 1; _in <= nidim; _in = _in + 1 ) {
    n = idim[_in-1];
    // Set LDC to 1 more than minimum value if room
    ldc = n;
    if ( ldc < nmax ) {
        ldc = ldc + 1;
    }
    // Skip tests if not enough room
    if ( ldc > nmax ) {
        goto Loop100;
    }
    lcc = ldc*n;
    _null = ( n <= 0 );
    //90
    for ( int ik = 1; ik <= nidim; ik = ik + 1 ) {
        k = idim[ik-1];
        //80
        for ( int ict = 1; ict <= 3; ict = ict + 1 ) {
            trans = icht.Substring(ict-1,1).ToUpper();
            tran = ((trans == "T") || (trans == "C"));
            if ( tran ) {
                ma = k;
                na = n;
            } else {
                ma = n;
                na = k;
            }

            // Set LDA to 1 more than minimum value if room
            lda = ma;
            if ( lda < nmax ) {
                lda = lda + 1;
            }
            // Skip tests if not enough room
            if ( lda > nmax ) {
                //GO TO 80
                goto Loop80;
            }
            laa = lda*na;

            // Generate the matrix A
            //DMAKE( 'GE', ' ', ' ', ' ', MA, NA, A, NMAX, AA, LDA, RESET, ZERO )
            DBLAT3_DMAKE("GE", " ", " ", " ", ma, na, ref a, nmax, ref aa, lda, ref reset, zero);
            for ( int icu = 1; icu <= 2; icu = icu + 1 ) {
                uplo = ichu.Substring(icu-1,1).ToUpper();
                upper = (uplo == "U");
                //60
                for ( int ia = 1; ia <= nalf; ia = ia + 1 ) {
                    alpha = alf[ia-1];
                    //50
                    for ( int ib = 1; ib <= nbet; ib = ib + 1 ) {
                        beta = bet[ib-1];

                        // Generate the matrix C
                        //DMAKE( 'SY', UPLO, ' ', N, N, C, NMAX, CC, LDC, RESET, ZERO )
                        DBLAT3_DMAKE("SY", uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);

                        nc = nc + 1;

                        //Save every datum before calling the subroutine.
                        uplos = uplo;
                        transs = trans;
                        ns = n;
                        ks = k;
                        als = alpha;
                        for ( int i = 1; i <= laa; i = i + 1 ) {
                            _as[i-1] = aa[i-1];
                        }
                        ldas = lda;
                        bets = beta;
                        for ( int i = 1; i <= lcc; i = i + 1 ) {
                            cs[i-1] = cc[i-1];
                        }
                        ldc = ldc;

```

```

// Call the subroutine
if (trace) {
    ntra.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C, {9,3:D})
        nc, sname, uplo, trans,n,k,alpha,lda,beta,ldc);
}
if (rewi) {
    //REWIND NTRA
}
//DSYRK( UPLD, TRANS, N, K, ALPHA, AA, LDA, BETA, CC, LDC )
//Console.WriteLine("bef 4a");

//nout.WriteLine("=in:{0} of {1}, ik:{2} of {3}, ict:{4} of 3, icu:{5} of 2, ia:{6} of {7}, ib:{8} of {9}",
//    _in, nidim, ik, nidim, ict,icu,ia,nalf,ib,nbet);
//nout.WriteLine("n={0}, k={1}, trans={2}, uplo={3}, alpha={4}, beta={5}", n, k, trans, uplo, alpha,beta);

//_in nidim n
//_ik nidim k
//_ict 3 trans
//_icu 2 uplo
//_ia nalf alpha
//_ib nbet beta

tmp2d_cc = get_2dfromld(nmax * nmax, 0, ldc, cc);
DSYRK(uplo, trans, n, k, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, beta, ref tmp2d_cc, ldc);
cc = get_ldfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);
//nout.WriteLine("aft DSYRK");
//DSYRK(string uplo, string trans, int n, int k, double alpha,
//double[, ] a,
//int lda, double beta, ref double[, ] c, int ldc

//
//$      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
//$      AS( NMAX*NMAX ), B( NMAX, NMAX ),
//$      BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
//$      C( NMAX, NMAX ), CC( NMAX*NMAX ),
//$      CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )

// Check if error-exit was taken incorrectly
if (!ok) {
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C, {9,3:D})
        nc, sname, uplo, trans, n, k, alpha, lda, beta, ldc);
}

//See what data changed inside subroutines.
isame[0] = (uplos == uplo);
isame[1] = (transs == trans);
isame[2] = (ns == n);
isame[3] = (ks == k);
isame[4] = (als == alpha);
isame[5] = LDE(_as, aa, laa);
isame[6] = (ldas == lda);
isame[7] = (bets == beta);
if (_null) {
    isame[8] = LDE(cs, cc, lcc);
} else {
    isame[8] = LDERES("SY",uplo, n, get_2dfromld(nmax*nmax,0,ldc,cs), get_2dfromld(nmax*nmax,0,ldc,cc), ldc);
}
isame[9] = (ldcs == ldc);

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1) {
    same = (same && isame[i-1]);
    if (!isame[i-1]) {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****",i);
    }
}
if (!same) {
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C, {9,3:D})
        nc, sname, uplo, trans, n, k, alpha, lda, beta, ldc);
}
if (!_null) {
    // Check the result column by column
    jc = 1;
    //40
    for (int j = 1; j <= n; j = j + 1) {
        if (upper) {
            jj = 1;
            lj = j;
        } else {
            jj = j;
            lj = n-j+1;
        }
        if (tran) {
            //DMMCH( 'T', 'N', LJ, 1, K, ALPHA, A( 1, JJ ), NMAX, A( 1, J ), NMAX, BETA, C( JJ, J ), NMAX, CT, G, CC(
JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
            //DMMCH( TRANSA, TRANSB, M, N, KK, ALPHA, A, LDA, B, LDB,BETA, C, LDC, CT, G, CC, LDCC, EPS, ERR,
FATAL,NOUT, MV )
            //DOUBLE PRECISION  A( LDA, * ), B( LDB, * ), C( LDC, * ), CC( LDCC, * ), CT( * ), G( * )
            //double[] get_ldfrom2d(int l, int start, int lda, double[, ] arr)
            //double[, ] get_2dfromld(int l, int start, int lda, double[] arr)
            //double[] get_darr(int start, int end, double[] arr)
            //Console.WriteLine("bef 4b");
            DMMCH("T", "N", lj, 1, k, alpha, get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, 0, nmax,jj-
1, a)),nmax,

```

```

        get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, 0,nmax, j-1, a)),nmax, beta,
        get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1, nmax,j-1, c)),nmax,
        ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1,nmax*nmax-1, cc)), ldc, eps,
ref err, ref fatal, nout, true);

//Console.WriteLine("aft 4b");

} else {
//DMMCH( 'N', 'T', LJ, 1, K, ALPHA, A( JJ, 1 ), NMAX, A( J, 1 ), NMAX, BETA, C( JJ, J ), NMAX, CT, G, CC(
JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
//Console.WriteLine("bef 4c1");
//double[,] g1 = get_ldfrom2d(nmax * nmax, jj - 1, nmax, a);
//double[,] g12d = get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d(nmax * nmax, jj - 1, nmax, a));
//Console.WriteLine("bef 4c2");
//double[,] g2 = get_ldfrom2d(nmax * nmax, j - 1, nmax, a);
//double[,] g22d = get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d(nmax * nmax, j - 1, nmax, a));
//Console.WriteLine("bef 4c3");
//double[,] g3 = get_ldfrom2d_starti(nmax, jj - 1,nmax, j - 1, c);
//Console.WriteLine("bef 4c3-1");
//double[,] g32d = get_2dfromld(nmax*nmax,0,nmax,get_ldfrom2d_starti_full(nmax, jj - 1, nmax, j - 1, c));

//tatic double[] get_ldfrom2d_starti(int l1, int start1, int l2, int start2, double[,] arr)
//Console.WriteLine("bef 4c");
DMMCH("N", "T", lj, 1, k, alpha, get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1,
nmax,0, a)), nmax,
        get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, j-1, nmax,0,a)),nmax, beta,
        get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1, nmax,j-1, c)),nmax,
        ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1,nmax*nmax-1, cc)), ldc, eps,
ref err, ref fatal, nout, true);

//Console.WriteLine("aft 4c");
//ldfrom2d static double[] get_ldfrom2d(int l, int start, int lda, double[,] arr)
//2dfromld static double[,] get_2dfromld(int l, int start, int lda, double[,] arr)
//getdarr static double[] get_darr(int start, int end, double[,] arr)
//DMMCH(string transa, string transb, int m, int n, int kk, double alpha,
//double[,] a, int lda, A(JJ,1) [NMAX,NMAX]
//double[,] b, int ldb, double beta, A(J,1) [NMAX,NMAX]
//double[,] c, int ldc, ref double[] ct, ref double[] g, C(JJ,J) [NMAX,NMAX]
//double[,] cc, int ldcc, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv CC(JC)
[NMAX*NMAX]

}
if (upper) {
    jc = jc + ldc;
} else {
    jc = jc + ldc + 1;
}
errmax = Math.Max(errmax,err);

// If got really bad answer, report and return
if (fatal) {
    //GO TO 110
    if (n > 1) {
        nout.WriteLine( "          THESE ARE THE RESULTS FOR COLUMN {0,3:D}",j);
    }
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:",sname);
    nout.WriteLine(" {0,6:D}: (1,6) ((2,1), {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C,
(9,3:D))      .",
        nc, sname, uplo, trans, n, k, alpha, lda, beta, ldc);
    }
} //40
} //50
} //60
} //70
Loop80: ;
} //80
} //90
Loop100: ;
} //100

// Report result
if (errmax < thresh) {
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ((1,6:D) CALLS)",sname, nc);
} else {
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ((1,6:D) CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
}
}
static void DBLAT3_DCHK5(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
int nalf, double[] alf, int nbet, double[] bet, int nmax, double[] ab, double[] aa, double[] _as, double[] bb, double[] bs, double[,] c,
double[] cc, double[] cs, double[] ct, double[] g, double[] w)
{
    //DCHK5( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )
    // Tests DSYR2K
    // Auxiliary routine for test program for Level 3 Blas

    // Parameters
    const double zero = 0.0;
    /*
    DOUBLE PRECISION AA( NMAX*NMAX ), AB( 2*NMAX*NMAX ),
    $ ALF( NALF ), AS( NMAX*NMAX ), BB( NMAX*NMAX ),
    $ BET( NBET ), BS( NMAX*NMAX ), C( NMAX, NMAX ),
    $ CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ),
    $ G( NMAX ), W( 2*NMAX )*/

    // Local Scalars
    double alpha, als, beta, bets, err, errmax;
    //int i, ia, ib, ict, icu, ik, _in, j, jc, jj, jjab, k, ks, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
    int jc, jj, jjab, k, ks, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
    bool _null, reset, same, tran, upper;
    string trans, transs, uplo, uplos, ichu, icht;

    // Local Arrays
    bool[] isame = new bool[13];
    //COMMON /INFOC/INFOT, NOUTC, OK, LERR
    double[,] tmp_ab;
    double[] tmp_ab_ld;
    icht = "NTC";
    ichu = "UL";
    double[,] tmp2d_cc;

    // Executable Statements
    nargs = 12;

```

```

nc = 0;
reset = true;
errmax = zero;
err = 0; //added

//130
for (int _in = 1; _in <= nidim; _in = _in + 1) {
    n = idim[_in-1];
    // Set LDC to 1 more than minimum value if room
    ldc = n;
    if (ldc < nmax) {
        ldc = ldc + 1;
    }
    // Skip tests if not enough room
    if (ldc > nmax) {
        //GO TO 130
        goto Loop130;
    }
    lcc = ldc*n;
    _null = (n <= 0);

//120
for (int ik = 1; ik <= nidim; ik = ik + 1) {
    k = idim[ik-1];
    //110
    for (int ict = 1; ict <= 3; ict = ict + 1) {
        trans = icht.Substring(ict-1,1).ToUpper();
        tran = (trans == "T") || (trans == "C");
        if (tran) {
            ma = k;
            na = n;
        } else {
            ma = n;
            na = k;
        }
        // Set LDA to 1 more than minimum value if room
        lda = ma;
        if (lda < nmax) {
            lda = lda + 1;
        }
        // Skip tests if not enough room
        if (lda > nmax) {
            //GO TO 110
            goto Loop110;
        }
        laa = lda*na;

        // Generate the matrix A
        if (tran) {
            //DMAKE( 'GE', ' ', ' ', MA, NA, AB, 2*NMAX, AA, LDA, RESET, ZERO )
            //Console.WriteLine("bef 5a");

            tmp_ab = get_2dfromld(2 * nmax * nmax, 0, 2 * nmax, ab);
            DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2*nmax, ref aa, lda, ref reset, zero);
            //Console.WriteLine("bef 1a");
            ab = get_ldfrom2d(2 * nmax * nmax, 0, 2 * nmax, tmp_ab);
            //Console.WriteLine("aft 1a");
            //Console.WriteLine("aft 5a");
        } else {
            //DMAKE( 'GE', ' ', ' ', MA, NA, AB, NMAX, AA, LDA, RESET, ZERO )
            //Console.WriteLine("bef 5b");
            tmp_ab = get_2dfromld(2 * nmax * nmax, 0, nmax, ab);
            DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref aa, lda, ref reset, zero);
            //Console.WriteLine("bef 2a");
            ab = get_ldfrom2d(2 * nmax * nmax, 0, nmax, tmp_ab);
            //Console.WriteLine("aft 2a");
            //Console.WriteLine("aft 5b");
        }
    }

    // Generate the matrix B
    ldb = lda;
    lbb = laa;
    if (tran) {
        //DMAKE( 'GE', ' ', ' ', MA, NA, AB( K + 1 ), 2*NMAX, BB, LDB, RESET, ZERO )
        //ic double[,] get_2dfromld(int l, int start, int lda, double[] arr)
        //Console.WriteLine("bef 5c");
        tmp_ab = get_2dfromld(2*nmax*nmax, 0, 2*nmax, get_darr_full(2*nmax*nmax, k+1-1, 2*nmax*nmax-1, ab));
        DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2*nmax, ref bb, ldb, ref reset, zero);
        tmp_ab_ld = get_ldfrom2d(2 * nmax * nmax, 0, 2 * nmax, tmp_ab);
        copy_ld(tmp_ab_ld, 0, 2*nmax*nmax-(k+1)+1, ab, k+1-1);
        //static void Copy_ld(double[] src1, int s1, int l1, double[] src2, int s2)
        //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2 * nmax, ref bb, ldb, ref reset, zero);
        //tmp_ab_ld = get_ldfrom2d(2 * nmax * nmax - (k + 1) + 1, 0, 2 * nmax, tmp_ab);
        //copy_ld(tmp_ab_ld, 0, 2 * nmax * nmax - (k + 1) + 1, ab, k + 1 - 1);
        //Console.WriteLine("aft 5c");
        //get_2dfromld(nmax*nmax, 0, ldc, get_darr_full(nmax*nmax, jc-1, nmax*nmax-1, cc));
    } else {
        //DMAKE( 'GE', ' ', ' ', MA, NA, AB( K*NMAX + 1 ), NMAX, BB, LDB, RESET, ZERO )
        //Console.WriteLine("bef 6c");
        tmp_ab = get_2dfromld(2 * nmax * nmax, 0, nmax, get_darr_full(2*nmax*nmax, k*nmax+1-1, 2*nmax*nmax-1, ab));
        DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref bb, ldb, ref reset, zero);
        tmp_ab_ld = get_ldfrom2d(2*nmax*nmax, 0, nmax, tmp_ab);
        copy_ld(tmp_ab_ld, 0, 2 * nmax * nmax - (k * nmax + 1) + 1, ab, k * nmax + 1 - 1);
        //Console.WriteLine("aft 6c");
        //tmp_ab = get_2dfromld(2 * nmax * nmax - (k * nmax + 1) + 1, k * nmax + 1 - 1, nmax, ab);
        //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref bb, ldb, ref reset, zero);
        //tmp_ab_ld = get_ldfrom2d(2 * nmax * nmax - (k * nmax + 1) + 1, 0, nmax, tmp_ab);
        //copy_ld(tmp_ab_ld, 0, 2 * nmax * nmax - (k * nmax + 1) + 1, ab, k * nmax + 1 - 1);
    }
}

//100
for (int icu = 1; icu <= 2; icu = icu + 1) {
    uplo = ichu.Substring(icu-1,1).ToUpper();
    upper = (uplo == "U");

//90
for (int ia = 1; ia <= nalf; ia = ia + 1) {
    alpha = alf[ia-1];
    //80
    for (int ib = 1; ib <= nbet; ib = ib + 1) {

```

```

        beta = bet[ib-1];

        // Generate the matrix C
        //DMAKE( 'SY', UPLO, ' ', N, N, C, NMAX, CC, LDC, RESET, ZERO )
        DBLAT3_DMAKE("SY", uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);

        nc = nc + 1;

        // Save every datum before calling the subroutine
        uplos = uplo;
        transs = trans;
        ns = n;
        ks = k;
        als = alpha;
        for (int i = 1; i <= laa; i = i + 1) {
            _as[i-1] = aa[i-1];
        }
        ldas = lda;
        for (int i = 1; i <= lbb; i = i + 1) {
            bs[i-1] = bb[i-1];
        }
        ldbs = ldb;
        bets = beta;
        for (int i = 1; i <= lcc; i = i + 1) {
            cs[i-1] = cc[i-1];
        }
        ldc = ldc;

        // Call the subroutine
        if (trace) {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C,
(10,3:D)) .",
                nc, sname, uplo, trans, n, k, alpha, lda, ldb, beta, ldc);
        }
        if (rewi) {
            //REWIND NTRA
        }
        //DSYR2K( UPLO, TRANS, N, K, ALPHA, AA, LDA, BB, LDB, BETA, CC, LDC )
        tmp2d_cc = get_2dfromlid(nmax*nmax,0,ldc,cc);
        DSYR2K(uplo, trans, n, k, alpha, get_2dfromlid(nmax*nmax,0,lda,aa), lda, get_2dfromlid(nmax*nmax,0,ldb,bb), ldb, beta,
ref tmp2d_cc, ldc);

        //Console.WriteLine("bef 7c");
        cc = get_ldfromlid(nmax*nmax,0,ldc,tmp2d_cc);
        //Console.WriteLine("aft 7c");

        // Check if error-exit was taken incorrectly
        if (!ok) {
            nout.WriteLine( "***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");

            fatal = true;
            //GO TO 150
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C
(10,3:D)) .", nc, sname, uplo, trans, n, k, alpha, lda, ldb, beta, ldc);
            return;
        }

        // See what data changed inside subroutines
        isame[0] = (uplos == uplo);
        isame[1] = (transs == trans);
        isame[2] = (ns == n);
        isame[3] = (ks == k);
        isame[4] = (als == alpha);
        isame[5] = LDE(_as, aa, laa);
        isame[6] = (ldas == lda);
        isame[7] = LDE(bs, bb, lbb);
        isame[8] = (ldbs == ldb);
        isame[9] = (bets == beta);

        if (_null) {
            isame[10] = LDE(cs, cc, lcc);
        } else {
            isame[10] = LDERES("SY", uplo, n, n, get_2dfromlid(nmax*nmax,0,ldc,cs), get_2dfromlid(nmax*nmax,0,ldc,cc), ldc);
        }

        isame[11] = (ldcs == ldc);

        // If data was incorrectly changed, report and return
        same = true;
        for (int i = 1; i <= nargs; i = i + 1) {
            same = (same && isame[i-1]);
            if (!isame[i-1]) {
                nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****",i);
            }
        }
        if (!same) {
            fatal = true;
            //GO TO 150
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C
(10,3:D)) .", nc, sname, uplo, trans, n, k, alpha, lda, ldb, beta, ldc);
            return;
        }

        if (!_null) {
            // Check the result column by column
            jjab = 1;
            jc = 1;
            //70
            for (int j = 1; j <= n; j = j + 1) {
                if (upper) {
                    jj = 1;
                    lj = j;
                } else {
                    jj = j;
                    lj = n-j+1;
                }
                if (tran) {

```

```

        for (int i = 1; i <= k; i = i + 1) {
            w[i-1] = ab[(j-1)*2*nmax+k+i-1];
            w[k+i-1] = ab[(j-1)*2*nmax+i-1];
        }
//DMMCH( 'T', 'N', LJ, 1, 2*K, ALPHA, AB( JJAB ), 2*NMAX, W, 2*NMAX, BETA, C( JJ, J ), NMAX, CT, G, CC( JC
), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )

//Console.WriteLine("bef 7c");
DMMCH("T", "N", lj, 1, 2*k, alpha, get_2dfromld( 2*nmax*nmax, jjab-1, 2*nmax, ab), 2*nmax,
get_2dfromld(2*nmax, 0, 2*nmax, w), 2*nmax, beta,
get_2dfromld(nmax, 0, ldc, get_ldfrom2d_starti_full(nmax, jj-1, nmax, j-1, c)), nmax,
ref ct, ref g, get_2dfromld(nmax*nmax, 0, ldc, get_darr_full(nmax*nmax, jc-1, nmax * nmax - 1, cc)),
ldc, eps, ref err, ref fatal, nout, true);

//Console.WriteLine("aft 7c");
// static double[] get_ldfrom2d_starti(int l1, int start1, int l2, int start2, double[,] ar

    } else {
        for (int i = 1; i <= k; i = i + 1) {
            w[i-1] = ab[(k+i-1)*nmax+j-1];
            w[k+i-1] = ab[(i-1)*nmax+j-1];
        }
//DMMCH( 'N', 'N', LJ, 1, 2*K, ALPHA, AB( JJ ), NMAX, W, 2*NMAX, BETA, C( JJ, J ), NMAX, CT, G, CC( JC ),
), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )

//Console.WriteLine("bef 8c");
DMMCH("N", "N", lj, 1, 2*k, alpha, get_2dfromld(2*nmax*nmax, jj-1, nmax, ab), nmax,
get_2dfromld(2*nmax, 0, 2*nmax, w), 2*nmax, beta,
get_2dfromld(nmax, 0, ldc, get_ldfrom2d_starti_full(nmax, jj-1, nmax, j-1, c)), nmax,
ref ct, ref g, get_2dfromld(nmax*nmax, 0, ldc, get_darr_full(nmax*nmax, jc-1, nmax*nmax-1, cc)), ldc, eps, ref
err, ref fatal, nout, true);

//Console.WriteLine("aft 8c");
    }
    if (upper) {
        jc = jc + ldc;
    } else {
        jc = jc + ldc + 1;
        if (tran) {
            jjab = jjab + 2*nmax;
        }
    }
    errmax = Math.Max(errmax, err);

    // If got really bad answer, report and return
    if (fatal) {
        //GO TO 140
        if (n > 1) {
            nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
        }
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, B, {8,3:D}, {9,4:F1},
C {10,3:D})      .", nc, sname, uplo, trans, n, k, alpha, lda, ldb, beta, ldc);
        return;
    }
} //70
    } //80
    } //90
    } //100
    Loop110:;
    } //110
    } //120
    Loop130:;
    } //130

    // Report result
    if (errmax < thresh) {
        nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
    } else {
        nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
    }
}
static void DBLAT3_DCHKE(int isnum, string srnamt, StreamWriter nout)
{
    // Tests the error exits from the Level 3 Blas.
    // Requires a special version of the error-handling routine XERBLA.
    // A, B and C should not need to be defined.

    // Auxiliary routine for test program for Level 3 Blas.

    // Parameters
    const double one = 1.0;
    const double two = 2.0;

    // Local Scalars
    double alpha, beta;

    // Local Arrays
    double[,] a = new double[2,1];
    double[,] b = new double[2,1];
    double[,] c = new double[2,1];

    //COMMON          /INFOC/INFOT, NOUTC, OK, LERR

    // Executable Statements
    // OK is set to .FALSE. by the special version of XERBLA or by CHKXER if anything is wrong
    ok = true;
    // LERR is set to .TRUE. by the special version of XERBLA each time it is called, and is then tested and re-set by CHKXER.
    // LERR = .FALSE.
    lerr = false;

    // Initialize ALPHA and BETA
    alpha = one;
    beta = two;
    switch (isnum) {
        case 1:
            infot = 1;
            DGEMM("/", "N", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);

            CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 1;
            DGEMM("/", "T", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 2;

```



```

DTRMM("R","L","N","N",0,2,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRMM("R","L","T","N",0,2,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("L","U","N","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("L","U","T","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("R","U","N","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("R","U","T","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("L","L","N","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 11;
DTRMM("L","L","T","N",2,0,alpha,a,2,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
break;

case 4:
infot = 1;
DTRSM("/","U","N","N",0,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 2;
DTRSM("L","/", "N","N",0,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 3;
DTRSM("L","U","N","N",0,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 4;
DTRSM("L","U","N","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("L","U","T","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("L","U","N","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("L","L","N","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("L","L","T","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("R","L","N","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 5;
DTRSM("R","L","T","N",-1,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("L","U","N","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("L","U","T","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("R","U","N","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("R","U","T","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("L","L","N","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("L","L","T","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("R","L","N","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 6;
DTRSM("R","L","T","N",0,-1,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 7;
DTRSM("L","U","N","N",2,0,alpha,a,1,ref b,2);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("L","U","T","N",2,0,alpha,a,1,ref b,2);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("R","U","N","N",2,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("R","U","T","N",2,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("L","L","N","N",2,0,alpha,a,1,ref b,2);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("L","L","T","N",2,0,alpha,a,1,ref b,2);
CHKXER(srnamt,infot,nout,lerr,ok);
infot = 9;
DTRSM("R","L","N","N",2,0,alpha,a,1,ref b,1);
CHKXER(srnamt,infot,nout,lerr,ok);

```



```

        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 4;
        DSYR2K("L", "T", 0, -1, alpha, a, 1, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 7;
        DSYR2K("U", "N", 2, 0, alpha, a, 1, b, 1, beta, ref c, 2);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 7;
        DSYR2K("U", "T", 0, 2, alpha, a, 1, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 7;
        DSYR2K("L", "N", 2, 0, alpha, a, 1, b, 1, beta, ref c, 2);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 7;
        DSYR2K("L", "T", 0, 2, alpha, a, 1, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        DSYR2K("U", "N", 2, 0, alpha, a, 2, b, 1, beta, ref c, 2);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        DSYR2K("U", "T", 0, 2, alpha, a, 2, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        DSYR2K("L", "N", 2, 0, alpha, a, 2, b, 1, beta, ref c, 2);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        DSYR2K("L", "T", 0, 2, alpha, a, 2, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        DSYR2K("U", "N", 2, 0, alpha, a, 2, b, 2, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        DSYR2K("U", "T", 2, 0, alpha, a, 1, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        DSYR2K("L", "N", 2, 0, alpha, a, 2, b, 2, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        DSYR2K("L", "T", 2, 0, alpha, a, 1, b, 1, beta, ref c, 1);
        CHKXER(srnamt, infot, nout, lerr, ok);
        break;

    default:
        break;
}
if (ok) {
    nout.WriteLine(" {0,6} PASSED THE TESTS OF ERROR-EXITS",srnamt);
} else {
    nout.WriteLine("***** {0,6} FAILED THE TESTS OF ERROR-EXITS *****",srnamt);
}
}
static void DBLAT3_DMAKE(string type, string uplo, string diag, int m, int n, ref double[,] a, int nmax, ref double[] aa, int lda, ref bool reset,
double transl)
{
    //DMAKE( TYPE, UPLO, DIAG, M, N, A, NMAX, AA, LDA, RESET,TRANSL )

    // Generates values for an M by N matrix A.  Stores the values in the array AA in the data structure required
    // by the routine, with unwanted elements set to rogue value.
    // TYPE is 'GE', 'SY' or 'TR'.
    // Auxiliary routine for test program for Level 3 Blas.

    // Parameters
    const double zero = 0.0;
    const double one = 1.0;
    const double rogue = -1.0e10;

    // DOUBLE PRECISION  A( NMAX, * ), AA( * )

    // Local Scalars
    //int i, ibeg, iend, j;
    int ibeg, iend;
    bool gen, lower, sym, tri, unit, upper;

    // Executable Statements
    gen = (type.Substring(0,2).ToUpper() == "GE");
    sym = (type.Substring(0,2).ToUpper() == "SY");
    tri = (type.Substring(0,2).ToUpper() == "TR");
    upper = ((sym || tri) && (uplo.Substring(0,1).ToUpper() == "U"));
    lower = ((sym || tri) && (uplo.Substring(0,1).ToUpper() == "L"));
    unit = tri && (diag.Substring(0,1).ToUpper() == "U");

    // Generate data in array A

    //20
    for (int j = 1; j <= n; j = j + 1){
        //10
        for (int i = 1; i <= m; i = i + 1) {
            if (gen || (upper && (i <= j)) || (lower && (i >= j))) {
                a[i-1,j-1] = DBEG(ref reset)+transl;
                if (i != j) {
                    // Set some elements to zero
                    if ((n > 3) && (j == n/2)) {
                        a[i-1,j-1] = zero;
                    }
                    if (sym) {
                        a[j-1, i-1] = a[i-1, j-1];
                    } else if (tri) {
                        a[j-1,i-1] = zero;
                    }
                }
            }
        }
        //10
        if (tri) {
            a[j-1,j-1] = a[j-1,j-1] + one;
        }
        if (unit) {
            a[j-1,j-1] = one;
        }
    }
    //20

    // Store elements in array AS in data structure required by routine
    if (type.Substring(0,2).ToUpper() == "GE") {
        //50
        //Console.WriteLine("3_DMAKE GE");
    }
}

```

```

        for (int j = 1; j <= n; j = j + 1) {
            //30
            for (int i = 1; i <= m; i = i + 1) {
                aa[(i+(j-1)*lda) -1] = a[i-1,j-1];
            }
            for (int i = m+1; i <= lda; i = i + 1) {
                aa[(i+(j-1)*lda) -1] = rogue;
            }
        } //50
    } else if ((type.Substring(0,2).ToUpper() == "SY") || (type.Substring(0,2).ToUpper() == "TR")) {
        //90
        for (int j = 1; j <= n; j = j + 1) {
            if (upper) {
                ibeg = 1;
                if (unit) {
                    iend = j - 1;
                } else {
                    iend = j;
                }
            } else {
                if (unit) {
                    ibeg = j + 1;
                } else {
                    ibeg = j;
                }
                iend = n;
            }
            for (int i = 1; i <= ibeg -1; i = i + 1) {
                aa[(i+(j-1)*lda) -1] = rogue;
            }
            for (int i = ibeg; i <= iend; i = i + 1) {
                aa[(i+(j-1)*lda) -1] = a[i-1,j-1];
            }
            for (int i = iend+1; i <= lda; i = i + 1) {
                aa[(i+(j-1)*lda) -1] = rogue;
            }
        } //90
    }
}
//ref ?
static void DMMCH(string transa, string transb, int m, int n, int kk, double alpha, double[,] a, int lda, double[,] b, int ldb, double beta,
double[,] c, int ldc,
    ref double[] ct, ref double[] g, double[,] cc, int ldcc, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv)
{
    // DMMCH( TRANSA, TRANSB, M, N, KK, ALPHA, A, LDA, B, LDB, BETA, C, LDC, CT, G, CC, LDCC, EPS, ERR, FATAL, NOUT, MV )
    // Checks the results of the computational tests
    // Auxiliary routine for test program for Level 3 Blas

    // Parameters
    double zero = 0.0;
    double one = 1.0;

    // Local Scalars
    double erri;
    bool trana, tranb;

    // DOUBLE PRECISION  A( LDA, * ), B( LDB, * ), C( LDC, * ), CC( LDCC, * ), CT( * ), G( * )

    // Executable Statements
    trana = (transa.Substring(0,1).ToUpper() == "T") || (transa.Substring(0,1).ToUpper() == "C");
    tranb = (transb.Substring(0,1).ToUpper() == "T") || (transb.Substring(0,1).ToUpper() == "C");

    // Compute expected result, one column at a time, in CT using data in A, B and C.  Compute gauges in G.
    //120
    for (int j = 1; j <= n; j = j + 1) {
        for (int i = 1; i <= m; i = i + 1) {
            ct[i-1] = zero;
            g[i-1] = zero;
        }
        if (!trana && !tranb) {
            for (int k = 1; k <= kk; k = k + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    ct[i-1] = ct[i-1] + a[i-1,k-1]*b[k-1,j-1];
                    g[i-1] = g[i-1] + Math.Abs(a[i-1,k-1])*Math.Abs(b[k-1,j-1]);
                }
            }
        } else if (trana && !tranb) {
            for (int k = 1; k <= kk; k = k + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    ct[i-1] = ct[i-1] + a[k-1,i-1]*b[k-1,j-1];
                    g[i-1] = g[i-1] + Math.Abs(a[k-1,i-1])*Math.Abs(b[k-1,j-1]);
                }
            }
        } else if (!trana && tranb) {
            for (int k = 1; k <= kk; k = k + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    ct[i-1] = ct[i-1] + a[i-1,k-1]*b[j-1,k-1];
                    g[i-1] = g[i-1] + Math.Abs(a[i-1,k-1])*Math.Abs(b[j-1,k-1]);
                }
            }
        } else if (trana && tranb) {
            for (int k = 1; k <= kk; k = k + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    ct[i-1] = ct[i-1] + a[k-1,i-1]*b[j-1,k-1];
                    g[i-1] = g[i-1] + Math.Abs(a[k-1,i-1])*Math.Abs(b[j-1,k-1]);
                }
            }
        }
        for (int i = 1; i <= m; i = i + 1) {
            ct[i-1] = alpha*ct[i-1]+beta*c[i-1,j-1];
            g[i-1] = Math.Abs(alpha)*g[i-1]+Math.Abs(beta)*Math.Abs(c[i-1,j-1]);
        }

        // Compute the error ratio for this result
        err = zero;
        for (int i = 1; i <= m; i = i + 1) {
            erri = Math.Abs(ct[i-1] - cc[i-1,j-1])/eps;
            if (g[i-1] != zero) {
                erri = erri/g[i-1];
            }
            err = Math.Max(err,erri);
            if (err*Math.Sqrt(eps) >= one) {
                //GO TO 130
            }
        }
    }
}

```

```

// Report fatal error
fatal = true;
nout.WriteLine("***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HALF ACCURATE *****\n
RESULT");
EXPECTED RESULT    COMPUTED

for (int ii = 1; ii <= m; ii = ii + 1) {
    if (mv) {
        nout.WriteLine(" {0,7:D} {1,18:F6} {2,18:F6}", ii, ct[ii-1], cc[ii-1,j-1]);
    } else {
        nout.WriteLine(" {0,7:D} {1,18:F6} {2,18:F6}",ii, cc[ii-1,j-1], ct[ii-1]);
    }
}
if (n > 1) {
    nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}",j);
}
return;
}
} //110
} //120

// If the loop completes, all results are at least half accurate
//GO TO 150
}
public static void DGEMM(string transa, string transb, int m, int n, int k, double alpha, double[,] a, int lda, double[,] b, int ldb, double beta,
ref double[,] c, int ldc)
{
    //DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
    //DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)

    // DGEMM performs one of the matrix-matrix operations
    //
    // C := alpha*op( A )*op( B ) + beta*C,
    //
    // where op( X ) is one of
    //
    // op( X ) = X   or   op( X ) = X**T,
    //
    // alpha and beta are scalars, and A, B and C are matrices, with op( A ) an m by k matrix, op( B ) a k by n matrix and C an m by n matrix.

    // On entry, TRANSA specifies the form of op( A ) to be used in the matrix multiplication as follows:
    // TRANSA = 'N' or 'n', op( A ) = A.
    // TRANSA = 'T' or 't', op( A ) = A**T.
    // TRANSA = 'C' or 'c', op( A ) = A**T.

    // On entry, TRANSB specifies the form of op( B ) to be used in the matrix multiplication as follows:
    // TRANSB = 'N' or 'n', op( B ) = B.
    // TRANSB = 'T' or 't', op( B ) = B**T.
    // TRANSB = 'C' or 'c', op( B ) = B**T.

    // On entry, M specifies the number of rows of the matrix op( A ) and of the matrix C. M must be at least zero.

    // On entry, N specifies the number of columns of the matrix op( B ) and the number of columns of the matrix C. N must be at least zero.

    // On entry, K specifies the number of columns of the matrix op( A ) and the number of rows of the matrix op( B ). K must be at least zero.

    // On entry, ALPHA specifies the scalar alpha.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with
    TRANSA = 'N' or 'n', the leading m by k
    // part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at
    least max( 1, m ), otherwise LDA must be at
    // least max( 1, k ).

    // B is DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with
    TRANSB = 'N' or 'n', the leading k by n
    // part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at
    least max( 1, k ), otherwise LDB must be at
    // least max( 1, n ).

    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

    // C is DOUBLE PRECISION array of DIMENSION ( LDC, n ). Before entry, the leading m by n part of the array C must contain the matrix C,
    except when beta is zero, in which
    // case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix ( alpha*op( A )*op( B ) + beta*C ).

    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, m ).

    // Level 3 Blas routine.
    // DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)

    // Local Scalars
    double temp;
    int info, ncola, nrowa, nrowb;
    bool nota, notb;

    // Parameters
    double one = 1.0;
    double zero = 0.0;

    // Set NOTA and NOTB as true if A and B respectively are not transposed and set NROWA, NCOLA and NROWB as the number of rows and
    columns of A and the number of rows of B respectively.
    nota = (transa.Substring(0,1).ToUpper() == "N");
    notb = (transb.Substring(0,1).ToUpper() == "N");
    if (nota) {
        nrowa = m;
        ncola = k;
    } else {
        nrowa = k;
        ncola = m;
    }
    if (notb) {
        nrowb = k;
    } else {
        nrowb = n;
    }

    // Test the input parameters
    info = 0;
    if ( !nota && !(transa.Substring(0,1).ToUpper() == "C") && !(transa.Substring(0,1).ToUpper() == "T") ) {
        info = 1;
    } else if ( !notb && !(transb.Substring(0,1).ToUpper() == "C") && !(transb.Substring(0,1).ToUpper() == "T") ) {

```

```

        info = 2;
    } else if (m < 0) {
        info = 3;
    } else if (n < 0) {
        info = 4;
    } else if (k < 0) {
        info = 5;
    } else if (lda < Math.Max(1,nrowa)) {
        info = 8;
    } else if (ldb < Math.Max(1,nrowb)) {
        info = 10;
    } else if (ldc < Math.Max(1,m)) {
        info = 13;
    }
    if (info != 0) {
        XERBLA("DGEMM ", info);
        return;
    }

    // Quick return if possible
    if ((m == 0) || (n == 0) || ((alpha == zero) || (k == 0)) && (beta == one)) {
        return;
    }

    // And if alpha.eq.zero.
    if (alpha == zero) {
        if (beta == zero) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
        }
        return;
    }

    // Start the operations.
    if (notb) {
        if (nota) {
            // Form C := alpha*A*B + beta*C
            for (int j = 1; j <= n; j = j + 1) {
                if (beta == zero) {
                    for (int i = 1; i <= m; i = i + 1) {
                        c[i-1,j-1] = zero;
                    }
                } else if (beta != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        c[i-1,j-1] = beta*c[i-1,j-1];
                    }
                }
                for (int l = 1; l <= k; l = l + 1) {
                    if (b[l-1,j-1] != zero) {
                        temp = alpha*b[l-1,j-1];
                        for (int i = 1; i <= m; i = i + 1) {
                            c[i-1,j-1] = c[i-1,j-1] + temp*a[i-1,l-1];
                        }
                    }
                }
            }
        } else {
            // Form C := alpha*A**T*B + beta*C
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= m; i = i + 1) {
                    temp = zero;
                    for (int l = 1; l <= k; l = l + 1) {
                        temp = temp + a[l-1,i-1]*b[l-1,j-1];
                    }
                    if (beta == zero) {
                        c[i-1,j-1] = alpha*temp;
                    } else {
                        c[i-1,j-1] = alpha*temp+beta*c[i-1,j-1];
                    }
                }
            }
        }
    } else {
        if (nota) {
            // Form C := alpha*A*B**T + beta*C
            //l70
            for (int j = 1; j <= n; j = j + 1) {
                if (beta == zero) {
                    for (int i = 1; i <= m; i = i + 1) {
                        c[i-1,j-1] = zero;
                    }
                } else if (beta != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        c[i-1,j-1] = beta*c[i-1,j-1];
                    }
                }
                for (int l = 1; l <= k; l = l + 1) {
                    if (b[j-1,l-1] != zero) {
                        temp = alpha*b[j-1,l-1];
                        for (int i = 1; i <= m; i = i + 1) {
                            c[i-1,j-1] = c[i-1,j-1] + temp*a[i-1,l-1];
                        }
                    }
                }
            }
        } //l70
    } else {
        // Form C := alpha*A**T*B**T + beta*C
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1) {
                    temp = temp + a[l-1,i-1]*b[j-1,l-1];
                }
                if (beta == zero) {
                    c[i-1,j-1] = alpha*temp;
                }
            }
        }
    }
}

```

```

        } else {
            c[i-1,j-1] = alpha*temp + beta*c[i-1,j-1];
        }
    }
}
}

public static void DSymm(string side, string uplo, int m, int n, double alpha, double[,] a, int lda, double[,] b, int ldb, double beta, ref
double[,] c, int ldc)
{
    //DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
    //DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)

    // DSYMM performs one of the matrix-matrix operations
    //   C := alpha*A*B + beta*C,
    //   or
    //   C := alpha*B*A + beta*C,
    // where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

    // On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:
    //   SIDE = 'L' or 'l' C := alpha*A*B + beta*C,
    //   SIDE = 'R' or 'r' C := alpha*B*A + beta*C,
    //
    // On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as
follows:
    //   UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.
    //   UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.
    //
    // On entry, M specifies the number of rows of the matrix C. M must be at least zero.

    // On entry, N specifies the number of columns of the matrix C. N must be at least zero.

    // On entry, ALPHA specifies the scalar alpha.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE
= 'L' or 'l', the m by m part of
    // the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array
A must contain the upper triangular part
    // of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by
m lower triangular part of the array A
    // must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.
    // Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U'
or 'u', the leading n by n upper triangular
    // part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not
referenced, and when UPLO = 'L' or 'l',
    // the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the
strictly upper triangular part of A is not
    // referenced.

    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at
least max( 1, m ), otherwise LDA must be at
    // least max( 1, n ).

    // B is DOUBLE PRECISION array of DIMENSION ( LDB, n ).
    // Before entry, the leading m by n part of the array B must contain the matrix B.

    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max( 1, m ).

    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

    // C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
    // Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be
set on entry.
    // On exit, the array C is overwritten by the m by n updated matrix.

    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, m ).

    // Reference BLAS level3 routine (version 3.4.0) --

    // Local Scalars
    double temp1, temp2;
    int info, nrowa;
    bool upper;

    // Parameters
    const double one = 1.0;
    const double zero = 0.0;

    // Set NROWA as the number of rows of A
    if (side.Substring(0,1).ToUpper() == "L") {
        nrowa = m;
    } else {
        nrowa = n;
    }
    upper = (uplo.Substring(0,1).ToUpper() == "U");

    // Test the input parameters
    info = 0;
    if (!side.Substring(0,1).ToUpper() == "L") && !(side.Substring(0,1).ToUpper() == "R") {
        info = 1;
    } else if (!upper && !(uplo.Substring(0,1).ToUpper() == "L")) {
        info = 2;
    } else if (m < 0) {
        info = 3;
    } else if (n < 0) {
        info = 4;
    } else if (lda < Math.Max(1,nrowa)) {
        info = 7;
    } else if (ldb < Math.Max(1,m)) {
        info = 9;
    } else if (ldc < Math.Max(1,m)) {
        info = 12;
    }
    if (info != 0) {
        XERBLA("DSYMM ",info);
        return;
    }

    // Quick return if possible
    if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one))) {
        return;
    }
}

```

```

// And when alpha.eq.zero
if (alpha == zero) {
    if (beta == zero) {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = zero;
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = beta*c[i-1,j-1];
            }
        }
    }
    return;
}

// Start the operations
if (side.Substring(0,1).ToUpper() == "L") {
    // Form C := alpha*A*B + beta*C
    if (upper) {
        //70
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                temp1 = alpha*b[i-1,j-1];
                temp2 = zero;
                for (int k = 1; k <= i - 1; k = k + 1) {
                    {
                        c[k - 1, j - 1] = c[k - 1, j - 1] + temp1 * a[k - 1, i - 1];
                        temp2 = temp2 + b[k - 1, j - 1] * a[k - 1, i - 1];
                    }
                }
                if (beta == zero) {
                    c[i-1,j-1] = temp1*a[i-1,i-1] + alpha*temp2;
                } else {
                    c[i-1,j-1] = beta*c[i-1,j-1] + temp1*a[i-1,i-1] + alpha*temp2;
                }
            }
        } //70
    } else {
        //100
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = m; i >= 1; i = i - 1) { //DO 90 I = M,1,-1
                temp1 = alpha*b[i-1,j-1];
                temp2 = zero;
                for (int k = i+1; k <= m; k = k + 1) { // DO 80 K = I + 1,M
                    c[k-1,j-1] = c[k-1,j-1] + temp1*a[k-1,i-1];
                    temp2 = temp2 + b[k-1,j-1]*a[k-1,i-1];
                }
                if (beta == zero) {
                    c[i-1,j-1] = temp1*a[i-1,i-1] + alpha*temp2;
                } else {
                    c[i-1,j-1] = beta*c[i-1,j-1] + temp1*a[i-1,i-1] + alpha*temp2;
                }
            }
        }
    }
} else {
    // Form C := alpha*B*A + beta*C
    //170
    for (int j = 1; j <= n; j = j + 1) {
        temp1 = alpha*a[j-1,j-1];
        if (beta == zero) {
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = temp1*b[i-1,j-1];
            }
        } else {
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = beta*c[i-1,j-1] + temp1*b[i-1,j-1];
            }
        }
        for (int k = 1; k <= j-1; k = k + 1) {
            if (upper) {
                temp1 = alpha*a[k-1,j-1];
            } else {
                temp1 = alpha*a[j-1,k-1];
            }
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = c[i-1,j-1] + temp1*b[i-1,k-1];
            }
        }
        for (int k = j+1; k <= n; k = k + 1) {
            if (upper) {
                temp1 = alpha*a[j-1,k-1];
            } else {
                temp1 = alpha*a[k-1,j-1];
            }
            for (int i = 1; i <= m; i = i + 1) {
                c[i-1,j-1] = c[i-1,j-1] + temp1*b[i-1,k-1];
            }
        }
    }
}

}

public static void DTRMM(string side, string uplo, string transa, string diag, int m, int n, double alpha, double[,] a, int lda, ref double[,] b,
int ldb)
{
    // DOUBLE PRECISION A(LDA,*),B(LDB,*)

    // DTRMM performs one of the matrix-matrix operations

    // B := alpha*op( A )*B, or B := alpha*B*op( A ),

    // where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op( A ) is one of

    // op( A ) = A or op( A ) = A**T.

    // SIDE is CHARACTER*1
    // On entry, SIDE specifies whether op( A ) multiplies B from the left or right as follows:
    // SIDE = 'L' or 'l' B := alpha*op( A )*B.
    // SIDE = 'R' or 'r' B := alpha*B*op( A ).

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

```

```

//          UPLO = 'U' or 'u'   A is an upper triangular matrix.
//          UPLO = 'L' or 'l'   A is a lower triangular matrix.

// TRANSA is CHARACTER*1
// On entry, TRANSA specifies the form of op( A ) to be used in the matrix multiplication as follows:
//          TRANSA = 'N' or 'n'   op( A ) = A.
//          TRANSA = 'T' or 't'   op( A ) = A**T.
//          TRANSA = 'C' or 'c'   op( A ) = A**T.

// DIAG is CHARACTER*1
// On entry, DIAG specifies whether or not A is unit triangular as follows:
//          DIAG = 'U' or 'u'   A is assumed to be unit triangular.
//          DIAG = 'N' or 'n'   A is not assumed to be unit triangular.

// M is INTEGER
// On entry, M specifies the number of rows of B. M must be at least zero.

// N is INTEGER
// On entry, N specifies the number of columns of B. N must be at least zero.

// ALPHA is DOUBLE PRECISION.
// On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

// A is DOUBLE PRECISION array of DIMENSION ( LDA, k ), where k is m when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry
with UPLO = 'U' or 'u', the leading k by k
// upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not
referenced.
// Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix
and the strictly upper triangular part of
// A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be
unity.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at
least max( 1, m ), when SIDE = 'R' or 'r'
// then LDA must be at least max( 1, n ).

// B is DOUBLE PRECISION array of DIMENSION ( LDB, n ). Before entry, the leading m by n part of the array B must contain the matrix B,
and on exit is overwritten by the
// transformed matrix.

// LDB is INTEGER
// On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max( 1, m ).

// Parameters
const double zero = 0.0;
const double one = 1.0;

// Local Scalars
double temp;
int info, nrowa; //i,j,k
bool lside,nounit,upper;

//kx = 0; // added

// Test the input parameters
lside = (side.Substring(0,1).ToUpper() == "L");
if (lside) {
    nrowa = m;
} else {
    nrowa = n;
}
nounit = (diag.Substring(0,1).ToUpper() == "N");
upper = (uplo.Substring(0,1).ToUpper() == "U");

//
info = 0;
if (!lside && !(side.Substring(0,1).ToUpper() == "R")) {
    info = 1;
} else if (!upper && !(uplo.Substring(0,1).ToUpper() == "L")) {
    info = 2;
} else if (!(transa.Substring(0,1).ToUpper() == "N") && !(transa.Substring(0,1).ToUpper() == "T") && !(transa.Substring(0,1).ToUpper() == "C"))
{
    info = 3;
} else if (!(diag.Substring(0,1).ToUpper() == "U") && !(diag.Substring(0,1).ToUpper() == "N")) {
    info = 4;
} else if (m < 0) {
    info = 5;
} else if (n < 0) {
    info = 6;
} else if (lda < Math.Max(1,nrowa)) {
    info = 9;
} else if (ldb < Math.Max(1,m)) {
    info = 11;
}

if (info != 0) {
    XERBLA("DTRMM ",info);
    return;
}

// Quick return if possible
if (m == 0 || n == 0) {
    return;
}

// And when alpha.eq.zero.
if (alpha == zero) {
    for (int j = 1; j <= n; j = j + 1) {
        for (int i = 1; i <= m; i = i + 1) {
            b[i-1,j-1] = zero;
        }
    }
    return;
}

// Start the operations.
if (lside) {
    if (transa.Substring(0,1).ToUpper() == "N") {
        // Form B := alpha*A*B.
        if (upper) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int k = 1; k <= m; k = k + 1) {

```

```

        if (b[k-1,j-1] != zero) {
            temp = alpha*b[k-1,j-1];
            for (int i = 1; i <= k-1; i = i + 1) {
                b[i-1,j-1] = b[i-1,j-1] + temp*a[i-1,k-1];
            }
            if (nunit) {
                temp = temp*a[k-1,k-1];
            }
            b[k-1,j-1] = temp;
        }
    }
} else {
    for (int j = 1; j <= n; j = j + 1) {
        for (int k = m; k >= 1; k = k - 1) {
            if (b[k-1,j-1] != zero) {
                temp = alpha*b[k-1,j-1];
                b[k-1,j-1] = temp;
                if (nunit) {
                    b[k-1,j-1] = b[k-1,j-1]*a[k-1,k-1];
                }
                for (int i = k+1; i <= m; i = i + 1) {
                    b[i-1,j-1] = b[i-1,j-1] + temp*a[i-1,k-1];
                }
            }
        }
    }
}
} else {
    // Form B := alpha*A**T*B.
    if (upper) {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = m; i >= 1; i = i - 1) {
                temp = b[i-1,j-1];
                if (nunit) {
                    temp = temp*a[i-1,i-1];
                }
                for (int k = 1; k <= i-1; k = k + 1) {
                    temp = temp + a[k-1,i-1]*b[k-1,j-1];
                }
                b[i-1,j-1] = alpha*temp;
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                temp = b[i-1,j-1];
                if (nunit) {
                    temp = temp * a[i-1,i-1];
                }
                for (int k = i+1; k <= m; k = k + 1) {
                    temp = temp + a[k-1,i-1]*b[k-1,j-1];
                }
                b[i-1,j-1] = alpha*temp;
            }
        }
    }
}
} else {
    if (transa.Substring(0,1).ToUpper() == "N") {
        // Form B := alpha*B*A.
        if (upper) {
            for (int j = n; j >= 1; j = j - 1) {
                temp = alpha;
                if (nunit) {
                    temp = temp*a[j-1,j-1];
                }
                for (int i = 1; i <= m; i = i + 1) {
                    b[i-1,j-1] = temp*b[i-1,j-1];
                }
                for (int k = 1; k <= j-1; k = k + 1) {
                    if (a[k-1,j-1] != zero) {
                        temp = alpha*a[k-1,j-1];
                        for (int i = 1; i <= m; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] + temp*b[i-1,k-1];
                        }
                    }
                }
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            temp = alpha;
            if (nunit) {
                temp = temp*a[j-1,j-1];
            }
            for (int i = 1; i <= m; i = i + 1) {
                b[i-1,j-1] = temp*b[i-1,j-1];
            }
            for (int k = j+1; k <= n; k = k + 1) {
                if (a[k-1,j-1] != zero) {
                    temp = alpha*a[k-1,j-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = b[i-1,j-1] + temp*b[i-1,k-1];
                    }
                }
            }
        }
    }
}
} else {
    // Form B := alpha*B*A**T.
    if (upper) {
        for (int k = 1; k <= n; k = k + 1) {
            for (int j = 1; j <= k-1; j = j + 1) {
                if (a[j-1,k-1] != zero) {
                    temp = alpha*a[j-1,k-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = b[i-1,j-1] + temp*b[i-1,k-1];
                    }
                }
            }
        }
        temp = alpha;
        if (nunit) {
            temp = temp*a[k-1,k-1];
        }
    }
}

```



```

        nrowa = n;
    }
    nounit = (diag.Substring(0,1).ToUpper() == "N");
    upper = (uplo.Substring(0,1).ToUpper() == "U");

    //
    info = 0;
    if (!lside && !(side.Substring(0,1).ToUpper() == "R")) {
        info = 1;
    } else if (!upper && !(uplo.Substring(0,1).ToUpper() == "L")) {
        info = 2;
    } else if (!(transa.Substring(0,1).ToUpper() == "N") && !(transa.Substring(0,1).ToUpper() == "T") && !(transa.Substring(0,1).ToUpper() == "C"))
{
        info = 3;
    } else if (!(diag.Substring(0,1).ToUpper() == "U") && !(diag.Substring(0,1).ToUpper() == "N")) {
        info = 4;
    } else if (m < 0) {
        info = 5;
    } else if (n < 0) {
        info = 6;
    } else if (lda < Math.Max(1,nrowa)) {
        info = 9;
    } else if (ldb < Math.Max(1,m)) {
        info = 11;
    }
}

if (info != 0) {
    XERBLA("DTRSM ",info);
    return;
}

// Quick return if possible
if (m == 0 || n == 0) {
    return;
}

// And when alpha.eq.zero.
if (alpha == zero) {
    for (int j = 1; j <= n; j = j + 1) {
        for (int i = 1; i <= m; i = i + 1) {
            b[i-1,j-1] = zero;
        }
    }
    return;
}

// Start the operations.
if (lside) {
    if (transa.Substring(0,1).ToUpper() == "N") {
        // Form B := alpha*inv( A )*B.
        if (upper) {
            //m=1, n=1, side=L, uplo=U, transa=N, diag=N, alpha=1
            //Console.WriteLine("in m=1, n=1, side=L, uplo=U, transa=N, diag=N, alpha=1");
            for (int j = 1; j <= n; j = j + 1) {
                if (alpha != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = alpha*b[i-1,j-1];
                    }
                }
                for (int k = m; k >= 1; k = k - 1) {
                    if (b[k-1,j-1] != zero) {
                        if (nounit) {
                            b[k-1,j-1] = b[k-1,j-1]/a[k-1,k-1];
                        }
                        for (int i = 1; i <= k-1; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] - b[k-1,j-1]*a[i-1,k-1];
                        }
                    }
                }
            }
        }
        // Console.WriteLine("out m=1, n=1, side=L, uplo=U, transa=N, diag=N, alpha=1");
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            if (alpha != one) {
                for (int i = 1; i <= m; i = i + 1) {
                    b[i-1,j-1] = alpha*b[i-1,j-1];
                }
            }
            for (int k = 1; k <= m; k = k + 1) {
                if (b[k-1,j-1] != zero) {
                    if (nounit) {
                        b[k-1,j-1] = b[k-1,j-1]/a[k-1,k-1];
                    }
                    for (int i = k+1; i <= m; i = i + 1) {
                        b[i-1,j-1] = b[i-1,j-1] - b[k-1,j-1]*a[i-1,k-1];
                    }
                }
            }
        }
    }
} else {
    // Form B := alpha*inv( A**T )*B.
    if (upper) {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= m; i = i + 1) {
                temp = alpha*b[i-1,j-1];
                for (int k = 1; k <= i-1; k = k + 1) {
                    temp = temp - a[k-1,i-1]*b[k-1,j-1];
                }
                if (nounit) {
                    temp = temp/a[i-1,i-1];
                }
                b[i-1,j-1] = temp;
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = m; i >= 1; i = i - 1) {
                temp = alpha*b[i-1,j-1];
            }
        }
    }
}

```

```

        for (int k = i+1; k <= m; k = k + 1) {
            temp = temp - a[k-1,i-1]*b[k-1,j-1];
        }
        if (nunit) {
            temp = temp / a[i-1,i-1];
        }
        b[i-1,j-1] = temp;
    }
}
} else {
    if (transa.Substring(0,1).ToUpper() == "N") {
        // Form B := alpha*B*inv( A ).
        if (upper) {
            for (int j = 1; j <= n; j = j + 1) {
                if (alpha != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = alpha*b[i-1,j-1];
                    }
                }
                for (int k = 1; k <= j-1; k = k + 1) {
                    if (a[k-1,j-1] != zero) {
                        for (int i = 1; i <= m; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] - a[k-1,j-1]*b[i-1,k-1];
                        }
                    }
                }

                if (nunit) {
                    temp = one/a[j-1,j-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = temp*b[i-1,j-1];
                    }
                }
            }
        } else {
            for (int j = n; j >= 1; j = j - 1) {
                if (alpha != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = alpha*b[i-1,j-1];
                    }
                }
                for (int k = j+1; k <= n; k = k + 1) {
                    if (a[k-1,j-1] != zero) {
                        for (int i = 1; i <= m; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] - a[k-1,j-1]*b[i-1,k-1];
                        }
                    }
                }
                if (nunit) {
                    temp = one/a[j-1,j-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,j-1] = temp*b[i-1,j-1];
                    }
                }
            }
        }
    } else {
        // Form B := alpha*B*inv( A**T ).
        if (upper) {
            for (int k = n; k >= 1; k = k - 1) {
                if (nunit) {
                    temp = one/a[k-1,k-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,k-1] = temp*b[i-1,k-1];
                    }
                }
                for (int j = 1; j <= k-1; j = j + 1) {
                    if (a[j-1,k-1] != zero) {
                        temp = a[j-1,k-1];
                        for (int i = 1; i <= m; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] - temp*b[i-1,k-1];
                        }
                    }
                }
                if (alpha != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,k-1] = alpha*b[i-1,k-1];
                    }
                }
            }
        } else {
            for (int k = 1; k <= n; k = k + 1) {
                if (nunit) {
                    temp = one/a[k-1,k-1];
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,k-1] = temp*b[i-1,k-1];
                    }
                }
                for (int j = k+1; j <= n; j = j + 1) {
                    if (a[j-1,k-1] != zero) {
                        temp = a[j-1,k-1];
                        for (int i = 1; i <= m; i = i + 1) {
                            b[i-1,j-1] = b[i-1,j-1] - temp*b[i-1,k-1];
                        }
                    }
                }

                if (alpha != one) {
                    for (int i = 1; i <= m; i = i + 1) {
                        b[i-1,k-1] = alpha*b[i-1,k-1];
                    }
                }
            }
        }
    }
}

```

```

    }
    }
}

}

public static void DSYRK(string uplo, string trans, int n, int k, double alpha, double[,] a, int lda, double beta, ref double[,] c, int ldc)
{
    //Console.WriteLine("uplo={0}, trans={1}, n={2}, k={3}, alpha={4}, lda={5}, beta={6}, ldc={7}", uplo, trans, n, k, alpha, lda, beta, ldc);
    //DSYRK(UPLO,TRANS,N,K,ALPHA,A,LDL,BETA,C,LDC)
    //DOUBLE PRECISION A(LDA,*),C(LDC,*)

    // DSYRK performs one of the symmetric rank k operations
    // C := alpha*A*A**T + beta*C,
    // C := alpha*A**T*A + beta*C,
    // where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix
    in the second case.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' C := alpha*A*A**T + beta*C.
    // TRANS = 'T' or 't' C := alpha*A**T*A + beta*C.
    // TRANS = 'C' or 'c' C := alpha*A**T*A + beta*C.

    // N is INTEGER
    // On entry, N specifies the order of the matrix C. N must be at least zero.

    // K is INTEGER
    // On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't'
    or 'C' or 'c', K specifies the number
    // of rows of the matrix A. K must be at least zero.

    // ALPHA is DOUBLE PRECISION.
    // On entry, ALPHA specifies the scalar alpha.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with
    TRANS = 'N' or 'n', the leading n by k
    // part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
    at least max( 1, n ), otherwise LDA must
    // be at least max( 1, k ).

    // BETA is DOUBLE PRECISION.
    // On entry, BETA specifies the scalar beta.

    // C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
    the symmetric matrix and the strictly
    // lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
    part of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
    the symmetric matrix and the strictly
    // upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
    part of the updated matrix.

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

    // Local Scalars
    double temp;
    int info, nrowa;
    bool upper;

    // Parameters
    const double one = 1.0;
    const double zero = 0.0;

    // Test the input parameters
    if (trans.Substring(0,1).ToUpper() == "N") {
        nrowa = n;
    } else {
        nrowa = k;
    }
    upper = (uplo.Substring(0,1).ToUpper() == "U");
    info = 0;

    if (!upper && !(uplo.Substring(0,1).ToUpper() == "L")) {
        info = 1;
    } else if (!(trans.Substring(0,1).ToUpper() == "N") && !(trans.Substring(0,1).ToUpper() == "T") && !(trans.Substring(0,1).ToUpper() == "C")) {
        info = 2;
    } else if (n < 0) {
        info = 3;
    } else if (k < 0) {
        info = 4;
    } else if (lda < Math.Max(1,nrowa)) {
        info = 7;
    } else if (ldc < Math.Max(1,n)) {
        info = 10;
    }
    if (info != 0) {
        XERBLA("DSYRK ",info);
        return;
    }

    // Quick return if possible.
    if ((n == 0) || (((alpha == zero) || (k == 0)) && (beta == one))) {
        return;
    }

    // And when alpha.eq.zero.
    if (alpha == zero) {
        //Console.WriteLine("a=0");
        if (upper) {
            //Console.WriteLine("a=0 1");

```

```

        if (beta == zero) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
        }
    } else {
        //Console.WriteLine("a=0 2");
        if (beta == zero) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
        }
    }
}
return;
}

// Start the operations.
if (trans.Substring(0,1).ToUpper() == "N") {
    //Form C := alpha*A*A**T + beta*C.
    if (upper) {
        //Console.WriteLine("bef dsyrk 1");
        for (int j = 1; j <= n; j = j + 1) {
            if (beta == zero) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            } else if (beta != one) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
            for (int l = 1; l <= k; l = l + 1) {
                if (a[j-1,l-1] != zero) {
                    temp = alpha*a[j-1,l-1];
                    for (int i = 1; i <= j; i = i + 1) {
                        c[i-1,j-1] = c[i-1,j-1] + temp*a[i-1,l-1];
                    }
                }
            }
        }
        //Console.WriteLine("dsyrk 1");
    } else {
        //Console.WriteLine("bef dsyrk 2");
        for (int j = 1; j <= n; j = j + 1) {
            if (beta == zero) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            } else if (beta != one) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
            for (int l = 1; l <= k; l = l + 1) {
                if (a[j-1,l-1] != zero) {
                    temp = alpha*a[j-1,l-1];
                    for (int i = j; i <= n; i = i + 1) {
                        c[i-1,j-1] = c[i-1,j-1] + temp*a[i-1,l-1];
                    }
                }
            }
        }
    }
}
//Console.WriteLine("dsyrk 2");

} else {
    // Form C := alpha*A**T*A + beta*C.
    if (upper) {
        //Console.WriteLine("bef dsyrk 3");
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= j; i = i + 1) {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1) {
                    temp = temp + a[l-1,i-1]*a[l-1,j-1];
                }
                if (beta == zero) {
                    c[i-1,j-1] = alpha*temp;
                } else {
                    c[i-1,j-1] = alpha*temp+beta*c[i-1,j-1];
                }
            }
        }
        //Console.WriteLine("dsyrk 3");
    } else {
        //Console.WriteLine("bef dsyrk 4");
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = j; i <= n; i = i + 1) {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1) {
                    temp = temp + a[l-1,i-1]*a[l-1,j-1];
                }
                if (beta == zero) {
                    c[i-1,j-1] = alpha*temp;
                } else {
                    c[i-1,j-1] = alpha*temp + beta*c[i-1,j-1];
                }
            }
        }
    }
}
}

```

```

    }
    }
    //Console.WriteLine("dsyrk 4");
}

//Console.WriteLine("dsyrk out");

}

public static void DSYR2K(string uplo, string trans, int n, int k, double alpha, double[,] a, int lda, double[,] b, int ldb, double beta, ref
double[,] c, int ldc)
{
    // DSYR2K (UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
    // DOUBLE PRECISION A (LDA,*),B (LDB,*),C (LDC,*)

    // DSYR2K performs one of the symmetric rank 2k operations
    // C := alpha*A*B**T + alpha*B*A**T + beta*C,
    // or
    // C := alpha*A**T*B + alpha*B**T*A + beta*C,
    // where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n
    matrices in the second case.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' C := alpha*A*B**T + alpha*B*A**T + beta*C.
    // TRANS = 'T' or 't' C := alpha*A**T*B + alpha*B**T*A + beta*C.
    // TRANS = 'C' or 'c' C := alpha*A**T*B + alpha*B**T*A + beta*C.

    // N is INTEGER
    // On entry, N specifies the order of the matrix C. N must be at least zero.

    // K is INTEGER
    // On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't'
    or 'C' or 'c', K specifies the number
    // of rows of the matrices A and B. K must be at least zero.

    // ALPHA is DOUBLE PRECISION.
    // On entry, ALPHA specifies the scalar alpha.

    // A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with
    TRANS = 'N' or 'n', the leading n by k
    // part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
    at least max( 1, n ), otherwise LDA must
    // be at least max( 1, k ).

    // B is DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with
    TRANS = 'N' or 'n', the leading n by k
    // part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

    // LDB is INTEGER
    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be
    at least max( 1, n ), otherwise LDB must
    // be at least max( 1, k ).

    // BETA is DOUBLE PRECISION.
    // On entry, BETA specifies the scalar beta.

    // C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
    the symmetric matrix and the strictly
    // lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
    part of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
    the symmetric matrix and the strictly
    // upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
    part of the updated matrix.

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

    // Local Scalars
    double temp1, temp2;
    int info, nrowa;
    bool upper;

    // Parameters
    const double one = 1.0;
    const double zero = 0.0;

    // Test the input parameters
    if (trans.Substring(0,1).ToUpper() == "N") {
        nrowa = n;
    } else {
        nrowa = k;
    }
    upper = (uplo.Substring(0,1).ToUpper() == "U");

    info = 0;

    if (!upper && !(uplo.Substring(0,1).ToUpper() == "L")) {
        info = 1;
    } else if (!(trans.Substring(0,1).ToUpper() == "N") && !(trans.Substring(0,1).ToUpper() == "T") && !(trans.Substring(0,1).ToUpper() == "C")) {
        info = 2;
    } else if (n < 0) {
        info = 3;
    } else if (k < 0) {
        info = 4;
    } else if (lda < Math.Max(1,nrowa)) {
        info = 7;
    } else if (ldb < Math.Max(1,nrowa)) {
        info = 9;
    } else if (ldc < Math.Max(1,n)) {
        info = 12;
    }
    if (info != 0) {
        XERBLA("DSYR2K",info);
    }
}

```

```

    return;
}

// Quick return if possible.
if ((n == 0) || ((alpha == zero) || (k == 0)) && (beta == one)) {
    return;
}

// And when alpha.eq.zero.
if (alpha == zero) {
    if (upper) {
        if (beta == zero) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
        }
    } else {
        if (beta == zero) {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            }
        } else {
            for (int j = 1; j <= n; j = j + 1) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
        }
    }
    return;
}

// Start the operations.
if (trans.Substring(0,1).ToUpper() == "N") {
    // Form C := alpha*A*B**T + alpha*B*A**T + C.
    if (upper) {
        for (int j = 1; j <= n; j = j + 1) {
            if (beta == zero) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            } else if (beta != one) {
                for (int i = 1; i <= j; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
            for (int l = 1; l <= k; l = l + 1) {
                if ((a[j-1,l-1] != zero) || (b[j-1,l-1] != zero)) {
                    temp1 = alpha*b[j-1,l-1];
                    temp2 = alpha*a[j-1,l-1];
                    for (int i = 1; i <= j; i = i + 1) {
                        c[i-1,j-1] = c[i-1,j-1] + a[i-1,l-1]*temp1+b[i-1,l-1]*temp2;
                    }
                }
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            if (beta == zero) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = zero;
                }
            } else if (beta != one) {
                for (int i = j; i <= n; i = i + 1) {
                    c[i-1,j-1] = beta*c[i-1,j-1];
                }
            }
            for (int l = 1; l <= k; l = l + 1) {
                if ((a[j-1,l-1] != one) || (b[j-1,l-1] != zero)) {
                    temp1 = alpha*b[j-1,l-1];
                    temp2 = alpha*a[j-1,l-1];
                    for (int i = j; i <= n; i = i + 1) {
                        c[i-1,j-1] = c[i-1,j-1] + a[i-1,l-1]*temp1+b[i-1,l-1]*temp2;
                    }
                }
            }
        }
    }
} else {
    // Form C := alpha*A**T*B + alpha*B**T*A + C.
    if (upper) {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = 1; i <= j; i = i + 1) {
                temp1 = zero;
                temp2 = zero;
                for (int l = 1; l <= k; l = l + 1) {
                    temp1 = temp1 + a[l-1,i-1]*b[l-1,j-1];
                    temp2 = temp2 + b[l-1,i-1]*a[l-1,j-1];
                }
                if (beta == zero) {
                    c[i-1,j-1] = alpha*temp1 + alpha*temp2;
                } else {
                    c[i-1,j-1] = beta*c[i-1,j-1] + alpha*temp1+alpha*temp2;
                }
            }
        }
    } else {
        for (int j = 1; j <= n; j = j + 1) {
            for (int i = j; i <= n; i = i + 1) {
                temp1 = zero;
                temp2 = zero;
                for (int l = 1; l <= k; l = l + 1) {
                    temp1 = temp1 + a[l-1,i-1]*b[l-1,j-1];
                    temp2 = temp2 + b[l-1,i-1]*a[l-1,j-1];
                }
            }
        }
    }
}

```



```

        return dtmp;
    }*/
    static Complex[] get_zarr2d(int start, int end, int endl, Complex[,] zarr)
    {
        Complex[] ztmp = new Complex[end - start + 1];
        int cnt = 0;
        for (int ii = start; ii <= end; ii = ii + 1)
        {
            ztmp[cnt] = zarr[ii, endl];
            cnt++;
        }
        return ztmp;
    }
    static Complex[] get_zarr3d(int start, int end, int endl, int end2, Complex[, ,] zarr)
    {
        Complex[] ztmp = new Complex[end - start + 1];
        int cnt = 0;
        for (int ii = start; ii <= end; ii = ii + 1)
        {
            ztmp[cnt] = zarr[ii, endl, end2];
            cnt++;
        }
        return ztmp;
    }
    static Complex[] get_zarr_full(int l, int start, int end, Complex[] arr)
    {
        //double[] dtmp = new double[end - start + 1];
        Complex[] ztmp = new Complex[1];
        int cnt = 0;
        for (int ii = start; ii <= end; ii = ii + 1)
        {
            ztmp[cnt] = arr[ii];
            cnt++;
        }
        return ztmp;
    }
    /*
    static float[] get_darr2df(int start, int end, int endl, double[,] arr)
    {
        float[] dtmp = new float[end - start + 1];
        int cnt = 0;
        for (int ii = start; ii <= end; ii = ii + 1)
        {
            dtmp[cnt] = (float)arr[ii, endl];
            cnt++;
        }
        return dtmp;
    }*/
    /*
    static double mysign(double a, double b)
    {
        Console.WriteLine("{0} {1} {2}", a, b, Math.Sign(b));
        if (b < 0.0) {
            return - Math.Abs(a);
        } else {
            return Math.Abs(a);
        }
    }
    */
    static void HEADER()
    {
        // Parameters
        const int nout = 6;

        // Local Arrays
        String[] l = {
            /*" DDOT ",
            "DAXPY ",
            "DROTG ",
            " DROT ",
            "DCOPY ",
            "DSWAP ",
            "DNRM2 ",
            "DASUM ",
            "DSCAL ",
            "IDAMAX",
            "DROTMG",
            "DROTM ",
            "DSDOT "};*/

            "ZDOTC ",
            "ZDOTU ",
            "ZAXPY ",
            "ZCOPY ",
            "ZSWAP ",
            "DZNRM2",
            "DZASUM",
            "ZSCAL ",
            "ZDSCAL",
            "IZAMAX"};

        //          COMMON          /COMBLA/ICASE, N, INCX, INCY, PASS, MODE

        /*{ index[,alignment] [ :formatString] }
        Console.WriteLine("");
        //',13,12X,A6)ICASE, L(ICASE)
        Console.WriteLine("Test of subprogram number {0,3:D}          {1:-6}", icase, l[icase - 1]);
    }
    static void CHECK1(double sfac)
    {
        // Parameters
        const int nout = 6;

        // Local Scalars
        Complex ca = new Complex(0.4, -0.7);
        double sa = 0.3;
        int i, j, len, npl;

        //COMPLEX*16          CTRUE5(8,5,2), CTRUE6(8,5,2), CV(8,5,2), CX(8),
        //+                      MWPCS(5), MWPCT(5)
        //DOUBLE PRECISION    STRUE2(5), STRUE4(5)
        //INTEGER              ITRUE3(5)

```

```

// Local Arrays //

Complex[, ] ctr5 = new Complex[8, 5, 2]; // = new Complex[8,5,2]();

double[, ] ctr5_rk0 = {
    {0.1, -0.16, -0.17, 0.11, 0.19},
    {1.0, 3.0, 0.13, -0.17, 0.2},
    {1.0, 3.0, 5.0, -0.17, 0.35},
    {1.0, 3.0, 5.0, 7.0, 0.14},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0}};
double[, ] ctr5_ilk0 = {
    {0.1, -0.37, -0.19, -0.03, -0.17},
    {2.0, 4.0, -0.39, 0.46, -0.35},
    {2.0, 4.0, 6.0, -0.19, 0.2},
    {2.0, 4.0, 6.0, 8.0, 0.08},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0}};
double[, ] ctr5_rk1 = {
    {0.1, -0.16, -0.17, 0.11, 0.19},
    {4.0, 6.0, 8.0, 3.0, 5.0},
    {4.0, 6.0, 0.13, -0.17, 0.2},
    {4.0, 6.0, 2.0, 4.0, 6.0},
    {4.0, 6.0, 2.0, -0.17, 0.35},
    {4.0, 6.0, 2.0, 7.0, 8.0},
    {4.0, 6.0, 2.0, 7.0, 0.14},
    {4.0, 6.0, 2.0, 7.0, 9.0}};
double[, ] ctr5_ilk1 = {
    {0.1, -0.37, -0.19, -0.03, -0.17},
    {5.0, 7.0, 9.0, 6.0, 8.0},
    {5.0, 7.0, -0.39, 0.46, -0.35},
    {5.0, 7.0, 5.0, 7.0, 9.0},
    {5.0, 7.0, 5.0, -0.19, 0.2},
    {5.0, 7.0, 5.0, 2.0, 3.0},
    {5.0, 7.0, 5.0, 2.0, 0.08},
    {5.0, 7.0, 5.0, 2.0, 4.0}};

Complex[, ] ctr6 = new Complex[8, 5, 2];
double[, ] ctr6_rk0 = {
    {0.10, 0.09, 0.03, 0.03, 0.09},
    {1.0, 3.0, 0.15, -0.18, 0.15},
    {1.0, 3.0, 5.0, 0.03, 0.0},
    {1.0, 3.0, 5.0, 7.0, 0.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0}};
double[, ] ctr6_ilk0 = {
    {0.1, -0.12, -0.09, 0.03, 0.03},
    {2.0, 4.0, -0.03, 0.03, 0.0},
    {2.0, 4.0, 6.0, -0.09, 0.15},
    {2.0, 4.0, 6.0, 8.0, 0.06},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0}};
double[, ] ctr6_rk1 = {
    {0.1, 0.09, 0.03, 0.03, 0.09},
    {4.0, 6.0, 8.0, 3.0, 5.0},
    {4.0, 6.0, 0.15, -0.18, 0.15},
    {4.0, 6.0, 2.0, 4.0, 6.0},
    {4.0, 6.0, 2.0, 0.03, 0.0},
    {4.0, 6.0, 2.0, 7.0, 8.0},
    {4.0, 6.0, 2.0, 7.0, 0.0},
    {4.0, 6.0, 2.0, 7.0, 9.0}};
double[, ] ctr6_ilk1 = {
    {0.1, -0.12, -0.09, 0.03, 0.03},
    {5.0, 7.0, 9.0, 6.0, 8.0},
    {5.0, 7.0, -0.03, 0.03, 0.0},
    {5.0, 7.0, 5.0, 7.0, 9.0},
    {5.0, 7.0, 5.0, -0.09, 0.15},
    {5.0, 7.0, 5.0, 2.0, 3.0},
    {5.0, 7.0, 5.0, 2.0, 0.06},
    {5.0, 7.0, 5.0, 2.0, 4.0}};

Complex[, ] cv = new Complex[8, 5, 2];

double[, ] cv_rk0 = {
    {0.1, 0.3, 0.1, 0.1, 0.3},
    {1.0, 3.0, 0.5, -0.6, 0.5},
    {1.0, 3.0, 5.0, 0.1, 0.0},
    {1.0, 3.0, 5.0, 7.0, 0.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0},
    {1.0, 3.0, 5.0, 7.0, 2.0}};
double[, ] cv_ilk0 = {
    {0.1, -0.4, -0.3, 0.1, 0.1},
    {2.0, 4.0, -0.1, 0.1, 0.0},
    {2.0, 4.0, 6.0, -0.3, 0.5},
    {2.0, 4.0, 6.0, 8.0, 0.2},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0},
    {2.0, 4.0, 6.0, 8.0, 3.0}};

double[, ] cv_rk1 = {
    {0.1, 0.3, 0.1, 0.1, 0.3},
    {4.0, 6.0, 8.0, 3.0, 5.0},
    {4.0, 6.0, 0.5, -0.6, 0.5},
    {4.0, 6.0, 2.0, 4.0, 6.0},
    {4.0, 6.0, 2.0, 0.1, 0.0},
    {4.0, 6.0, 2.0, 7.0, 8.0},
    {4.0, 6.0, 2.0, 7.0, 0.0},
    {4.0, 6.0, 2.0, 7.0, 9.0}};
double[, ] cv_ilk1 = {
    {0.1, -0.4, -0.3, 0.1, 0.1},

```

```

        {5.0, 7.0, 9.0, 6.0, 8.0},
        {5.0, 7.0, -0.1, 0.1, 0.0},
        {5.0, 7.0, 5.0, 7.0, 9.0},
        {5.0, 7.0, 5.0, -0.3, 0.5},
        {5.0, 7.0, 5.0, 2.0, 3.0},
        {5.0, 7.0, 5.0, 2.0, 0.2},
        {5.0, 7.0, 5.0, 2.0, 4.0});

Complex[] cx = new Complex[8];
Complex[] mwpcs = new Complex[5];
Complex[] mwpct = new Complex[5];

double[] strue2 = { 0.0, 0.5, 0.6, 0.7, 0.8 };

double[] strue4 = { 0.0, 0.7, 1.0, 1.3, 1.6 };
int[] itrue3 = { 0, 1, 2, 2, 2 };

//COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
// Initialize Arrays
for (i = 1; i <= 8; i = i + 1)
{
    for (j = 1; j <= 5; j = j + 1)
    {
        ctrue5[i - 1, j - 1, 0] = new Complex(ctrue5_rkl[i - 1, j - 1], ctrue5_ilk0[i - 1, j - 1]);
        ctrue5[i - 1, j - 1, 1] = new Complex(ctrue5_rkl[i - 1, j - 1], ctrue5_ilk1[i - 1, j - 1]);
        ctrue6[i - 1, j - 1, 0] = new Complex(ctrue6_rkl[i - 1, j - 1], ctrue6_ilk0[i - 1, j - 1]);
        ctrue6[i - 1, j - 1, 1] = new Complex(ctrue6_rkl[i - 1, j - 1], ctrue6_ilk1[i - 1, j - 1]);
        cv[i - 1, j - 1, 0] = new Complex(cv_rkl[i - 1, j - 1], cv_ilk0[i - 1, j - 1]);
        cv[i - 1, j - 1, 1] = new Complex(cv_rkl[i - 1, j - 1], cv_ilk1[i - 1, j - 1]);
        // Console.WriteLine("{0} {1}", i, j);
        // Console.WriteLine("{0:N15}", ctrue5[i - 1, j - 1, 0]);
        // Console.WriteLine("{0:N15}", ctrue5[i - 1, j - 1, 1]);
        // Console.WriteLine("{0:N15}", ctrue6[i - 1, j - 1, 0]);
        // Console.WriteLine("{0:N15}", ctrue6[i - 1, j - 1, 1]);
        // Console.WriteLine("{0:N15}", cv[i - 1, j - 1, 0]);
        // Console.WriteLine("{0:N15}", cv[i - 1, j - 1, 1]);
    }
}
//Environment.Exit(1);
// Executable Statements
for (incx = 1; incx <= 2; incx = incx + 1)
{
    for (npl = 1; npl <= 5; npl = npl + 1)
    {
        n = npl - 1;
        len = 2 * Math.Max(n, 1);
        // Set vector arguments
        for (i = 1; i <= len; i = i + 1)
        {
            cx[i - 1] = cv[i - 1, npl - 1, incx - 1];
            //sx[i-1] = dv[i-1,npl-1, incx -1];
        }
        if (icase == 6)
        {
            // DZNRM2
            //STEST1(DZNRM2(n,cx,incx),strue2[npl-1],strue2[npl-1], sfac);
            // Console.WriteLine("n:{0} incx:{1}",n,incx);
            //for (int ii = 0; ii < 8; ii = ii + 1)
            //{
            //    Console.WriteLine("{0}",cx[ii]);
            //}
            //Console.WriteLine("{0}", DZNRM2(n, cx, incx));
            //Console.WriteLine("END DZNRM2***");
            //Environment.Exit(1);
            STEST(1, new double[] { DZNRM2(n, cx, incx) }, new double[] { strue2[npl - 1] }, get_darr(npl - 1, 4, strue2), sfac);
        }
        //STEST(int len, double[] scomp, double[] strue, double[] ssize, double sfa
    }
    else if (icase == 7)
    {
        // DZASUM
        //STEST1(DZASUM(n,cx,incx),strue4[npl-1],strue4[npl-1], sfac);
        //Console.WriteLine("n:{0} incx:{1}",n,incx);
        //for (int ii = 0; ii < 8; ii = ii + 1)
        //{
        //    Console.WriteLine("{0}",cx[ii]);
        //}
        //Console.WriteLine("{0}", DZASUM(n, cx, incx));
        //Console.WriteLine("END DZASUM***");
        STEST(1, new double[] { DZASUM(n, cx, incx) }, new double[] { strue4[npl - 1] }, get_darr(npl - 1, 4, strue4), sfac);
    }
    else if (icase == 8)
    {
        // ZSCAL
        ZSCAL(n, ca, ref cx, incx);
        CTEST(len, cx, get_zarr3d(0, 7, npl - 1, incx - 1, ctrue5), get_zarr3d(0, 7, npl - 1, incx - 1, ctrue5), sfac);
    }
    else if (icase == 9)
    {
        // ZDSCAL
        ZDSCAL(n, sa, ref cx, incx);
        CTEST(len, cx, get_zarr3d(0, 7, npl - 1, incx - 1, ctrue6), get_zarr3d(0, 7, npl - 1, incx - 1, ctrue6), sfac);
    }
    else if (icase == 10)
    {
        // IZAMAX
        ITEST1(IZAMAX(n, cx, incx), itrue3[npl - 1]);
    }
    else
    {
        Console.WriteLine("Shouldn't be here in CHECK1");
        Environment.Exit(1);
    }
}
//COMPLEX*16 CTRUE5(8,5,2), CTRUE6(8,5,2), CV(8,5,2), CX(8),
//+ MWPCS(5), MWPCT(5)
//DOUBLE PRECISION STRUE2(5), STRUE4(5)
//INTEGER ITRUE3(5)
}
incx = 1;
if (icase == 8)
{

```

```

// ZSCAL
// Add a test for alpha equal to zero.
ca = new Complex(0.0, 0.0);
for (i = 1; i <= 5; i = i + 1)
{
    mwpct[i - 1] = new Complex(0.0, 0.0);
    mwpcs[i - 1] = new Complex(1.0, 1.0);
}
ZSCAL(5, ca, ref cx, incx);
CTEST(5, cx, mwpct, mwpcs, sfac);
}
else if (icase == 9)
{
    // ZDSCAL
    // Add a test for alpha equal to zero.
    sa = 0.0;
    for (i = 1; i <= 5; i = i + 1)
    {
        mwpct[i - 1] = new Complex(0.0, 0.0);
        mwpcs[i - 1] = new Complex(1.0, 1.0);
    }
    ZDSCAL(5, sa, ref cx, incx);
    CTEST(5, cx, mwpct, mwpcs, sfac);
    // Add a test for alpha equal to one.
    sa = 1.0;
    for (i = 1; i <= 5; i = i + 1)
    {
        mwpct[i - 1] = cx[i - 1];
        mwpcs[i - 1] = cx[i - 1];
    }
    ZDSCAL(5, sa, ref cx, incx);
    CTEST(5, cx, mwpct, mwpcs, sfac);
    // Add a test for alpha equal to minus one.
    sa = -1.0;
    for (i = 1; i <= 5; i = i + 1)
    {
        mwpct[i - 1] = -cx[i - 1];
        mwpcs[i - 1] = -cx[i - 1];
    }
    ZDSCAL(5, sa, ref cx, incx);
    CTEST(5, cx, mwpct, mwpcs, sfac);
}
}
static void CHECK2(double sfac)
{
    // Parameters
    const int nout = 6;

    // Local Scalars
    Complex ca = new Complex(0.4, -0.7);
    double sa = 0.3;
    //int i, j, ki, kn, kni, kpar, ksize, lenx, leny, mx, my;
    int i, j, ki, kn, ksize, lenx, leny, mx, my;

    /* COMPLEX*16          CDOT(1), CSIZE1(4), CSIZE2(7,2), CSIZE3(14),
+      CT10X(7,4,4), CT10Y(7,4,4), CT6(4,4), CT7(4,4),
+      CT8(7,4,4), CX(7), CX1(7), CY(7), CY1(7)
+      INTEGER          INCXS(4), INCYS(4), LENS(4,2), NS(4) */

    // Local Arrays
    Complex[] cdot = new Complex[1];
    Complex[] csizel = new Complex[4];
    double[] csizel_r1 = { 0.0, 0.9, 1.63, 2.9 };
    double[] csizel_i1 = { 0.0, 0.9, 1.73, 2.78 };

    Complex[,] csizel2 = new Complex[7, 2];
    double[,] csizel2_r1 = {
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54}};
    double[,] csizel2_i1 = {
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54},
        {0.0, 1.54}};

    Complex[] csizel3 = new Complex[14];
    double[] csizel3_r1 = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17 };
    double[] csizel3_i1 = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17 };

    Complex[, ] ct10x = new Complex[7, 4, 4];
    double[,] ct10x_r1k0 = {
        {0.7, 0.6, 0.6, 0.6},
        {0.0, 0.0, -0.9, -0.9},
        {0.0, 0.0, 0.0, 0.7},
        {0.0, 0.0, 0.0, 0.1},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0}};
    double[,] ct10x_i1k0 = {
        {-0.8, -0.6, -0.6, -0.6},
        {0.0, 0.0, 0.5, 0.5},
        {0.0, 0.0, 0.0, -0.6},
        {0.0, 0.0, 0.0, -0.5},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.8, 0.0, 0.0, 0.0}};

    double[,] ct10x_r1k1 = {
        {0.7, 0.6, 0.7, 0.8},
        {0.0, 0.0, -0.4, -0.4},
        {0.0, 0.0, 0.6, -0.1},
        {0.0, 0.0, 0.0, 0.2},
        {0.0, 0.0, 0.0, 0.7},
        {0.0, 0.0, 0.0, 0.1},

```

```

        {0.0, 0.0, 0.0, 0.6});
double[,] ct10x_ilk1 = {
    {-0.8, -0.6, -0.6, -0.7},
    {0.0, 0.0, -0.7, -0.7},
    {0.0, 0.0, -0.6, -0.2},
    {0.0, 0.0, 0.0, -0.8},
    {0.0, 0.0, 0.0, -0.6},
    {0.0, 0.0, 0.0, 0.4},
    {0.0, 0.0, 0.0, -0.6}};

double[,] ct10x_rlk2 = {
    {0.7, 0.6, -0.9, 0.1},
    {0.0, 0.0, -0.4, -0.4},
    {0.0, 0.0, 0.6, 0.7},
    {0.0, 0.0, 0.0, 0.2},
    {0.0, 0.0, 0.0, -0.9},
    {0.0, 0.0, 0.0, 0.1},
    {0.0, 0.0, 0.0, 0.6}};

double[,] ct10x_ilk2 = {
    {-0.8, -0.6, 0.5, -0.5},
    {0.0, 0.0, -0.7, -0.7},
    {0.0, 0.0, -0.6, -0.6},
    {0.0, 0.0, 0.0, -0.8},
    {0.0, 0.0, 0.0, 0.5},
    {0.0, 0.0, 0.0, 0.4},
    {0.0, 0.0, 0.0, -0.6}};

double[,] ct10x_rlk3 = {
    {0.7, 0.6, 0.6, 0.6},
    {0.0, 0.0, 0.7, 0.7},
    {0.0, 0.0, 0.0, -0.1},
    {0.0, 0.0, 0.0, 0.8},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct10x_ilk3 = {
    {-0.8, -0.6, -0.6, -0.6},
    {0.0, 0.0, -0.6, -0.6},
    {0.0, 0.0, 0.0, -0.2},
    {0.0, 0.0, 0.0, -0.7},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

Complex[, ] ct10y = new Complex[7, 4];

double[,] ct10y_rlk0 = {
    {0.6, 0.7, 0.7, 0.7},
    {0.0, 0.0, -0.4, -0.4},
    {0.0, 0.0, 0.0, -0.1},
    {0.0, 0.0, 0.0, 0.2},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct10y_ilk0 = {
    {-0.6, -0.8, -0.8, -0.8},
    {0.0, 0.0, -0.7, -0.7},
    {0.0, 0.0, 0.0, -0.9},
    {0.0, 0.0, 0.0, -0.8},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct10y_rlk1 = {
    {0.6, 0.7, -0.1, -0.6},
    {0.0, 0.0, -0.9, -0.9},
    {0.0, 0.0, 0.7, -0.9},
    {0.0, 0.0, 0.0, 0.1},
    {0.0, 0.0, 0.0, -0.1},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, 0.7}};

double[,] ct10y_ilk1 = {
    {-0.6, -0.8, -0.9, 0.6},
    {0.0, 0.0, 0.5, 0.5},
    {0.0, 0.0, -0.8, -0.4},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, -0.9},
    {0.0, 0.0, 0.0, -0.3},
    {0.0, 0.0, 0.0, -0.8}};

double[,] ct10y_rlk2 = {
    {0.6, 0.7, -0.1, -0.6},
    {0.0, 0.0, 0.7, -0.9},
    {0.0, 0.0, 0.0, -0.1},
    {0.0, 0.0, 0.0, 0.7},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct10y_ilk2 = {
    {-0.6, -0.8, -0.9, 0.6},
    {0.0, 0.0, -0.8, -0.4},
    {0.0, 0.0, 0.0, -0.9},
    {0.0, 0.0, 0.0, -0.8},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct10y_rlk3 = {
    {0.6, 0.7, 0.7, 0.7},
    {0.0, 0.0, -0.9, -0.9},
    {0.0, 0.0, -0.4, -0.4},
    {0.0, 0.0, 0.0, 0.1},
    {0.0, 0.0, 0.0, -0.1},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, 0.2}};

double[,] ct10y_ilk3 = {
    {-0.6, -0.8, -0.8, -0.8},
    {0.0, 0.0, 0.5, 0.5},
    {0.0, 0.0, -0.7, -0.7},
    {0.0, 0.0, 0.0, -0.5},

```

```

        {0.0, 0.0, 0.0, -0.9},
        {0.0, 0.0, 0.0, -0.3},
        {0.0, 0.0, 0.0, -0.8});
Complex[,] ct6 = new Complex[4, 4];

double[,] ct6_r1 = {
    {0.0, 0.0, 0.0, 0.0},
    {0.9, 0.9, 0.9, 0.9},
    {0.91, 1.45, -0.55, 1.04},
    {1.8, 0.2, 0.83, 1.95}};
double[,] ct6_i1 = {
    {0.0, 0.0, 0.0, 0.0},
    {0.06, 0.06, 0.06, 0.06},
    {-0.77, 0.74, 0.23, 0.79},
    {-0.1, 0.9, -0.39, 1.22}};
Complex[,] ct7 = new Complex[4, 4];

double[,] ct7_r1 = {
    {0.0, 0.0, 0.0, 0.0},
    {-0.06, -0.06, -0.06, -0.06},
    {0.65, -0.59, -0.83, -0.76},
    {-0.34, -1.04, 0.07, -1.33}};
double[,] ct7_i1 = {
    {0.0, 0.0, 0.0, 0.0},
    {-0.9, -0.9, -0.9, -0.9},
    {-0.47, -1.46, 0.59, -1.15},
    {-1.22, -0.04, -0.37, -1.82}};
Complex[, ] ct8 = new Complex[7, 4, 4];

double[,] ct8_r1k0 = {
    {0.6, 0.32, 0.32, 0.32},
    {0.0, 0.0, -1.55, -1.55},
    {0.0, 0.0, 0.0, 0.03},
    {0.0, 0.0, 0.0, -0.38},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};
double[,] ct8_i1k0 = {
    {-0.6, -1.41, -1.41, -1.41},
    {0.0, 0.0, 0.5, 0.5},
    {0.0, 0.0, 0.0, -0.89},
    {0.0, 0.0, 0.0, -0.96},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct8_r1k1 = {
    {0.6, 0.32, -0.07, 0.78},
    {0.0, 0.0, -0.9, -0.9},
    {0.0, 0.0, 0.42, 0.06},
    {0.0, 0.0, 0.0, 0.1},
    {0.0, 0.0, 0.0, -0.77},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, 0.52}};
double[,] ct8_i1k1 = {
    {-0.6, -1.41, -0.89, 0.06},
    {0.0, 0.0, 0.5, 0.5},
    {0.0, 0.0, -1.41, -0.13},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, -0.49},
    {0.0, 0.0, 0.0, -0.3},
    {0.0, 0.0, 0.0, -1.51}};

double[,] ct8_r1k2 = {
    {0.6, 0.32, -0.07, 0.78},
    {0.0, 0.0, -1.18, -1.54},
    {0.0, 0.0, 0.0, 0.03},
    {0.0, 0.0, 0.0, -0.18},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};
double[,] ct8_i1k2 = {
    {-0.6, -1.41, -0.89, 0.06},
    {0.0, 0.0, -0.31, 0.97},
    {0.0, 0.0, 0.0, -0.89},
    {0.0, 0.0, 0.0, -1.31},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}};

double[,] ct8_r1k3 = {
    {0.6, 0.32, 0.32, 0.32},
    {0.0, 0.0, -0.9, -0.9},
    {0.0, 0.0, 0.05, 0.05},
    {0.0, 0.0, 0.0, 0.1},
    {0.0, 0.0, 0.0, -0.77},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, 0.32}};
double[,] ct8_i1k3 = {
    {-0.6, -1.41, -1.41, -1.41},
    {0.0, 0.0, 0.5, 0.5},
    {0.0, 0.0, -0.6, -0.6},
    {0.0, 0.0, 0.0, -0.5},
    {0.0, 0.0, 0.0, -0.49},
    {0.0, 0.0, 0.0, -0.3},
    {0.0, 0.0, 0.0, -1.16}};

Complex[] cx = new Complex[7];
Complex[] cx1 = new Complex[7];
double[] cx1_r1 = { 0.7, -0.4, -0.1, 0.2, -0.9, 0.1, -0.6 };
double[] cx1_i1 = { -0.8, -0.7, -0.9, -0.8, -0.4, 0.4, 0.6 };
Complex[] cy = new Complex[7];
Complex[] cy1 = new Complex[7];

double[] cy1_r1 = { 0.6, -0.9, 0.7, 0.1, -0.1, -0.5, 0.8 };
double[] cy1_i1 = { -0.6, 0.5, -0.6, -0.5, -0.2, -0.3, -0.7 };

int[] incxs = { 1, 2, -2, -1 };
int[] incys = { 1, -2, 1, -2 };
int[,] lens = {{1, 1}},

```

```

        {1, 1 },
        {2, 3 },
        {4, 7 } };
int[] ns = { 0, 1, 2, 4 };

/* COMPLEX*16      CDOT(1), CSIZE1(4), CSIZE2(7,2), CSIZE3(14),
+      CT10X(7,4,4), CT10Y(7,4,4), CT6(4,4), CT7(4,4),
+      CT8(7,4,4), CX(7), CX1(7), CY(7), CY1(7)
+      INCXS(4), INCYS(4), LENS(4,2), NS(4) */

// Initialize Arrays
for (int ii = 1; ii <= 4; ii = ii + 1)
{
    csizel[ii - 1] = new Complex(csizel_r1[ii - 1], csizel_i1[ii - 1]);
}
for (int ii = 1; ii <= 7; ii = ii + 1)
{
    for (int jj = 1; jj <= 2; jj = jj + 1)
    {
        csizel2[ii - 1, jj - 1] = new Complex(csizel2_r1[ii - 1, jj - 1], csizel2_i1[ii - 1, jj - 1]);
    }
}
for (int ii = 1; ii <= 14; ii = ii + 1)
{
    csizel3[ii - 1] = new Complex(csizel3_r1[ii - 1], csizel3_i1[ii - 1]);
}

for (int ii = 1; ii <= 7; ii = ii + 1)
{
    for (int jj = 1; jj <= 4; jj = jj + 1)
    {
        ct10x[ii - 1, jj - 1, 0] = new Complex(ct10x_r1k0[ii - 1, jj - 1], ct10x_ik0[ii - 1, jj - 1]);
        ct10x[ii - 1, jj - 1, 1] = new Complex(ct10x_r1k1[ii - 1, jj - 1], ct10x_ik1[ii - 1, jj - 1]);
        ct10x[ii - 1, jj - 1, 2] = new Complex(ct10x_r1k2[ii - 1, jj - 1], ct10x_ik2[ii - 1, jj - 1]);
        ct10x[ii - 1, jj - 1, 3] = new Complex(ct10x_r1k3[ii - 1, jj - 1], ct10x_ik3[ii - 1, jj - 1]);

        ct10y[ii - 1, jj - 1, 0] = new Complex(ct10y_r1k0[ii - 1, jj - 1], ct10y_ik0[ii - 1, jj - 1]);
        ct10y[ii - 1, jj - 1, 1] = new Complex(ct10y_r1k1[ii - 1, jj - 1], ct10y_ik1[ii - 1, jj - 1]);
        ct10y[ii - 1, jj - 1, 2] = new Complex(ct10y_r1k2[ii - 1, jj - 1], ct10y_ik2[ii - 1, jj - 1]);
        ct10y[ii - 1, jj - 1, 3] = new Complex(ct10y_r1k3[ii - 1, jj - 1], ct10y_ik3[ii - 1, jj - 1]);

        ct8[ii - 1, jj - 1, 0] = new Complex(ct8_r1k0[ii - 1, jj - 1], ct8_ik0[ii - 1, jj - 1]);
        ct8[ii - 1, jj - 1, 1] = new Complex(ct8_r1k1[ii - 1, jj - 1], ct8_ik1[ii - 1, jj - 1]);
        ct8[ii - 1, jj - 1, 2] = new Complex(ct8_r1k2[ii - 1, jj - 1], ct8_ik2[ii - 1, jj - 1]);
        ct8[ii - 1, jj - 1, 3] = new Complex(ct8_r1k3[ii - 1, jj - 1], ct8_ik3[ii - 1, jj - 1]);
    }
}
for (int ii = 1; ii <= 4; ii = ii + 1)
{
    for (int jj = 1; jj <= 4; jj = jj + 1)
    {
        ct6[ii - 1, jj - 1] = new Complex(ct6_r1[ii - 1, jj - 1], ct6_i1[ii - 1, jj - 1]);
        ct7[ii - 1, jj - 1] = new Complex(ct7_r1[ii - 1, jj - 1], ct7_i1[ii - 1, jj - 1]);
    }
}
for (int ii = 1; ii <= 7; ii = ii + 1)
{
    cx1[ii - 1] = new Complex(cx1_r1[ii - 1], cx1_i1[ii - 1]);
    cyl[ii - 1] = new Complex(cyl_r1[ii - 1], cyl_i1[ii - 1]);
}
//
//Environment.Exit(1);
// Executable Statements

for (ki = 1; ki <= 4; ki = ki + 1)
{ //120
    incx = incxs[ki - 1];
    incy = incys[ki - 1];
    mx = Math.Abs(incx);
    my = Math.Abs(incy);

    for (kn = 1; kn <= 4; kn = kn + 1)
    { //100
        n = ns[kn - 1];
        ksize = Math.Min(2, kn);
        lenx = lens[kn - 1, mx - 1];
        leny = lens[kn - 1, my - 1];

        // Initialize all argument arrays
        for (i = 1; i <= 7; i = i + 1)
        { //20
            cx[i - 1] = cx1[i - 1];
            cy[i - 1] = cyl[i - 1];
        }

        if (icase == 1)
        {
            // DDOT
            //STEST1(DDOT(n, sx, incx, sy, incy), dt7[kn-1,ki-1], ssizel[kn-1], sfac);
            //STEST(1,DDOT(n, sx, incx, sy, incy), dt7[kn-1,ki-1], ssizel[kn-1], sfac);
            //STEST(1,new double[] {DDOT(n, sx, incx, sy, incy)}, get_darr2d(kn-1,3,ki-1,dt7),get_darr(kn-1,3, ssizel), sfac);

            // ZDOTC ..
            cdot[0] = ZDOTC(n, cx, incx, cy, incy);
            CTEST(1, cdot, get_zarr2d(kn - 1, 3, ki - 1, ct6), get_zarr(kn - 1, 3, csizel), sfac);
        }
        else if (icase == 2)
        {
            // ZDOTU
            cdot[0] = ZDOTU(n, cx, incx, cy, incy);
            CTEST(1, cdot, get_zarr2d(kn - 1, 3, ki - 1, ct7), get_zarr(kn - 1, 3, csizel), sfac);
        }

        else if (icase == 3)
        {
            // ZAXPY
            ZAXPY(n, ca, cx, incx, ref cy, incy);
            CTEST(leny, cy, get_zarr3d(0, 6, kn - 1, ki - 1, ct8), get_zarr2d(0, 6, ksize - 1, csizel2), sfac);
        }
    }
}

```

```

        else if (icase == 4)
        {
            // ZCOPY
            ZCOPY(n, cx, incx, ref cy, incy);
            CTEST(leny, cy, get_zarr3d(0, 6, kn - 1, ki - 1, ct10y), csize3, 1.0);
        }
        else if (icase == 5)
        {
            // ZSWAP
            ZSWAP(n, ref cx, incx, ref cy, incy);
            CTEST(lenx, cx, get_zarr3d(0, 6, kn - 1, ki - 1, ct10x), csize3, 1.0);
            CTEST(leny, cy, get_zarr3d(0, 6, kn - 1, ki - 1, ct10y), csize3, 1.0);
        }
        else
        {
            Console.WriteLine("Shouldn't be here in CHECK2");
            Environment.Exit(1);
        }
    }
    /* COMPLEX*16          CDOT(1), CSIZE1(4), CSIZE2(7,2), CSIZE3(14),
+      CT10X(7,4,4), CT10Y(7,4,4), CT6(4,4), CT7(4,4),
+      CT8(7,4,4), CX(7), CX1(7), CY(7), CY1(7)
+      INCXS(4), INCYS(4), LENS(4,2), NS(4) */
    INTEGER
    }
}

static void STEST(int len, double[] scomp, double[] strue, double[] ssize, double sfac)
{
    // THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
    // NEGLIGIBLE. C. L. LAWSON, JPL, 1974 DEC 10

    // Parameters
    const int nout = 6;
    const double zero = 0.0;

    // Local Scalars
    double sd;
    int i;
    // COMMON /COMBLA/ICASE, N, INCX, INCY, PASS
    /* Console.WriteLine("len: {0}", len);
    Console.WriteLine("scomp: {0}", scomp[0]);
    Console.WriteLine("strue: {0}", strue[0]);
    Console.WriteLine("ssize: {0}", ssize[0]);
    Console.WriteLine("sfac: {0}", sfac);
    Console.WriteLine("d eps: {0}", cs_BLAS.misc_d.eps);
    Console.WriteLine("s_eps: {0}", cs_BLAS.misc_s.eps); */
    //for (i = 1; i <= len; i = i + 1)
    //{
    //    Console.WriteLine("{0,3:D}{1,12:F3}{2,12:F3}{3,12:F3}{4,12:F3}{5,3:D}", i, scomp[i-1], strue[i-1], ssize[i-1], sfac, len);
    //}

    for (i = 1; i <= len; i = i + 1)
    {
        sd = scomp[i - 1] - strue[i - 1];
        // IF (ABS(SFAC*SD) .LE. ABS(SSIZE(I))*EPSILON(ZERO))
        if (Math.Abs(sfac * sd) <= (Math.Abs(ssize[i - 1]) * cs_BLAS.misc_d.eps))
        {
            //epsilon(zero))
            continue;
        }
        // HERE SCOMP(I) IS NOT CLOSE TO STRUE(I).
        if (pass)
        {
            pass = false;
            Console.WriteLine("
            Console.WriteLine("CASE N INCX INCY MODE I          COMP(I)          TRUE(I)  DIFFERENCE
SIZE(I)");
        }
        Console.WriteLine("{0,4:D}{1,3:D}{2,5:D}{3,5:D}{4,5:D}{5,3:D}{6,36:F8}{7,36:F8}{8,12:F4}{9,12:F4}", icase, n, incx, incy, mode, i, scomp[i
- 1], strue[i - 1], sd, ssize[i - 1]);
        /* IF ( .NOT. PASS) GO TO 20
        PRINT FAIL MESSAGE AND HEADER.

        PASS = .FALSE.
        WRITE (NOUT,99999)
        WRITE (NOUT,99998)
        20  WRITE (NOUT,99997) ICASE, N, INCX, INCY, I, SCOMP(I),
        +   STRUE(I), SD, SSIZE(I) */
        //99997 FORMAT (1X,I4,I3,2I5,I3,2D36.8,2D12.4)
    }
}

static void _STEST1(double scomp1, double strue1, double[] ssize, double sfac)
{
    // THIS IS AN INTERFACE SUBROUTINE TO ACCOMODATE THE FORTRAN REQUIREMENT THAT WHEN A DUMMY ARGUMENT IS AN ARRAY, THE
    // ACTUAL ARGUMENT MUST ALSO BE AN ARRAY OR AN ARRAY ELEMENT. -- C.L. LAWSON, JPL, 1978 DEC 6
    // Local Arrays
    double[] scomp = new double[1], strue = new double[1];

    scomp[0] = scomp1;
    strue[0] = strue1;
    //STEST1(sa, datrue[k - 1], datrue[k - 1], sfac);
    STEST(1, scomp, strue, ssize, sfac);
}

static double SDIFF(double sa, double sb)
{
    // COMPUTES DIFFERENCE OF TWO NUMBERS. C. L. LAWSON, JPL 1974 FEB 15
    return (sa - sb);
}

static void CTEST(int len, Complex[] ccomp, Complex[] ctrue, Complex[] csize, double sfac)
{
    // C.L. LAWSON, JPL, 1978 DEC 6
    // Local Scalars
    int i;

    // Local Arrays
    double[] scomp = new double[20];
    double[] ssize = new double[20];
    double[] strue = new double[20];

    // Executable Statements
    for (i = 1; i <= len; i = i + 1)
    {
        scomp[2 * i - 1] = ccomp[i - 1].Real;
        scomp[2 * i] = ccomp[i - 1].Imaginary;
    }
}

```



```

        {
            dtemp = dtemp + dx[ix - 1] * dy[iy - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
    return dtemp;
}

public static Complex ZDOTC(int n, Complex[] zx, int incx, Complex[] zy, int incy)
{
    // C# version of ZDOTC
    //
    // ZDOTC forms the dot product of a vector.

    // Local Scalars
    Complex ztemp;
    int i, ix, iy;

    //DDOT = 0.0d0
    ztemp = new Complex(0.0, 0.0);

    if (n <= 0)
    {
        return ztemp;
    }

    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = ztemp + Complex.Conjugate(zx[i - 1]) * zy[i - 1];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = ztemp + Complex.Conjugate(zx[ix - 1]) * zy[iy - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
    return ztemp;
}

public static Complex ZDOTU(int n, Complex[] zx, int incx, Complex[] zy, int incy)
{
    // C# version of ZDOTC
    //
    // ZDOTC forms the dot product of two vectors.

    // Local Scalars
    Complex ztemp;
    int i, ix, iy;

    //DDOT = 0.0d0
    ztemp = new Complex(0.0, 0.0);

    if (n <= 0)
    {
        return ztemp;
    }

    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = ztemp + zx[i - 1] * zy[i - 1];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = ztemp + zx[ix - 1] * zy[iy - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
    return ztemp;
}

public static void _DAXPY(int n, double da, double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DDOT
    //
    // DAXPY constant times a vector plus a vector. uses unrolled loops for increments equal to one.

```

```

// Local Scalars
int i, ix, iy, m, mpl;

if (n <= 0)
{
    return;
}
if (da == 0.0)
{
    return;
}
if ((incx == 1) & (incy == 1))
{
    // code for both increments equal to 1
    // clean-up loop
    m = n % 4;
    if (m != 0)
    {
        for (i = 1; i <= m; i = i + 1)
        {
            dy[i - 1] = dy[i - 1] + da * dx[i - 1];
        }
    }
    if (n < 4)
    {
        return;
    }
    mpl = m + 1;
    for (i = mpl; i <= n; i = i + 4)
    {
        dy[i - 1] = dy[i - 1] + da * dx[i - 1];
        dy[i] = dy[i] + da * dx[i];
        dy[i + 1] = dy[i + 1] + da * dx[i + 1];
        dy[i + 2] = dy[i + 2] + da * dx[i + 2];
    }
}
else
{
    // code for unequal increments or equal increments not equal to 1
    ix = 1;
    iy = 1;
    if (incx < 0)
    {
        ix = (-n + 1) * incx + 1;
    }
    if (incy < 0)
    {
        iy = (-n + 1) * incy + 1;
    }
    for (i = 1; i <= n; i = i + 1)
    {
        dy[iy - 1] = dy[iy - 1] + da * dx[ix - 1];
        ix = ix + incx;
        iy = iy + incy;
    }
}
}

public static void ZAXPY(int n, Complex za, Complex[] zx, int incx, ref Complex[] zy, int incy)
{
    // C# version of ZAXPY
    //
    // ZAXPY constant times a vector plus a vector. uses unrolled loops for increments equal to one.

    // Local Scalars
    int i, ix, iy;

    if (n <= 0)
    {
        return;
    }
    if (DCABSL(za) == 0.0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            zy[i - 1] = zy[i - 1] + za * zx[i - 1];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            zy[iy - 1] = zy[iy - 1] + za * zx[ix - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}

public static void DROTG(ref double da, ref double db, ref double c, ref double s)
{
    // C# version of DROTG
    //
    // DROTG construct gives plane rotation.

    // Local Scalars
    double r, roe, scale, z;

```

```

    roe = db;
    if (Math.Abs(da) > Math.Abs(db))
    {
        roe = da;
    }
    scale = Math.Abs(da) + Math.Abs(db);
    if (scale == 0.0)
    {
        c = 1.0;
        s = 0.0;
        r = 0.0;
        z = 0.0;
    }
    else
    {
        r = scale * Math.Sqrt(Math.Pow(da / scale, 2) + Math.Pow(db / scale, 2));
        //condition ? first_expression : second_expression
        r = (roe >= 0.0 ? 1.0 : -1.0) * r;
        c = da / r;
        s = db / r;
        z = 1.0;
        if (Math.Abs(da) > Math.Abs(db))
        {
            z = s;
        }
        if ((Math.Abs(db) >= Math.Abs(da)) & (c != 0.0))
        {
            z = 1.0 / c;
        }
    }
    da = r;
    db = z;
}
public static void DROT(int n, ref double[] dx, int incx, ref double[] dy, int incy, double c, double s)
{
    // C# version of DROT
    //
    // DROT applies a plane rotation

    // Local Scalars
    double dtemp;
    int i, ix, iy;

    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        for (i = 1; i <= n; i = i + 1)
        {
            dtemp = c * dx[i - 1] + s * dy[i - 1];
            dy[i - 1] = c * dy[i - 1] - s * dx[i - 1];
            dx[i - 1] = dtemp;
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dtemp = c * dx[ix - 1] + s * dy[iy - 1];
            dy[iy - 1] = c * dy[iy - 1] - s * dx[ix - 1];
            dx[ix - 1] = dtemp;
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}
public static void _DCOPY(int n, double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DCOPY
    //
    // DCOPY copies a vector, x, to a vector, y uses unrolled loops for increments equal to one

    // Local Scalars
    int i, ix, iy, m, mpl;
    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        // clean-up loop
        m = n % 7;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dy[i - 1] = dx[i - 1];
            }
            if (n < 7)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 7)
        {
            dy[i - 1] = dx[i - 1];
            dy[i] = dx[i];

```

```

        dy[i + 1] = dx[i + 1];
        dy[i + 2] = dx[i + 2];
        dy[i + 3] = dx[i + 3];
        dy[i + 4] = dx[i + 4];
        dy[i + 5] = dx[i + 5];
    }
}
else
{
    // code for unequal increments or equal increments not equal to 1
    ix = 1;
    iy = 1;
    if (incx < 0)
    {
        ix = (-n + 1) * incx + 1;
    }
    if (incy < 0)
    {
        iy = (-n + 1) * incy + 1;
    }
    for (i = 1; i <= n; i = i + 1)
    {
        dy[iy - 1] = dx[ix - 1];
        ix = ix + incx;
        iy = iy + incy;
    }
}
}
public static void ZCOPY(int n, Complex[] zx, int incx, ref Complex[] zy, int incy)
{
    // C# version of ZCOPY
    //
    // DCOPY copies a vector, x, to a vector, y.

    // Local Scalars
    int i, ix, iy;
    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            zy[i - 1] = zx[i - 1];
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            zy[iy - 1] = zx[ix - 1];
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}
}
public static void _DSWAP(int n, ref double[] dx, int incx, ref double[] dy, int incy)
{
    // C# version of DSWAP
    //
    // interchanges two vectors.  uses unrolled loops for increments equal one.

    // Local Scalars
    double dtemp;
    int i, ix, iy, m, mpl;

    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        //clean-up loop
        m = n % 3;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dtemp = dx[i - 1];
                dx[i - 1] = dy[i - 1];
                dy[i - 1] = dtemp;
            }
            if (n < 3)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 3)
        {
            dtemp = dx[i - 1];
            dx[i - 1] = dy[i - 1];
            dy[i - 1] = dtemp;

            dtemp = dx[i];
            dx[i] = dy[i];

```

```

        dy[i] = dtemp;

        dtemp = dx[i + 1];
        dx[i + 1] = dy[i + 1];
        dy[i + 1] = dtemp;
    }
}
else
{
    // code for unequal increments or equal increments not equal to 1
    ix = 1;
    iy = 1;
    if (incx < 0)
    {
        ix = (-n + 1) * incx + 1;
    }
    if (incy < 0)
    {
        iy = (-n + 1) * incy + 1;
    }
    for (i = 1; i <= n; i = i + 1)
    {
        dtemp = dx[ix - 1];
        dx[ix - 1] = dy[iy - 1];
        dy[iy - 1] = dtemp;
        ix = ix + incx;
        iy = iy + incy;
    }
}
}

public static void ZSWAP(int n, ref Complex[] zx, int incx, ref Complex[] zy, int incy)
{
    // C# version of ZSWAP
    //
    // ZSWAP interchanges two vectors.

    // Local Scalars
    Complex ztemp;
    int i, ix, iy;

    if (n <= 0)
    {
        return;
    }
    if ((incx == 1) & (incy == 1))
    {
        // code for both increments equal to 1
        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = zx[i - 1];
            zx[i - 1] = zy[i - 1];
            zy[i - 1] = ztemp;
        }
    }
    else
    {
        // code for unequal increments or equal increments not equal to 1
        ix = 1;
        iy = 1;
        if (incx < 0)
        {
            ix = (-n + 1) * incx + 1;
        }
        if (incy < 0)
        {
            iy = (-n + 1) * incy + 1;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            ztemp = zx[ix - 1];
            zx[ix - 1] = zy[iy - 1];
            zy[iy - 1] = ztemp;
            ix = ix + incx;
            iy = iy + incy;
        }
    }
}

}

public static double _DNRM2(int n, double[] x, int incx)
{
    // C# version of DNRM2
    //
    // DNRM2 returns the euclidean norm of a vector via the function name, so that DNRM2 := sqrt( x'*x )

    // Parameters
    const double one = 1.0, zero = 0.0;

    // Local Scalars
    double absxi, norm, scale, ssq;
    int ix;

    if ((n < 1) | (incx < 1))
    {
        return zero;
    }
    else if (n == 1)
    {
        return Math.Abs(x[0]);
    }
    else
    {
        scale = zero;
        ssq = one;
        // The following loop is equivalent to this call to the LAPACK auxiliary routine:
        // CALL DLAASSQ( N, X, INCX, SCALE, SSQ )

        for (ix = 1; ix <= 1 + (n - 1) * incx; ix = ix + incx)
        {
            if (x[ix - 1] != zero)
            {
                absxi = Math.Abs(x[ix - 1]);
                if (scale < absxi)
                {

```

```

        ssq = one + ssq * Math.Pow(scale / absxi, 2);
        scale = absxi;
    }
    else
    {
        ssq = ssq + Math.Pow(absxi / scale, 2);
    }
}
return scale * Math.Sqrt(ssq);
}
}
public static double DZNRM2(int n, Complex[] x, int incx)
{
    // C# version of DZNRM2
    //
    // DZNRM2 returns the euclidean norm of a vector via the function name, so that DZNRM2 := sqrt( x**H*x )
    //
    // Parameters
    const double one = 1.0, zero = 0.0;
    //
    // Local Scalars
    //double absxi, norm, scale, ssq;
    double norm, scale, ssq, temp;
    int ix;
    //Console.WriteLine("DZNRM2===IN");
    if ((n < 1) | (incx < 1))
    {
        return zero;
    }
    /*else if (n == 1)
    {
        return Math.Abs(x[0]);
    }*/
    else
    {
        scale = zero;
        ssq = one;
        // The following loop is equivalent to this call to the LAPACK auxiliary routine:
        // CALL ZLASSQ( N, X, INCX, SCALE, SSQ )
        for (ix = 1; ix <= 1 + (n - 1) * incx; ix = ix + incx)
        {
            if (x[ix - 1].Real != zero)
            {
                // Console.WriteLine("xinr {0}", x[ix-1]);
                //absxi = Math.Abs(x[ix - 1]);
                temp = Math.Abs(x[ix - 1].Real);
                //if (scale < absxi)
                if (scale < temp)
                {
                    //ssq = one + ssq * Math.Pow(scale / absxi, 2);
                    ssq = one + ssq * Math.Pow(scale / temp, 2);
                    //scale = absxi;
                    scale = temp;
                }
                else
                {
                    //ssq = ssq + Math.Pow(absxi / scale, 2);
                    ssq = ssq + Math.Pow(temp / scale, 2);
                }
            }
            //Console.WriteLine("R {0}", ssq);
            if (x[ix - 1].Imaginary != zero)
            {
                // Console.WriteLine("xini {0}", x[ix - 1]);
                temp = Math.Abs(x[ix - 1].Imaginary);
                if (scale < temp)
                {
                    ssq = one + ssq * Math.Pow(scale / temp, 2);
                    scale = temp;
                }
                else
                {
                    ssq = ssq + Math.Pow(temp / scale, 2);
                }
            }
            //Console.WriteLine("I {0}", ssq);
        }
        // Console.WriteLine("DZNRM2===OUT");
        return scale * Math.Sqrt(ssq);
    }
}
}
public static double DCABS1(Complex z)
{
    // C# version of DCABS1
    //
    // DCABS1 computes absolute value of a double complex number
    return Math.Abs(z.Real) + Math.Abs(z.Imaginary);
}
public static double _DASUM(int n, double[] dx, int incx)
{
    // C# version of DASUM
    //
    // DASUM takes the sum of the absolute values
    //
    // Local Scalars
    double dtemp;
    int i, m, mpi, nincx;
    //DASUM = 0.0;
    dtemp = 0.0;
    if ((n <= 0) | (incx <= 0))
    {
        return 0.0;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        // clean-up loop
        m = n % 6;
        if (m != 0)
    }

```

```

        {
            for (i = 1; i <= m; i = i + 1)
            {
                dtemp = dtemp + Math.Abs(dx[i - 1]);
            }
            if (n < 6)
            {
                return dtemp;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 6)
        {
            dtemp = dtemp + Math.Abs(dx[i - 1]) + Math.Abs(dx[i]) + Math.Abs(dx[i + 1]) + Math.Abs(dx[i + 2]) + Math.Abs(dx[i + 3]) + Math.Abs(dx[i
+ 4]);
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            dtemp = dtemp + Math.Abs(dx[i - 1]);
        }
    }
    return dtemp;
}
public static double DZASUM(int n, Complex[] zx, int incx)
{
    // C# version of DZASUM
    //
    // DZASUM takes the sum of the absolute values

    // Local Scalars
    //double dtemp;
    double stemp;
    //int i, m, mpl, nincx;
    int i, nincx;

    //DASUM = 0.0;
    stemp = 0.0;
    if ((n <= 0) | (incx <= 0))
    {
        return 0.0;
    }
    if (incx == 1)
    {
        // code for increment equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            stemp = stemp + DCABS1(zx[i - 1]);
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            stemp = stemp + DCABS1(zx[i - 1]);
        }
    }
    return stemp;
}
public static void _DSCAL(int n, double da, ref double[] dx, int incx)
{
    // C# version of DSCAL
    //
    // DSCAL scales a vector by a constant. uses unrolled loops for increment equal to one.

    // Local Scalars
    int i, m, mpl, nincx;
    if ((n <= 0) | (incx <= 0))
    {
        return;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        // clean-up loop

        m = n % 5;
        if (m != 0)
        {
            for (i = 1; i <= m; i = i + 1)
            {
                dx[i - 1] = da * dx[i - 1];
            }
            if (n < 5)
            {
                return;
            }
        }
        mpl = m + 1;
        for (i = mpl; i <= n; i = i + 5)
        {
            dx[i - 1] = da * dx[i - 1];
            dx[i] = da * dx[i];
            dx[i + 1] = da * dx[i + 1];
            dx[i + 2] = da * dx[i + 2];
            dx[i + 3] = da * dx[i + 3];
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            dx[i - 1] = da * dx[i - 1];
        }
    }
}

```

```

    }
}
public static void ZSCAL(int n, Complex za, ref Complex[] zx, int incx)
{
    // C# version of ZSCAL
    //
    // ZSCAL scales a vector by a constant. uses unrolled loops for increment equal to one.

    // Local Scalars
    int i, nincx;
    if ((n <= 0) | (incx <= 0))
    {
        return;
    }
    if (incx == 1)
    {
        // code for increment equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            zx[i - 1] = za * zx[i - 1];
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            zx[i - 1] = za * zx[i - 1];
        }
    }
}

public static void ZDSCAL(int n, double da, ref Complex[] zx, int incx)
{
    // C# version of ZDSCAL
    //
    // ZDSCAL scales a vector by a constant.

    // Local Scalars
    int i, nincx;
    if ((n <= 0) | (incx <= 0))
    {
        return;
    }
    if (incx == 1)
    {
        // code for increment equal to 1

        for (i = 1; i <= n; i = i + 1)
        {
            zx[i - 1] = new Complex(da, 0.0) * zx[i - 1];
        }
    }
    else
    {
        // code for increment not equal to 1
        nincx = n * incx;
        for (i = 1; i <= nincx; i = i + incx)
        {
            zx[i - 1] = new Complex(da, 0.0) * zx[i - 1];
        }
    }
}

public static int _IDAMAX(int n, double[] dx, int incx)
{
    // C# version of IDAMAX
    //
    // IDAMAX finds the index of element having max. absolute value.

    // Local Scalars
    double dmax;
    int i, ix, idamax;

    idamax = 0;
    if ((n < 1) | (incx <= 0))
    {
        return idamax;
    }
    idamax = 1;

    if (n == 1)
    {
        return idamax;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        dmax = Math.Abs(dx[0]);
        for (i = 2; i <= n; i = i + 1)
        {
            if (Math.Abs(dx[i - 1]) > dmax)
            {
                idamax = i;
                dmax = Math.Abs(dx[i - 1]);
            }
        }
    }
    else
    {
        // code for increment not equal to 1
        ix = 1;
        dmax = Math.Abs(dx[0]);
        ix = ix + incx;
        for (i = 2; i <= n; i = i + 1)
        {
            if (Math.Abs(dx[ix - 1]) > dmax)
            {
                idamax = i;
                dmax = Math.Abs(dx[ix - 1]);
            }
            ix = ix + incx;
        }
    }
}

```

```

    }
    return idamax;
}
public static int IZAMAX(int n, Complex[] zx, int incx)
{
    // C# version of IZAMAX
    //
    // IZAMAX finds the index of element having max. absolute value.

    // Local Scalars
    double dmax;
    int i, ix, izamax;

    izamax = 0;
    if ((n < 1) | (incx <= 0))
    {
        return izamax;
    }
    izamax = 1;

    if (n == 1)
    {
        return izamax;
    }
    if (incx == 1)
    {
        // code for increment equal to 1
        dmax = DCABS1(zx[0]);
        for (i = 2; i <= n; i = i + 1)
        {
            if (DCABS1(zx[i - 1]) > dmax)
            {
                izamax = i;
                dmax = DCABS1(zx[i - 1]);
            }
        }
    }
    else
    {
        // code for increment not equal to 1
        ix = 1;
        dmax = DCABS1(zx[0]);
        ix = ix + incx;
        for (i = 2; i <= n; i = i + 1)
        {
            if (DCABS1(zx[ix - 1]) > dmax)
            {
                izamax = i;
                dmax = DCABS1(zx[ix - 1]);
            }
            ix = ix + incx;
        }
    }
    return izamax;
}

public static void DROTMG(ref double dd1, ref double dd2, ref double dx1, ref double dyl, ref double[] dparam)
{
    // C# version of DROTMG
    //
    // CONSTRUCT THE MODIFIED GIVENS TRANSFORMATION MATRIX H WHICH ZEROS
    // THE SECOND COMPONENT OF THE 2-VECTOR (DSQRT(DD1)*DX1,DSQRT(DD2))*> DY2)**T.
    // WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..

    // DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0

    // (DH11  DH12)      (1.D0  DH12)      (DH11  1.D0)      (1.D0  0.D0)
    // H=(      )      (      )      (      )      (      )
    // (DH21  DH22), (DH21  1.D0), (-1.D0 DH22), (0.D0  1.D0).

    // LOCATIONS 2-4 OF DPARAM CONTAIN DH11, DH21, DH12, AND DH22
    // RESPECTIVELY. (VALUES OF 1.D0, -1.D0, OR 0.D0 IMPLIED BY THE
    // VALUE OF DPARAM(1) ARE NOT STORED IN DPARAM.)

    // THE VALUES OF GAMSQ AND RGAMSQ SET IN THE DATA STATEMENT MAY BE
    // INEXACT. THIS IS OK AS THEY ARE ONLY USED FOR TESTING THE SIZE
    // OF DD1 AND DD2. ALL ACTUAL SCALING OF DATA IS DONE USING GAM.

    // DPARAM(1)=DFLAG
    // DPARAM(2)=DH11
    // DPARAM(3)=DH21
    // DPARAM(4)=DH12
    // DPARAM(5)=DH22

    // Local Scalars
    double dflag = 0.0, dh11 = 0.0, dh12 = 0.0, dh21 = 0.0, dh22 = 0.0, dp1, dp2, dq1, dq2, dtemp;
    double du, gam = 4096.0, gamsq = 16777216.0, one = 1.0, rgamsq = 5.9604645e-8, two = 2.0, zero = 0.0;

    if (dd1 < zero)
    {
        // GO ZERO-H-D-AND-DX1
        dflag = -one;
        dh11 = zero;
        dh12 = zero;
        dh21 = zero;
        dh22 = zero;

        dd1 = zero;
        dd2 = zero;
        dx1 = zero;
    }
    else
    {
        // CASE-DD1-NONNEGATIVE
        dp2 = dd2 * dyl;
        if (dp2 == zero)
        {
            dflag = -two;
            dparam[0] = dflag;
            return;
        }
        // REGULAR-CASE
        dp1 = dd1 * dx1;
        dq2 = dp2 * dyl;
    }
}

```

```

dq1 = dp1 * dx1;
if (Math.Abs(dq1) > Math.Abs(dq2))
{
    dh21 = -dy1 / dx1;
    dh12 = dp2 / dp1;

    du = one - dh12 * dh21;

    if (du > zero)
    {
        dflag = zero;
        dd1 = dd1 / du;
        dd2 = dd2 / du;
        dx1 = dx1 * du;
    }
}
else
{
    if (dq2 < zero)
    {
        // GO ZERO-H-D-AND-DX1
        dflag = -one;
        dh11 = zero;
        dh12 = zero;
        dh21 = zero;
        dh22 = zero;

        dd1 = zero;
        dd2 = zero;
        dx1 = zero;
    }
    else
    {
        dflag = one;
        dh11 = dp1 / dp2;
        dh22 = dx1 / dy1;
        du = one + dh11 * dh22;
        dtemp = dd2 / du;
        dd2 = dd1 / du;
        dd1 = dtemp;
        dx1 = dy1 * du;
    }
}
// PROCEDURE..SCALE-CHECK
if (dd1 != zero)
{
    while ((dd1 <= rgamsq) | (dd1 >= gamsq))
    {
        if (dflag == zero)
        {
            dh11 = one;
            dh22 = one;
            dflag = -one;
        }
        else
        {
            dh21 = -one;
            dh12 = one;
            dflag = -one;
        }
        if (dd1 <= rgamsq)
        {
            dd1 = dd1 * Math.Pow(gam, 2);
            dx1 = dx1 / gam;
            dh11 = dh11 / gam;
            dh12 = dh12 / gam;
        }
        else
        {
            dd1 = dd1 / Math.Pow(gam, 2);
            dx1 = dx1 * gam;
            dh11 = dh11 * gam;
            dh12 = dh12 * gam;
        }
    }
}
if (dd2 != zero)
{
    while ((Math.Abs(dd2) <= rgamsq) | (Math.Abs(dd2) >= gamsq))
    {
        if (dflag == zero)
        {
            dh11 = one;
            dh22 = one;
            dflag = -one;
        }
        else
        {
            dh21 = -one;
            dh12 = one;
            dflag = -one;
        }
        if (Math.Abs(dd2) <= rgamsq)
        {
            dd2 = dd2 * Math.Pow(gam, 2);
            dh21 = dh21 / gam;
            dh22 = dh22 / gam;
        }
        else
        {
            dd2 = dd2 / Math.Pow(gam, 2);
            dh21 = dh21 * gam;
            dh22 = dh22 * gam;
        }
    }
}
}
if (dflag < zero)
{
    dparam[1] = dh11;
    dparam[2] = dh21;
    dparam[3] = dh12;
    dparam[4] = dh22;
}

```

```

else if (dflag == zero)
{
    dparam[2] = dh21;
    dparam[3] = dh12;
}
else
{
    dparam[1] = dh11;
    dparam[4] = dh22;
}
dparam[0] = dflag;
}
public static void DROTM(int n, ref double[] dx, int incx, ref double[] dy, int incy, double[] dparam)
{
    // C# version of DROTM
    //
    // APPLY THE MODIFIED GIVENS TRANSFORMATION, H, TO THE 2 BY N MATRIX
    //
    // (DX**T) , WHERE **T INDICATES TRANSPOSE. THE ELEMENTS OF DX ARE IN
    // (DY**T)
    //
    // DX(LX+I*INCX), I = 0 TO N-1, WHERE LX = 1 IF INCX .GE. 0, ELSE
    // LX = (-INCX)*N, AND SIMILARLY FOR SY USING LY AND INCY.
    // WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..
    //
    // DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0
    //
    // (DH11 DH12)      (1.D0 DH12)      (DH11 1.D0)      (1.D0 0.D0)
    // H= ( )            ( )            ( )            ( )
    // (DH21 DH22), (DH21 1.D0), (-1.D0 DH22), (0.D0 1.D0).
    // SEE DROTMG FOR A DESCRIPTION OF DATA STORAGE IN DPARAM.
    //
    // DPARAM(1)=DFLAG
    // DPARAM(2)=DH11
    // DPARAM(3)=DH21
    // DPARAM(4)=DH12
    // DPARAM(5)=DH22
    //
    // Local Scalars
    double dflag, dh11, dh12, dh21, dh22, two = 2.0, w, z, zero = 0.0;
    int i, kx, ky, nsteps;

    dflag = dparam[0];
    if ((n <= 0) | (dflag + two == zero))
    {
        return;
    }
    if ((incx == incy) & (incx > 0))
    {
        nsteps = n * incx;
        if (dflag < zero)
        {
            {
                dh11 = dparam[1];
                dh12 = dparam[3];
                dh21 = dparam[2];
                dh22 = dparam[4];
                for (i = 1; i <= nsteps; i = i + incx)
                {
                    w = dx[i - 1];
                    z = dy[i - 1];
                    dx[i - 1] = w * dh11 + z * dh12;
                    dy[i - 1] = w * dh21 + z * dh22;
                }
            }
            else if (dflag == zero)
            {
                dh12 = dparam[3];
                dh21 = dparam[2];
                for (i = 1; i <= nsteps; i = i + incx)
                {
                    w = dx[i - 1];
                    z = dy[i - 1];
                    dx[i - 1] = w + z * dh12;
                    dy[i - 1] = w * dh21 + z;
                }
            }
            else
            {
                dh11 = dparam[1];
                dh22 = dparam[4];
                for (i = 1; i <= nsteps; i = i + incx)
                {
                    w = dx[i - 1];
                    z = dy[i - 1];
                    dx[i - 1] = w * dh11 + z;
                    dy[i - 1] = -w + dh22 * z;
                }
            }
        }
    }
    else
    {
        kx = 1;
        ky = 1;
        if (incx < 0)
        {
            kx = 1 + (1 - n) * incx;
        }
        if (incy < 0)
        {
            ky = 1 + (1 - n) * incy;
        }
        if (dflag < zero)
        {
            dh11 = dparam[1];
            dh12 = dparam[3];
            dh21 = dparam[2];
            dh22 = dparam[4];
            for (i = 1; i <= n; i = i + 1)
            {
                w = dx[kx - 1];
                z = dy[ky - 1];
                dx[kx - 1] = w * dh11 + z * dh12;
                dy[ky - 1] = w * dh21 + z * dh22;
                kx = kx + incx;
            }
        }
    }
}

```

```

        ky = ky + incy;
    }
}
else if (dflag == zero)
{
    dh12 = dparam[3];
    dh21 = dparam[2];
    for (i = 1; i <= n; i = i + 1)
    {
        w = dx[kx - 1];
        z = dy[ky - 1];
        dx[kx - 1] = w + z * dh12;
        dy[ky - 1] = w * dh21 + z;
        kx = kx + incx;
        ky = ky + incy;
    }
}
else
{
    dh11 = dparam[1];
    dh22 = dparam[4];
    for (i = 1; i <= n; i = i + 1)
    {
        w = dx[kx - 1];
        z = dy[ky - 1];
        dx[kx - 1] = w * dh11 + z;
        dy[ky - 1] = -w + dh22 * z;
        kx = kx + incx;
        ky = ky + incy;
    }
}
}
}
public static double DSDOT(int n, float[] sx, int incx, float[] sy, int incy)
{
    // C# version of DSDOT
    //
    // Compute the inner product of two vectors with extended precision accumulation and result.

    // Returns D.P. dot product accumulated in D.P., for S.P. SX and SY
    // DSDOT = sum for I = 0 to N-1 of SX(LX+I*INCX) * SY(LY+I*INCY),
    // where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
    // defined in a similar way using INCY.

    // Local Scalars
    int i, kx, ky, ns;
    double dsdot;

    dsdot = 0.0;
    if (n <= 0)
    {
        return dsdot;
    }
    if ((incx == incy) & (incx > 0))
    {
        // Code for equal, positive, non-unit increments.
        ns = n * incx;
        for (i = 1; i <= ns; i = i + incx)
        {
            dsdot = dsdot + (double)sx[i - 1] * (double)sy[i - 1];
        }
    }
    else
    {
        // Code for unequal or nonpositive increments.
        kx = 1;
        ky = 1;
        if (incx < 0)
        {
            kx = 1 + (1 - n) * incx;
        }
        if (incy < 0)
        {
            ky = 1 + (1 - n) * incy;
        }
        for (i = 1; i <= n; i = i + 1)
        {
            dsdot = dsdot + (double)sx[kx - 1] * (double)sy[ky - 1];
            kx = kx + incx;
            ky = ky + incy;
        }
    }
    return dsdot;
}

public static void ZBLAT2() { /*???STEST uses misc_d.eps, DLAMCH/ first
{
    /*
    // C# version of Test program for the COMPLEX*16 Level 2 Blas.

    /*
    The program must be driven by a short data file. The first 18 records
    of the file are read using list-directed input, the last 17 records
    are read using the format ( A6, L2 ). An annotated example of a data
    file can be obtained by deleting the first 3 characters from the following 35 lines:
    'zblat2.out'      NAME OF SUMMARY OUTPUT FILE
    6                UNIT NUMBER OF SUMMARY FILE
    'CBLA2T.SNAP'    NAME OF SNAPSHOT OUTPUT FILE
    -1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
    F              LOGICAL FLAG, T TO REMIND SNAPSHOT FILE AFTER EACH RECORD.
    F              LOGICAL FLAG, T TO STOP ON FAILURES.
    T              LOGICAL FLAG, T TO TEST ERROR EXITS.
    16.0           THRESHOLD VALUE OF TEST RATIO
    6              NUMBER OF VALUES OF N
    0 1 2 3 5 9   VALUES OF N
    4              NUMBER OF VALUES OF K
    0 1 2 4       VALUES OF K
    4              NUMBER OF VALUES OF INCX AND INCY
    1 2 -1 -2     VALUES OF INCX AND INCY
    3              NUMBER OF VALUES OF ALPHA
    (0.0,0.0) (1.0,0.0) (0.7,-0.9) VALUES OF ALPHA
    3              NUMBER OF VALUES OF BETA
    (0.0,0.0) (1.0,0.0) (1.3,-1.1) VALUES OF BETA
    ZGEMV T PUT F FOR NO TEST. SAME COLUMNS.

```

```

ZGBMV T PUT F FOR NO TEST. SAME COLUMNS.
ZHEMV T PUT F FOR NO TEST. SAME COLUMNS.
ZHBMV T PUT F FOR NO TEST. SAME COLUMNS.
ZHPMV T PUT F FOR NO TEST. SAME COLUMNS.
ZTRMV T PUT F FOR NO TEST. SAME COLUMNS.
ZTBMV T PUT F FOR NO TEST. SAME COLUMNS.
ZTPMV T PUT F FOR NO TEST. SAME COLUMNS.
ZTRSV T PUT F FOR NO TEST. SAME COLUMNS.
ZTBSV T PUT F FOR NO TEST. SAME COLUMNS.
ZTPSV T PUT F FOR NO TEST. SAME COLUMNS.
ZGERC T PUT F FOR NO TEST. SAME COLUMNS.
ZGERU T PUT F FOR NO TEST. SAME COLUMNS.
ZHER T PUT F FOR NO TEST. SAME COLUMNS.
ZHR T PUT F FOR NO TEST. SAME COLUMNS.
ZHER2 T PUT F FOR NO TEST. SAME COLUMNS.
ZHPR2 T PUT F FOR NO TEST. SAME COLUMNS.
*/

// Parameters
//const int nin = 5;
const int nsubs = 17;
Complex zero = new Complex(0.0, 0.0);
Complex one = new Complex(1.0, 0.0);
const double rzero = 0.0;

const int nmax = 65;
const int incmax = 2;

const int ninmax = 7;
const int nidmax = 9;
const int nkbmax = 7;
const int nalmx = 7;
const int nbemx = 7;

// Local Scalars
double eps, err, thresh;
int n, nalf, nbet, nidim, ninc, nkb, nout, ntra;
//i, isnum, j, n, nalf, nbet, nidim, ninc, nkb, nout, ntra;
bool fatal = false, ltestt, rewi, same, sfatal, trace, tstderr;

string trans, snamet, snaps, summy;
/* CHARACTER*1 TRANS
CHARACTER*6 SNAMET
CHARACTER*32 SNAPS, SUMMY
*/

// Local Arrays
Complex[,] a = new Complex[nmax, nmax];
Complex[] aa = new Complex[nmax * nmax];
Complex[] alf = new Complex[nalmx];
Complex[] _as = new Complex[nmax * nmax];
Complex[] Bet = new Complex[nbemx];
Complex[] x = new Complex[nmax];
Complex[] xs = new Complex[nmax * incmax];
Complex[] xx = new Complex[nmax * incmax];
Complex[] y = new Complex[nmax];
Complex[] ys = new Complex[nmax * incmax];
Complex[] yt = new Complex[nmax];
Complex[] yy = new Complex[nmax * incmax];
Complex[] z = new Complex[2 * nmax];

double[] g = new double[nmax];
int[] idim = new int[nidmax];
int[] inc = new int[ninmax];
int[] kb = new int[nkbmax];

bool[] ltest = new bool[nsubs];
string[] snames = {
    "ZGEMV ",
    "ZGBMV ",
    "ZHEMV ",
    "ZHBMV ",
    "ZHPMV ",
    "ZTRMV ",
    "ZTBMV ",
    "ZTPMV ",
    "ZTRSV ",
    "ZTBSV ",
    "ZTPSV ",
    "ZGERC ",
    "ZGERU ",
    "ZHER ",
    "ZHR ",
    "ZHER2 ",
    "ZHPR2 " };

// file I/O
//String fin = @".\dbl3.in";
String fin = @"STDIN";
ArrayList tmp = new ArrayList();
String[] findata = { };
String[] stmp_ld;
String stmp;

StreamWriter fout1, fout2;
err = 0; //added
/* External Functions
DOUBLE PRECISION DDIFF
LOGICAL LZE
EXTERNAL DDIFF, LZE
*/

/* .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
CHARACTER*6 SRNAMT
* .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
COMMON /SRNAMC/SRNAMT
*/

// Executable Statements
// read input file
try

```

```

{
    tmp.Clear();
    //using (StreamReader infile = new StreamReader(fin))
    using (StreamReader infile = new StreamReader(Console.OpenStandardInput()))
    {
        string line;
        int i = 0;
        while ((line = infile.ReadLine()) != null)
        {
            tmp.Add(line.Trim());
            Console.WriteLine("Read:{0}", line.Trim());
        }
    }
    findata = (String[])tmp.ToArray(typeof(string));
}
catch (Exception e)
{
    Console.WriteLine("Error Reading {0}", fin);
    Environment.Exit(1);
}

// Read name and unit number for summary output file and open file.
summy = findata[0].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
nout = int.Parse(findata[1].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

fout1 = new StreamWriter(summy, false); // true = append
//misc_d.nout = fout1; // for XERBLA
misc_z.nout = fout1; // for XERBLA
fout2 = null;
noutc = nout;

// Read name and unit number for snapshot output file and open file.
snaps = findata[2].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
ntra = int.Parse(findata[3].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

trace = (ntra >= 0);

//Console.WriteLine("ntra: {0}, trace: {1}", ntra, trace);
if (trace)
{
    fout2 = new StreamWriter(snaps, true); // true = append
}

// Read the flag that directs rewinding of the snapshot file.
//Console.WriteLine("4052: {0}", findata[4]);
//Console.WriteLine("4052: {0}", findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
//rewi = bool.Parse(findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    rewi = true;
}
else
{
    rewi = false;
}
//Console.WriteLine("4052: {0}", rewi);
rewi = rewi & trace;

// Read the flag that directs stopping on any failure.
//sfatal = bool.Parse(findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    sfatal = true;
}
else
{
    sfatal = false;
}

// Read the flag that indicates whether error exits are to be tested.
//tsterr = bool.Parse(findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    tsterr = true;
}
else
{
    tsterr = false;
}

// Read the threshold value of the test ratio
thresh = double.Parse(findata[7].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

// Read and check the parameter values for the tests.
// Values of N
nidim = int.Parse(findata[8].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

if ((nidim < 1) || nidim > nidmax)
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "N", nidmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4095:");
    Environment.Exit(1);
}

for (int i = 1; i <= nidim; i = i + 1)
{
    idim[i - 1] = int.Parse(findata[9].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((idim[i - 1] < 0) || (idim[i - 1] > nmax))
    {
        fout1.WriteLine("VALUE OF N IS LESS THAN 0 OR GREATER THAN {0,2:D}", nmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
    }
}

```

```

    }
    Console.WriteLine("4111:");
    Environment.Exit(1);
}

// Values of K
nkb = int.Parse(findata[10].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nkb < 1) || (nkb > nkbmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "K", nkbmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4127:");
    Environment.Exit(1);
}

for (int i = 1; i <= nkb; i = i + 1)
{
    kb[i - 1] = int.Parse(findata[11].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if (kb[i - 1] < 0)
    {
        fout1.WriteLine("VALUE OF K IS LESS THAN 0");
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4143:");
        Environment.Exit(1);
    }
}

// Values of INCX and INCY
ninc = int.Parse(findata[12].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((ninc < 1) || (ninc > nincmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "INCX AND INCY", nincmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4159:");
    Environment.Exit(1);
}

for (int i = 1; i <= ninc; i = i + 1)
{
    inc[i - 1] = int.Parse(findata[13].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((inc[i - 1] == 0) || Math.Abs(inc[i - 1]) > incmax)
    {
        fout1.WriteLine("ABSOLUTE VALUE OF INCX OR INCY IS 0 OR GREATER THAN {0,2:D}", incmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4175:");
        Environment.Exit(1);
    }
}

// Values of ALPHA
nalf = int.Parse(findata[14].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nalf < 1) || (nalf > nalfmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "ALPHA", nalfmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4191:");
    Environment.Exit(1);
}

for (int i = 1; i <= nalf; i = i + 1)
{
    //alf[i - 1] = Complex.Parse(findata[15].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    stmp = findata[15].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1];
    //Console.WriteLine("{0}", stmp);
    stmp_ld = stmp.Substring(1, stmp.Length - 2).Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
    //Console.WriteLine("{0} {1}", stmp_ld[0], stmp_ld[1]);
    alf[i - 1] = new Complex(double.Parse(stmp_ld[0]), double.Parse(stmp_ld[1]));
    //Console.WriteLine("{0}", alf[i - 1]);
}
//Environment.Exit(1);
// Values of BETA
nbt = int.Parse(findata[16].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nbt < 1) || (nbt > nbmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "BETA", nbmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4210:");
    Environment.Exit(1);
}

for (int i = 1; i <= nbt; i = i + 1)
{
    //bet[i - 1] = double.Parse(findata[17].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);

```

```

        stmp = findata[17].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1];
        stmp_id = stmp.Substring(1, stmp.Length - 2).Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
        bet[i - 1] = new Complex(double.Parse(stmp_id[0]), double.Parse(stmp_id[1]));
        //Console.WriteLine("{0}", bet[i - 1]);
    }

    // Report values of parameters
    foutl.WriteLine("TESTS OF THE COMPLEX LEVEL 2 BLAS \r\n\r\n THE FOLLOWING PARAMETER VALUES WILL BE USED:");
    foutl.WriteLine("  FOR N");
    for (int i = 1; i <= nldim; i = i + 1)
    {
        foutl.WriteLine("{0,6:D}", idim[i - 1]);
    }
    foutl.WriteLine();

    foutl.WriteLine("  FOR K");
    for (int i = 1; i <= nkb; i = i + 1)
    {
        foutl.WriteLine("{0,6:D}", kb[i - 1]);
    }
    foutl.WriteLine();

    foutl.WriteLine("  FOR INCX AND INCY");
    for (int i = 1; i <= ninc; i = i + 1)
    {
        foutl.WriteLine("{0,6:D}", inc[i - 1]);
    }
    foutl.WriteLine();

    foutl.WriteLine("  FOR ALPHA");
    for (int i = 1; i <= nalf; i = i + 1)
    {
        foutl.WriteLine("{0}", alf[i - 1].ToString("F1"));
        //7( '(', F4.1, ',', F4.1, ') ', : )
    }
    foutl.WriteLine();

    foutl.WriteLine("  FOR BETA");
    for (int i = 1; i <= nbet; i = i + 1)
    {
        foutl.WriteLine("{0}", bet[i - 1].ToString("F1"));
        //7( '(', F4.1, ',', F4.1, ') ', : )
    }
    foutl.WriteLine();
    if (!tsterr)
    {
        foutl.WriteLine();
        foutl.WriteLine("ERROR-EXITS WILL NOT BE TESTED");
    }
    foutl.WriteLine();
    foutl.WriteLine("ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN {0,8:F2}", thresh);
    foutl.WriteLine();

    // Read names of subroutines and flags which indicate whether they are to be tested.
    for (int i = 1; i <= nsubs; i = i + 1)
    {
        ltest[i - 1] = false;
    }
    for (int i = 1; i <= nsubs; i = i + 1)
    {
        sname = findata[18 + i - 1].Substring(0, 6);
        //ltestt = bool.Parse(findata[18 + i - 1].Substring(6, 2).Trim());
        if (findata[18 + i - 1].Substring(6, 2).Trim().ToUpper() == "T")
        {
            ltestt = true;
        }
        else
        {
            ltestt = false;
        }
        if (sname == snames[i - 1])
        {
            ltest[i - 1] = ltestt;
        }
        else
        {
            foutl.WriteLine("SUBPROGRAM NAME {0} NOT RECOGNIZED\n***** TESTS ABANDONED *****", sname);
            foutl.Flush();
            foutl.Close();
            Console.WriteLine("4276:");
            Environment.Exit(1);
        }
    }

    //foutl.Flush();
    //foutl.Close();
    //Console.WriteLine("4282:");
    // Compute EPS (the machine precision).
    // uses misc_d.eps, call DLAMCH first
    //eps = misc_d.eps;
    eps = misc_d.prec; // EPS = EPSILON(RZERO)
    foutl.WriteLine("RELATIVE MACHINE PRECISION IS TAKEN TO BE {0,9:E1}", eps);
    //Console.WriteLine("{0} {1} {0,9:E1}", eps, eps * 10, eps * 10);
    //Console.WriteLine("4295:");
    // Check the reliability of ZMVCH using exact data.
    n = Math.Min(32, nmax);
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            a[i - 1, j - 1] = Math.Max(i - j + 1, 0);
        }
        x[j - 1] = j;
        y[j - 1] = zero;
    }
    for (int j = 1; j <= n; j = j + 1)
    {
        yy[j - 1] = j * ((j + 1) * j) / 2 - ((j + 1) * j * (j - 1)) / 3;
    }
    // YY holds the exact result. On exit from ZMVCH YT holds the result computed by ZMVCH
    trans = "N";
    //DMVCH(trans, n, n, one, a, nmax, x, 1, zero, y, 1, yt, g, yy, eps, err, fatal, nout, true);
    ZMVCH(trans, n, n, one, a, nmax, x, 1, zero, y, 1, ref yt, ref g, yy, eps, ref err, ref fatal, foutl, true);

```

```

//static double[,] get_2dfrom1d(int l, int lda, double[] arr)
//static void DMVCH(string trans, int m, int n, double alpha, double[,] a, int nmax, double[] x, int indx, double beta, double[] y, int incy,
double[] yt,
//double[] g, double[] yy, double eps, double err, bool fatal, StreamWriter nout, bool mv
//Console.WriteLine("4319:");
same = LZE(yy, yt, n);
if ((!same) || (err != zero))
{
    fout1.WriteLine("ERROR IN ZMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("ZMVCH WAS CALLED WITH TRANS = {0} AND RETURNED SAME = {1} AND ERR = {2,12:F3}", trans, same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}

trans = "T";
//DMVCH(trans, n, n, one, a, nmax, x, -1, zero, y, -1, yt, g, yy, eps, err, fatal, nout, true);
//fout1.WriteLine("2nd DMVCH");
ZMVCH(trans, n, n, one, a, nmax, x, -1, zero, y, -1, ref yt, ref g, yy, eps, ref err, ref fatal, fout1, true);
//nout1.Flush();
//nout1.close();
//Environment.Exit(1);

same = LZE(yy, yt, n);
if ((!same) || (err != zero))
{
    fout1.WriteLine("ERROR IN DMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("DMVCH WAS CALLED WITH TRANS = {0} AND RETURNED SAME = {1} AND ERR = {2,12:F3}", trans, same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
//Console.WriteLine("4341:");
// Test each subroutine in turn
for (int isnum = 1; isnum <= nsubs; isnum = isnum + 1)
{
    //fout1.WriteLine("****SNAME={0} {1}", snames[isnum - 1], isnum);
    fout1.WriteLine();
    if (!test[isnum - 1])
    {
        // Subprogram is not to be tested
        fout1.WriteLine(" {0,6} WAS NOT TESTED", snames[isnum - 1]);
    }
    else
    {
        srnamt = snames[isnum - 1];
        // Test error exits
        if (tsterr)
        {
            ZCHK1(isnum, snames[isnum - 1], fout1);
            fout1.WriteLine();
        }
        // Test computations
        infot = 0;
        ok = true;
        fatal = false;
        //fout1.WriteLine("^^^isnum={0}", isnum);

        switch (isnum)
        //switch (isnum+50)
        {
            case 1:
            case 2:
                // Test ZGEMV, 01, and ZGBMV, 02.
                ZCHK1(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, nalf, alf, nbet, bet, ninc,
inc, nmax,
                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g);
                //fout1.Flush();
                //fout1.Close();
                //Environment.Exit(1);
                break;
            case 3:
            case 4:
            case 5:
                // Test ZHEMV, 03, ZHBMV, 04, and ZHPMV, 05.
                ZCHK2(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, nalf, alf, nbet, bet, ninc,
inc, nmax,
                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g);
                break;
            case 6:
            case 7:
            case 8:
            case 9:
            case 10:
            case 11:
                // Test ZTRMV, 06, ZTBMV, 07, ZTPMV, 08, ZTRSV, 09, ZTBSV, 10, and ZTPSV, 11.
                ZCHK3(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nkb, kb, ninc, inc, nmax,
                incmax, ref a, ref aa, _as, ref y, ref yy, ys, yt, g, z);
                break;
            case 12:
            case 13:
                // Test ZGERC, 12, ZGERU, 13.
                ZCHK4(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,
                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, z);
                break;
            case 14:
            case 15:
                //*****
                // if not ref a, aa, x, xx, y, yy then aa not same for DCHK5
                //
                // Test ZHER, 14, and ZHPR, 15.
                ZCHK5(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,
                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, z);
                break;
        }
    }
}

```

```

        case 16:
        case 17:
            // Test ZHER2, 16, and ZHPR2, 17.
            ZCHK6(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim, nalf, alf, ninc, inc, nmax,
                incmax, ref a, ref aa, _as, ref x, ref xx, xs, ref y, ref yy, ys, yt, g, zget_2dfrom1d(2 * nmax, 0, nmax, z));

            break;
        default:
            break;
    }
    if (fatal || sfatal)
    {
        fout1.WriteLine("\n***** FATAL ERROR - TESTS ABANDONED *****");
        if (trace)
        {
            fout2.Flush();
            fout2.Close();
        }
        fout1.Flush();
        fout1.Close();

        Environment.Exit(1);
    }
}

//Console.WriteLine("4421:");
fout1.WriteLine("\nEND OF TESTS");
if (trace)
{
    fout2.Flush();
    fout2.Close();
}

Console.WriteLine("end.");
fout1.Flush();
fout1.Close();
}

static Complex[,] zget_2dfrom1d(int l, int start, int lda, Complex[] arr)
{
    //updated 2015-09-02
    int k = 1 / lda;
    Complex[,] tmp = new Complex[lda, k];
    int cnt1 = 0;
    int cnt2 = 0;
    //Console.WriteLine("2dfrom1d D1:{0,4:D}", arr.GetLength(0));
    //Console.WriteLine("l:{0,4:D} lda:{1,4:D} k:{2,4:D}", l, lda, k);
    for (int i = start; i < k * lda; i = i + 1)
    {
        tmp[cnt1, cnt2] = arr[i];
        //Console.WriteLine("{0,4:D}{1,18:F5}{2,4:D}{3,4:D}{4,18:F5}", i, arr[i], cnt1, cnt2, tmp[cnt1,cnt2]);
        if (cnt1 == (lda - 1))
        {
            cnt2++;
            cnt1 = 0;
        }
        else
        {
            cnt1++;
        }
    }

    return tmp;
}

static void zprint_2dfrom1d(int l, int start, int lda, Complex[] arr, Complex[,] arr2d)
{
    int k = 0;
    k = 1 / lda;
    //Console.WriteLine("{0}", 1 / lda);
    //double[,] tmp = new Double[lda, k];
    int cnt1 = 0;
    int cnt2 = 0;
    Console.WriteLine("D1:{0,4:D} D2:{1,4:D}", arr2d.GetLength(0), arr2d.GetLength(1));
    Console.WriteLine("l:{0,4:D} lda:{1,4:D} k:{2,4:D}", l, lda, k);
    for (int i = 0; i < l; i = i + 1)
    {
        Console.WriteLine("{0,4:D}{1,18}{2,4:D}{3,4:D}{4,18}", i, arr[i].ToString("F5"), cnt1, cnt2, arr2d[cnt1, cnt2].ToString("F5"));    /// check
        format {1,18} for complex
        if (cnt1 == (lda - 1))
        {
            cnt2++;
            cnt1 = 0;
        }
        else
        {
            cnt1++;
        }
    }
}

static Complex[] zget_1dfrom2d(int l, int start, int lda, Complex[,] arr)
{
    int k = 1 / lda;
    Complex[] tmp = new Complex[l];
    int cnt1 = start;
    int cnt2 = 0;
    for (int i = 0; i < k * lda - start; i = i + 1)
    {
        tmp[i] = arr[cnt1, cnt2];
        if (cnt1 == (lda - 1))
        {
            cnt2++;
            cnt1 = 0;
        }
        else
        {
            cnt1++;
        }
    }

    return tmp;
}

static Complex[] zget_1dfrom2d_starti(int l1, int start1, int l2, int start2, Complex[,] arr)
{

```

```

        // start zero based
        //int k = 1 / lda;
        int l = (l1 - start1) * (l2 - start2);
        Complex[] tmp = new Complex[l];
        int cnt1 = start1;
        int cnt2 = start2;
        for (int i = 0; i < l; i = i + 1)
        {
            tmp[i] = arr[cnt1, cnt2];
            cnt1++;
            if (cnt1 >= l1)
            {
                cnt1 = 0;
                cnt2++;
            }
        }
        return tmp;
    }
    static Complex[] zget_ldfrom2d_starti_full(int l1, int start1, int l2, int start2, Complex[,] arr)
    {
        // start zero based
        //int k = 1 / lda;
        int l = (l1 - start1) * (l2 - start2);
        int lnew = l1 * l2; //see get_ldfrom2d_starti (works in some cases) where shortens length based on start positions for new array--here we use
        original length
        //double[] tmp = new Double[l];
        Complex[] tmp = new Complex[lnew];
        int cnt1 = start1;
        int cnt2 = start2;
        for (int i = 0; i < l; i = i + 1)
        {
            tmp[i] = arr[cnt1, cnt2];
            cnt1++;
            if (cnt1 >= l1)
            {
                cnt1 = 0;
                cnt2++;
            }
        }
        return tmp;
    }

    static void zcopy_ld(Complex[] src1, int s1, int l1, Complex[] src2, int s2)
    {
        // zero based
        int cnt1 = s2;
        for (int i = s1; i < l1; i = i + 1)
        {
            src2[cnt1] = src1[i];
            cnt1++;
        }
    }

    static void ZCHK1(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
    int[] idim,
        int nkb, int[] kb, int nalf, Complex[] alf, int nbet, Complex[] bet, int ninc, int[] inc, int nmax, int incmax,
        ref Complex[,] a, ref Complex[] aa, Complex[] _as, ref Complex[] x, ref Complex[] xx, Complex[] xs, ref Complex[] y, ref Complex[] yy,
    Complex[] ys, Complex[] yt, double[] g)
    {
        // Tests ZGEMV and ZGBMV

        // Parameters
        Complex zero = new Complex(0.0, 0.0);
        Complex half = new Complex(0.5, 0.0);
        const double rzero = 0.0;

        /*// Scalar Arguments
        double eps, thresh;
        int incmax, nalf, nbet, nidim, ninc, nkb, nmax; //, nout, ntra;
        StringWriter nout, ntra;
        bool fatal, rewi, trace;
        string sname;

        // Array Arguments
        Complex[,] a = new Complex[nmax,nmax];
        Complex[] aa = new Complex[nmax*nmax];
        Complex[] alf = new Complex[nalf];
        Complex[] _as = new Complex[nmax*nmax];
        Complex[] bet = new Complex[nbet];

        Complex[] x = new Complex[nmax];
        Complex[] xs = new Complex[nmax*incmax];
        Complex[] xx = new Complex[nmax*incmax];
        Complex[] y = new Complex[nmax];
        Complex[] ys = new Complex[nmax*incmax];
        Complex[] yt = new Complex[nmax];
        Complex[] yy = new Complex[nmax*incmax];
        double[] g = new Complex[nmax];
        int[] idim = new int[nidim];
        int[] inc = new int[ninc];
        int[] kb = new int[nkb];*/

        // Local Scalars
        Complex alpha, als, beta, bls, transl;
        double err, errmax;
        //int i, ia, ib, ic, iku, im, _in, incx, incxs, incy, incys, ix, iy, kl, kls, ku, kus, laa, lda, ldas, lx, ly, m, ml, ms, n, nargs, nc, nd, nk,
    nl, ns;

        int incx, incxs, incy, incys, kl, kls, ku, kus, laa, lda, ldas, lx, ly, m, ml, ms, n, nargs, nc, nd, nk, nl, ns;
        bool banded, full, null, reset, same, tran;
        string trans, transs, ich;
        /*
        CHARACTER*1      TRANS, TRANSS
        CHARACTER*3      ICH */

        // Local Arrays
        bool[] isame = new bool[13];

        Complex[,] tmp2d_x;
        Complex[,] tmp2d_y;
        /* External Functions
        LOGICAL          LZE, LZERES
        EXTERNAL          LZE, LZERES */

```

```

/* Common ..
INTEGER      INFOT, NOUTC
LOGICAL      LERR, OK
* .. Common blocks ..
COMMON       /INFOC/INFOT, NOUTC, OK, LERR */

err = 0.0; // added
ich = "NTC";

// Executable Statements
full = (sname.ToUpper().Substring(2, 1) == "E");
banded = (sname.ToUpper().Substring(2, 1) == "B");
//nout.WriteLine("^^^CHK1");
// Define the number of arguments
if (full)
{
    nargs = 11;
}
else if (banded)
{
    nargs = 13;
}
else
{
    nargs = 0; // added
}

nc = 0;
reset = true;
errmax = rzero;

m = 0; // added
//120
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    //Console.WriteLine("_in={0}", _in);
    //nout.WriteLine("^^^_in={0}", _in);
    n = idim[_in - 1];
    nd = n / 2 + 1;

    //110
    for (int im = 1; im <= 2; im = im + 1)
    {
        //Console.WriteLine("im={0}", im);
        switch (im)
        {
            case 1:
                m = Math.Max(n - nd, 0);
                break;
            case 2:
                m = Math.Min(n + nd, nmax);
                break;
            default:
                break;
        }
        if (banded)
        {
            nk = nkb;
        }
        else
        {
            nk = 1;
        }
    }
    //100

    for (int iku = 1; iku <= nk; iku = iku + 1)
    {
        //Console.WriteLine("iku={0}", iku);
        if (banded)
        {
            ku = kb[iku - 1];
            kl = Math.Max(ku - 1, 0);
        }
        else
        {
            ku = n - 1;
            kl = m - 1;
        }

        // Set LDA to 1 more than minimum value if room
        if (banded)
        {
            lda = kl + ku + 1;
        }
        else
        {
            lda = m;
        }
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            continue;
        }
        laa = lda * n;
        _null = (n <= 0) || (m <= 0);

        // Generate the matrix A
        transl = zero;
        //Console.WriteLine("bef DMAKE");
        ZMAKE(sname.Substring(1, 2), " ", " ", m, n, ref a, nmax, ref aa, lda, kl, ku, ref reset, transl);

        //Console.WriteLine("aft DMAKE");
    }
}

```

```

//90
for (int ic = 1; ic <= 3; ic = ic + 1)
{
    //Console.WriteLine("ic={0}", ic);
    trans = ich.ToUpper().Substring(ic - 1, 1);
    tran = (trans == "T") || (trans == "C");
    if (tran)
    {
        ml = n;
        nl = m;
    }
    else
    {
        ml = m;
        nl = n;
    }
}

//80
for (int ix = 1; ix <= ninc; ix = ix + 1)
{
    //Console.WriteLine("ix={0}", ix);
    incx = inc[ix - 1];
    lx = Math.Abs(incx) * nl;

    // Generate the vector X
    transl = half;
    //DMAKE("GE", " ", " ", 1, nl, x, 1, xx, Math.Abs(incx), 0, nl-1, reset, transl);
    //Console.WriteLine("bef DMAKE");
    tmp2d_x = zget_2dfromld(nmax, 0, 1, x);
    ZMAKE("GE", " ", " ", 1, nl, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, nl - 1, ref reset, transl);
    x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
    //Console.WriteLine("aft DMAKE");
    if (nl > 1)
    {
        x[nl / 2 - 1] = zero;
        xx[1 + Math.Abs(incx) * (nl / 2 - 1) - 1] = zero;
    }
}

//70
for (int iy = 1; iy <= ninc; iy = iy + 1)
{
    //Console.WriteLine("iy={0}", iy);
    incy = inc[iy - 1];
    ly = Math.Abs(incy) * ml;
    //60
    for (int ia = 1; ia <= nalf; ia = ia + 1)
    {
        //Console.WriteLine("ia={0}", ia);
        alpha = alf[ia - 1];
        //50
        for (int ib = 1; ib <= nbet; ib = ib + 1)
        {
            //Console.WriteLine("ib={0}", ib);
            //nout.WriteLine("ib={0}", ib);
            beta = bet[ib - 1];

            // Generate the vector Y
            transl = zero;
            //DMAKE("GE", " ", " ", 1, ml, y, 1, yy, Math.Abs(incy), 0, ml-1, reset, transl);
            //Console.WriteLine("bef DMAKE");
            tmp2d_y = zget_2dfromld(nmax, 0, 1, y);
            ZMAKE("GE", " ", " ", 1, ml, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, ml - 1, ref reset, transl);
            y = zget_ldfrom2d(nmax, 0, 1, tmp2d_y);
            //Console.WriteLine("bef DMAKE");
            nc = nc + 1;

            // Save every datum before calling the subroutine
            transs = trans;
            ms = m;
            ns = n;
            kls = kl;
            kus = ku;
            als = alpha;

            for (int i = 1; i <= laa; i = i + 1)
            {
                _as[i - 1] = aa[i - 1];
            }

            ldas = lda;
            for (int i = 1; i <= lx; i = i + 1)
            {
                xs[i - 1] = xx[i - 1];
            }

            incxs = incx;
            bls = beta;
            for (int i = 1; i <= ly; i = i + 1)
            {
                ys[i - 1] = yy[i - 1];
            }

            incys = incy;
            //Console.WriteLine("4660:");
            // Call the subroutine
            if (full)
            {
                if (trace)
                {
                    //fout2 = new StreamWriter(snaps, true); // true = append
                    ntra.WriteLine(" {0,6:D}: {1,-6}, {{2,1}, {3,3:D}, {4,3:D}, {5}, A, {6,3:D}, X, {7,2:D}, {8}, Y,
                                nc, sname, trans, m, n, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //5,8 {4:F1}
                }

                if (rewi)
                {
                    //REWIND NTRA - do nothing
                }
                ZGEMV(trans, m, n, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, xx, incx, beta, ref yy, incy);
            }
            else if (banded)

```

```

{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,-6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7}, A, {8,3:D}, X,
(9,2:D), {10}, Y, {11,2:D}) .",
nc, sname, trans, m, n, kl, ku, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy);
//7,10 {,4:F1}
    }

    if (rewi)
    {
        //REWIND NTRA - do nothing
    }
    //nout.WriteLine("calling DGBMV in DCHK1");
    ZGBMV(trans, m, n, kl, ku, alpha, zget_2dfrold(nmax * nmax, 0, lda, aa), lda, xx, incx, beta, ref yy,
incy);
}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 130
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1},
Y, {9,2:D}) .",
nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D}, X,
(9,2:D), {10,4:F1}, Y, {11,2:D}) .",
nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (trans == transs);
isame[1] = (ms == m);
isame[2] = (ns == n);
if (full)
{
    isame[3] = (als == alpha);
    isame[4] = LZE(_as, aa, laa);
    isame[5] = (ldas == lda);
    isame[6] = LZE(xs, xx, lx);
    isame[7] = (incxs == incx);
    isame[8] = (bls == beta);
    if (_null)
    {
        isame[9] = LZE(ys, yy, ly);
    }
    else
    {
        //isame[9] = LDERES("GE", " ", 1, ml, ys, yy, Math.Abs(incy)); // double[] ys = new
double[nmax*incmax];
        isame[9] = LDERES("GE", " ", 1, ml, zget_2dfrold(nmax * incmax, 0, Math.Abs(incy), ys),
zget_2dfrold(nmax * incmax, 0, Math.Abs(incy), yy), Math.Abs(incy)); //nmax*incmax
    }
    isame[10] = (incys == incy);
}
else if (banded)
{
    isame[3] = (kls == kl);
    isame[4] = (kus == ku);
    isame[5] = (als == alpha);
    isame[6] = LZE(_as, aa, laa);
    isame[7] = (ldas == lda);
    isame[8] = LZE(xs, xx, lx);
    isame[9] = (incxs == incx);
    isame[10] = (bls == beta);
    if (_null)
    {
        isame[11] = LZE(ys, yy, ly);
    }
    else
    {
        //isame[11] = LDERES("GE", " ", 1, ml, ys, yy, Math.Abs(incy));
        isame[11] = LDERES("GE", " ", 1, ml, zget_2dfrold(nmax * incmax, 0, Math.Abs(incy), ys),
zget_2dfrold(nmax * incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[12] = (incys == incy);
}

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GOTO 130
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D}, {8,4:F1},
Y, {9,2:D}) .",
nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
    }
    else if (banded)

```

```

        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D}, X,
(9,2:D), {10,4:F1}, Y, {11,2:D}) .",
                nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
        }
        return;
    }
    if (!_null)
    {
        // Check the result
        //nout.WriteLine("DMVCH in DCHK1");
        ZMVCH(trans, m, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal,
nout, true);
        errmax = Math.Max(errmax, err);

        // If got really bad answer, report and return
        if (fatal)
        {
            //GO TO 130
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            if (full)
            {
                nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,4:F1}, A, {6,3:D}, X, {7,2:D},
(8,4:F1}, Y, {9,2:D}) .",
                    nc, sname, trans, m, n, alpha, lda, incx, beta, incy);
            }
            else if (banded)
            {
                nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5,3:D}, {6,3:D}, {7,4:F1}, A, {8,3:D},
X, {9,2:D}, {10,4:F1}, Y, {11,2:D}) .",
                    nc, sname, trans, m, n, kl, ku, alpha, lda, incx, beta, incy);
            }
            return;
        }
    }
    else
    {
        // Avoid repeating tests with M.le.0 or N.le.0.
        goto Loop110;
    }
} //50
} //60
} //70
} //80
} //90
} //100
Loop110: ;
//NOOP();
} //110
} //120

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^1");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
    nout.WriteLine("***** BUT WITH MAXIMUM TEST RATIO {0,8:F2} - SUSPECT *****", errmax);
}
return;
}
static void ZCHK2(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
    int nkb, int[] kb, int nalf, Complex[] alf, int nbet, Complex[] bet, int ninc, int[] inc, int nmax, int incmax,
    ref Complex[,] a, ref Complex[] aa, Complex[] _as, ref Complex[] x, ref Complex[] xx, Complex[] xs, ref Complex[] y, ref Complex[] yy,
Complex[] ys, Complex[] yt, double[] g)
{
    // Tests ZHEMV, ZHBMV and ZHPMV

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex half = new Complex(0.5, 0.0);
    const double rzero = 0.0;

    // Scalar Arguments
    /*double eps, thresh;
    int incmax, nalf, nbet, nidim, ninc, nkb, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    Complex[,] a = new Complex[nmax,nmax];
    Complex[] aa = new Complex[nmax*nmax];
    Complex[] alf = new Complex[nalf];
    Complex[] _as = new Complex[nmax*nmax];
    Complex[] bet = new Complex[nbet];
    Complex[] x = new Complex[nmax];
    Complex[] xs = new Complex[nmax*incmax];
    Complex[] xx = new Complex[nmax*incmax];
    Complex[] y = new Complex[nmax];
    Complex[] ys = new Complex[nmax*incmax];
    Complex[] yt = new Complex[nmax];
    Complex[] yy = new Complex[nmax*incmax];

    double[] g = new double[nmax];
    int[] idim = new int[nidim];
    int[] inc = new int[ninc];
    int[] kb = new int[nkb];*/

    // Local Scalars
    Complex alpha, als, beta, bls, transl;
    double err, errmax;
    //int i, ia, ib, ic, ik, _in, incx, incxs, incy, incys, ix, iy, k, ks, laa, lda, ldas, lx, ly, n, nargs, nc, nk, ns;
    int incx, incxs, incy, incys, k, ks, laa, lda, ldas, lx, ly, n, nargs, nc, nk, ns;
    bool banded, full, _null, packed, reset, same;
    string uplo, uplos, ich;
    /*
        CHARACTER*1          UPLO, UPLOS
    */
}

```

```

CHARACTER*2      ICH

*/
// Local Arrays
bool[] isame = new bool[13];

/* External Functions
LOGICAL      LZE, LZERES
EXTERNAL     LZE, LZERES */

Complex[,] tmp2d_x;
Complex[,] tmp2d_y;
*/
* .. Scalars in Common ..
  INTEGER      INFOT, NOUTC
  LOGICAL      LERR, OK
* .. Common blocks ..
  COMMON       /INFOC/INFOT, NOUTC, OK, LERR */

err = 0.0; // added
ich = "UL";

// Executable Statements
full = (sname.Substring(2, 1) == "F");
banded = (sname.Substring(2, 1) == "B");
packed = (sname.Substring(2, 1) == "P");

// Define the number of arguments
if (full)
{
  nargs = 10;
}
else if (banded)
{
  nargs = 11;
}
else if (packed)
{
  nargs = 9;
}
else
{
  nargs = 0; // added
}

nc = 0;
reset = true;
errmax = rzero;

//110
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
  n = idim[_in - 1];
  if (banded)
  {
    nk = nkb;
  }
  else
  {
    nk = 1;
  }

  //100
  for (int ik = 1; ik <= nk; ik = ik + 1)
  {
    if (banded)
    {
      k = kb[ik - 1];
    }
    else
    {
      k = n - 1;
    }

    // Set LDA to 1 more than minimum value if room
    if (banded)
    {
      lda = k + 1;
    }
    else
    {
      lda = n;
    }
    if (lda < nmax)
    {
      lda = lda + 1;
    }

    // Skip tests if not enough room
    if (lda > nmax)
    {
      //GO TO 100
      continue;
    }
    if (packed)
    {
      laa = (n * (n + 1)) / 2;
    }
    else
    {
      laa = lda * n;
    }
    _null = (n <= 0);

    //90
    for (int ic = 1; ic <= 2; ic = ic + 1)
    {
      uplo = ich.ToUpper().Substring(ic - 1, 1);

      // Generate the matrix A
      transl = zero;
      ZMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, k, k, ref reset, transl);
    }
  }
}

```

```

//80
for (int ix = 1; ix <= ninc; ix = ix + 1)
{
    incx = inc[ix - 1];
    lx = Math.Abs(incx) * n;

    // Generate the vector X
    transl = half;
    tmp2d_x = zget_2dfromld(nmax, 0, 1, x);
    ZMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
    x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
    if (n > 1)
    {
        x[(n / 2) - 1] = zero;
        xx[1 + Math.Abs(incx) * (n / 2 - 1) - 1] = zero;
    }

// 70
for (int iy = 1; iy <= ninc; iy = iy + 1)
{
    incy = inc[iy - 1];
    ly = Math.Abs(incy) * n;

//60
for (int ia = 1; ia <= nalf; ia = ia + 1)
{
    alpha = alf[ia - 1];

//50
for (int ib = 1; ib <= nbet; ib = ib + 1)
{
    beta = bet[ib - 1];

    // Generate the vector Y
    transl = zero;
    tmp2d_y = zget_2dfromld(nmax, 0, 1, y);
    ZMAKE("GE", " ", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
    y = zget_ldfrom2d(nmax, 0, 1, tmp2d_y);
    nc = nc + 1;

    // Save every datum before calling the subroutine
    uplo = uplo;
    ns = n;
    ks = k;
    als = alpha;

    for (int i = 1; i <= laa; i = i + 1)
    {
        _as[i - 1] = aa[i - 1];
    }

    ldas = lda;
    for (int i = 1; i <= lx; i = i + 1)
    {
        xs[i - 1] = xx[i - 1];
    }

    incxs = incx;
    bls = beta;
    for (int i = 1; i <= ly; i = i + 1)
    {
        ys[i - 1] = yy[i - 1];
    }

    incys = incy;

    // Call the subroutine
    if (full)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, A, {5,3:D}, X, {6,2:D}, {7}, Y, {8,2:D})
                nc, sname, uplo, n, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); // 4,7 {,4:F1}
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        ZHEMV(uplo, n, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, xx, incx, beta, ref yy, incy);
    }
    else if (banded)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5}, A, {6,3:D}, X, {7,2:D}, {8}, Y, {9,2:D})
                nc, sname, uplo, n, k, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //5,8 {,4:F1}
        }
        if (rewi)
        {
            //REWIND NTRA
        }
        ZHEMV(uplo, n, k, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, xx, incx, beta, ref yy, incy);
    }

    else if (packed)
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, AP, X, {5,2:D}, {6}, Y, {7,2:D})
                nc, sname, uplo, n, alpha.ToString("F1"), incx, beta.ToString("F1"), incy); //4,6 {,4:F1}
        }
        if (rewi)
        {
            //REWIND NTRA

```

```

    }
    /* nout.WriteLine("NC:{0}",nc);
    nout.WriteLine("UPLO, N, ALPHA, INCX, BETA, INCY");
    nout.WriteLine("{0,1}{1,4:D}{2,10:F3}{3,4:D}{4,10:F3}{5,4:D}",uplo,n,alpha,incx,beta,incy);
    nout.WriteLine("^^^AA");
    for (int i = 1; i <= nmax*nmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",aa[i-1]);
    }
    nout.WriteLine("^^^XX");
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",xx[i-1]);
    }
    nout.WriteLine("^^^YY");
    for (int i = 1; i <= nmax*incmax; i = i + 1) {
        nout.WriteLine("{0,18:F5}",yy[i-1]);
    }
    */
    /* nout.Flush();
    nout.Close();
    Environment.Exit(1);*/

    //misc_d.myflag = true;
    ZHPMV(uplo, n, alpha, aa, xx, incx, beta, ref yy, incy);
    /*nout.WriteLine("^^^OUT YY");
    for (int i = 1; i <= nmax * incmax; i = i + 1)
    {
        nout.WriteLine("{0,18:F5}", yy[i - 1]);
    }
    */

}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, A, {5,3:D}, X, {6,2:D}, {7}, Y, {8,2:D})
        nc, sname, uplo, n, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //4,7 {,4:F1}
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5}, A, {6,3:D}, X, {7,2:D}, {8}, Y, {9,2:D})
        nc, sname, uplo, n, k, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //5,8 {,4:F1}
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, AP, X, {5,2:D}, {6}, Y, {7,2:D})
        nc, sname, uplo, n, alpha.ToString("F1"), incx, beta.ToString("F1"), incy); //4,6 {,4:F1}
    }
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (ns == n);
if (full)
{
    isame[2] = (als == alpha);
    isame[3] = LZE(_as, aa, laa);
    isame[4] = (lda == lda);
    isame[5] = LZE(xs, xx, lx);
    isame[6] = (incxs == incx);
    isame[7] = (bls == beta);
    if (_null)
    {
        isame[8] = LZE(ys, yy, ly);
    }
    else
    {
        //isame[8] = LDERES("GE", " ", 1, n, ys, yy, Math.Abs(incy));
        isame[8] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incy), ys), zget_2dfromld(nmax
* incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[9] = (incys == incy);
}

else if (banded)
{
    isame[2] = (ks == k);
    isame[3] = (als == alpha);
    isame[4] = LZE(_as, aa, laa);
    isame[5] = (lda == lda);
    isame[6] = LZE(xs, xx, lx);
    isame[7] = (incxs == incx);
    isame[8] = (bls == beta);
    if (_null)
    {
        isame[9] = LZE(ys, yy, ly);
    }
    else
    {
        //isame[9] = LDERES("GE", " ", 1, n, ys, yy, Math.Abs(incy));
        isame[9] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incy), ys), zget_2dfromld(nmax
* incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[10] = (incys == incy);
}

else if (packed)
{
    isame[2] = (als == alpha);
    isame[3] = LZE(_as, aa, laa);
    isame[4] = LZE(xs, xx, lx);
    isame[5] = (incxs == incx);
    isame[6] = (bls == beta);
    if (_null)
    {
        isame[7] = LZE(ys, yy, ly);
    }
    else
    {

```

```

        isame[7] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incy), ys), zget_2dfromld(nmax
* incmax, 0, Math.Abs(incy), yy), Math.Abs(incy));
    }
    isame[8] = (incys == incy);
}

// If data was incorrectly changed, report and return
same = true;
//40
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, A, {5,3:D}, X, {6,2:D}, {7}, Y, {8,2:D})
        nc, sname, uplo, n, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //4,7 {,4:F1}
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5}, A, {6,3:D}, X, {7,2:D}, {8}, Y, {9,2:D})
        nc, sname, uplo, n, k, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //5,8 {,4:F1}
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, AP, X, {5,2:D}, {6}, Y, {7,2:D})
        nc, sname, uplo, n, alpha.ToString("F1"), incx, beta.ToString("F1"), incy); //4,6 {,4:F1}
    }
    return;
}
if (!_null)
{
    // Check the result
    //DMVCH("N", n, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal, nout,
true);
    //nout.WriteLine("take out");
    //misc_d.myflag = true;
    //ZMVCH("N", n, n, alpha, a, nmax, x, incx, beta, y, incy, ref yt, ref g, yy, eps, ref err, ref fatal, nout,
true);
    //misc_d.myflag = false;
    errmax = Math.Max(errmax, err);
    // If got really bad answer, report and return.
    if (fatal)
    {
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, A, {5,3:D}, X, {6,2:D}, {7}, Y, {8,2:D})
            nc, sname, uplo, n, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //4,7 {,4:F1}
        }
        else if (banded)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,3:D}, {5}, A, {6,3:D}, X, {7,2:D}, {8}, Y, {9,2:D})
            nc, sname, uplo, n, k, alpha.ToString("F1"), lda, incx, beta.ToString("F1"), incy); //5,8 {,4:F1}
        }
        else if (packed)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, AP, X, {5,2:D}, {6}, Y, {7,2:D})
            nc, sname, uplo, n, alpha.ToString("F1"), incx, beta.ToString("F1"), incy); //4,6 {,4:F1}
        }
        return;
    }
}
}
else
{
    // Avoid repeating tests with N.le.0
    goto Loop110;
}
} //50
} //60
} //70
} //80
} //90
} //100
Loop110: ;
} //110

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^2");
    nout.WriteLine(" {0,-6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,-6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}
static void ZCHK3(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
int nkb, int[] kb, int ninc, int[] inc, int nmax, int incmax, ref Complex[,] a, ref Complex[,] aa, Complex[] _as, ref Complex[] x, ref Complex[]
xx, Complex[] xs, Complex[] xt, double[] g, Complex[] z)
{
    // Tests ZTRMV, ZTBMV, ZTPMV, ZTRSV, ZTBSV and ZTPSV

```

```

// Parameters
Complex zero = new Complex(0.0, 0.0);
Complex half = new Complex(0.5, 0.0);
Complex one = new Complex(1.0, 0.0);
const double rzero = 0.0;

// Scalar Arguments
/*double eps, thresh;
int incmax, nidim, ninc, nkb, nmax; // nout, ntra;
StringWriter nout, ntra;
bool fatal, rewi, trace;
string sname;

// Array Arguments
Complex[,] a = new Complex[nmax,nmax];
Complex[] aa = new Complex[nmax*nmax];
Complex[] _as = new Complex[nmax*nmax];
Complex[] x = new Complex[nmax];
Complex[] xs = new Complex[nmax*incmax];
Complex[] xt = new Complex[nmax];
Complex[] xx = new Complex[nmax*incmax];
Complex[] z = new Complex[nmax];
*
double[] g = new double[nmax];
int[] idim = new int[nidim];
int[] inc = new int[ninc];
int[] kb = new int[nkb]; */

// Local Scalars
Complex transl;
double err, errmax;
//int i, id, ict, icu, ik, _in, incx, incxs, ix, k, ks, laa, lda, ldas, lx, n, nargs, nc, nk, ns;
int incx, incxs, k, ks, laa, lda, ldas, lx, n, nargs, nc, nk, ns;
bool banded, full, _null, packed, reset, same;
string diag, diags, trans, transs, uplo, uplos, ichd, ichu, icht;
/* CHARACTER*1          DIAG, DIAGS, TRANS, TRANS, UPLO, UPLOS
   CHARACTER*2          ICHD, ICHU
   CHARACTER*3          ICHT
*/
// Local Arrays
bool[] isame = new bool[13];

Complex[,] tmp2d_x;
/*... Scalars in Common ...
INTEGER      INFOT, NOUTC
LOGICAL      LERR, OK
* .. Common blocks ..
COMMON      /INFOC/INFOT, NOUTC, OK, LERR*/

err = 0.0; //added
ichu = "UL";
icht = "NTC";
ichd = "UN";

// Executable Statements
full = (sname.Substring(2, 1) == "R");
banded = (sname.Substring(2, 1) == "B");
packed = (sname.Substring(2, 1) == "P");

// Define the number of arguments
if (full)
{
    nargs = 8;
}
else if (banded)
{
    nargs = 9;
}
else if (packed)
{
    nargs = 7;
}
else
{
    nargs = 0; // added
}

nc = 0;
reset = true;
errmax = rzero;

// Set up zero vector for DMVCH
for (int i = 1; i <= nmax; i = i + 1)
{
    z[i - 1] = zero;
}

//110
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    if (banded)
    {
        nk = nkb;
    }
    else
    {
        nk = 1;
    }
}

//100
for (int ik = 1; ik <= nk; ik = ik + 1)
{
    if (banded)
    {
        k = kb[ik - 1];
    }
    else
    {
        k = n - 1;
    }
    // Set LDA to 1 more than minimum value if room
    if (banded)
    {

```

```

        lda = k + 1;
    }
    else
    {
        lda = n;
    }
    if (lda < nmax)
    {
        lda = lda + 1;
    }

    // Skip tests if not enough room
    if (lda > nmax)
    {
        //GO TO 100
        goto Loop100;
    }
    if (packed)
    {
        laa = (n * (n + 1)) / 2;
    }
    else
    {
        laa = lda * n;
    }
    _null = (n <= 0);

    //90
    for (int icu = 1; icu <= 2; icu = icu + 1)
    {
        uplo = ichu.Substring(icu - 1, 1);
        //80
        for (int ict = 1; ict <= 3; ict = ict + 1)
        {
            trans = icht.Substring(ict - 1, 1);
            //70
            for (int icd = 1; icd <= 2; icd = icd + 1)
            {
                diag = ichd.Substring(icd - 1, 1);

                // Generate the matrix A
                transl = zero;
                ZMAKE(sname.Substring(1, 2), uplo, diag, n, n, ref a, nmax, ref aa, lda, k, k, ref reset, transl);
                //60
                for (int ix = 1; ix <= ninc; ix = ix + 1)
                {
                    incx = inc[ix - 1];
                    lx = Math.Abs(incx) * n;

                    // Generate the vector X
                    transl = half;
                    tmp2d x = zget_2dfromld(nmax, 0, 1, x);
                    ZMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
                    x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
                    if (n > 1)
                    {
                        x[(n / 2) - 1] = zero;
                        xx[1 + Math.Abs(incx) * (n / 2 - 1) - 1] = zero;
                    }
                    nc = nc + 1;

                    // Save every datum before calling the subroutine
                    uplos = uplo;
                    transs = trans;
                    diags = diag;
                    ns = n;
                    ks = k;
                    for (int i = 1; i <= laa; i = i + 1)
                    {
                        _as[i - 1] = aa[i - 1];
                    }
                    ldas = lda;
                    for (int i = 1; i <= lx; i = i + 1)
                    {
                        xs[i - 1] = xx[i - 1];
                    }
                    incxs = incx;

                    // Call the subroutine
                    if (sname.Substring(3, 2) == "MV")
                    {
                        if (full)
                        {
                            if (trace)
                            {
                                ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})

                                nc, sname, uplo, trans, diag, n, lda, incx);
                            }
                            if (rewi)
                            {
                                //REWIND NTRA
                            }
                            ZTRMV(uplo, trans, diag, n, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, ref xx, incx);
                        }
                        else if (banded)
                        {
                            if (trace)
                            {
                                ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})

                                nc, sname, uplo, trans, diag, n, lda, incx);
                            }
                            if (rewi)
                            {
                                //REWIND NTRA
                            }
                            ZTBMV(uplo, trans, diag, n, k, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, ref xx, incx);
                        }
                        else if (packed)
                        {
                            if (trace)

```



```

        return;
    }

    // See what data changed inside subroutines
    isame[0] = (uplo == uplos);
    isame[1] = (trans == transs);
    isame[2] = (diag == diags);
    isame[3] = (ns == n);
    if (full)
    {
        isame[4] = LZE(_as, aa, laa);
        isame[5] = (lda == lda);
        if (_null)
        {
            isame[6] = LZE(xs, xx, lx);
        }
        else
        {
            isame[6] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incx), xs), zget_2dfromld(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
        }
    }
    isame[7] = (incxs == incx);
}
else if (banded)
{
    isame[4] = (ks == k);
    isame[5] = LZE(_as, aa, laa);
    isame[6] = (lda == lda);
    if (_null)
    {
        isame[7] = LZE(xs, xx, lx);
    }
    else
    {
        isame[7] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incx), xs), zget_2dfromld(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
    }
    isame[8] = (incxs == incx);
}

else if (packed)
{
    isame[4] = LZE(_as, aa, laa);
    if (_null)
    {
        isame[5] = LZE(xs, xx, lx);
    }
    else
    {
        isame[5] = LZERES("GE", " ", 1, n, zget_2dfromld(nmax * incmax, 0, Math.Abs(incx), xs), zget_2dfromld(nmax *
incmax, 0, Math.Abs(incx), xx), Math.Abs(incx));
    }
    isame[6] = (incxs == incx);
}

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D}) .",
nc, sname, uplo, trans, diag, n, lda, incx);
    }
    else if (banded)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D}) .",
nc, sname, uplo, trans, diag, n, k, lda, incx);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1}, {5,3:d}), AP, X, {6,2:D}) .",
nc, sname, uplo, trans, diag, n, incx);
    }
}
return;

}
if (!_null)
{
    if (sname.Substring(3, 2) == "MV")
    {
        // Check the result
        ZMVCH(trans, n, n, one, a, nmax, x, incx, zero, z, incx, ref xt, ref g, xx, eps, ref err, ref fatal, nout,
true);
    }
    else if (sname.Substring(3, 2) == "SV")
    {
        // Compute approximation to original vector
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1] = xx[1 + (i - 1) * Math.Abs(incx) - 1];
            xx[1 + (i - 1) * Math.Abs(incx) - 1] = x[i - 1];
        }
        ZMVCH(trans, n, n, one, a, nmax, z, incx, zero, x, incx, ref xt, ref g, xx, eps, ref err, ref fatal, nout,
false);
    }
}

```

```

    }
    errmax = Math.Max(errmax, err);

    // If got really bad answer, report and return
    if (fatal)
    {
        //GO TO 120
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full)
        {
            nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, A, {6,3:D}, X, {7,2:D})

            nc, sname, uplo, trans, diag, n, lda, incx);
        }
        else if (banded)
        {
            nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1} {5,3:D}, {6,3:D}, A, {7,3:D}, X, {8,2:D})

            nc, sname, uplo, trans, diag, n, k, lda, incx);
        }
        else if (packed)
        {
            nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,1}, {5,3:d}, AP, X, {6,2:D})

            nc, sname, uplo, trans, diag, n, incx);
        }
        return;
    }
}
else
{
    // Avoid repeating tests with N.le.0.
    goto Loop110;
}
} //60
} //70
} //80
} //90
Loop100: ;
} //100
Loop110: ;
} //110

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^3");

    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}

static void ZCHK4(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
int nalf, Complex[] alf, int ninc, int[] inc, int nmax, int incmax, ref Complex[,] a, ref Complex[] aa, Complex[] _as, ref Complex[] x, ref
Complex[] xx, Complex[] xs,
ref Complex[] y, ref Complex[] yy, Complex[] ys, Complex[] yt, double[] g, Complex[] z)
{
    // Tests ZGERC and ZGERU

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex half = new Complex(0.5, 0.0);
    Complex one = new Complex(1.0, 0.0);
    const double rzero = 0.0;

    // Scalar Arguments
    /*double eps, thresh;
    int incmax, nalf, nidim, ninc, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    Complex[,] a = new Complex[nmax,nmax];
    Complex[] aa = new Complex[nmax*nmax];
    Complex[] alf = new Complex[nalf];
    Complex[] _as = new Complex[nmax*nmax];

    Complex[] x = new Complex[nmax];
    Complex[] xs = new Complex[nmax*incmax];
    Complex[] xx = new Complex[nmax*incmax];
    Complex[] y = new Complex[nmax];
    Complex[] ys = new Complex[nmax*incmax];
    Complex[] yt = new Complex[nmax];
    Complex[] yy = new Complex[nmax*incmax];
    Complex[] z = new Complex[nmax];

    double[] g = new double[nmax];

    int[] idim = new int[nidim];
    int[] inc = new int[ninc];*/

    // Local Scalars
    Complex alpha, als, transl;
    double err, errmax;
    //int i, ia, im, _in, incx, incxs, incy, incys, ix, iy, j, laa, lda, ldas, lx, ly, m, ms, n, nargs, nc, nd, ns;
    int incx, incxs, incy, incys, laa, lda, ldas, lx, ly, m, ms, n, nargs, nc, nd, ns;
    bool conj, _null, reset, same;

    // Local Arrays
    Complex[] w = new Complex[1];
    bool[] isame = new bool[13];

    Complex[, ] tmp2d_x, tmp2d_y, tmp2d_aa;

    /* .. Scalars in Common ..

```

```

        INTEGER          INFOT, NOUTC
        LOGICAL          LERR, OK
        * .. Common blocks ..
        COMMON           /INFOC/INFOT, NOUTC, OK, LERR */

// Executable Statements
err = 0.0; // added
conj = (sname.Substring(4, 1) == "C");
// Define the number of arguments
nargs = 9;
nc = 0;
reset = true;
errmax = rzero;
m = 0; // added

//120
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    nd = n / 2 + 1;

//110
    for (int im = 1; im <= 2; im = im + 1)
    {
        if (im == 1)
        {
            m = Math.Max(n - nd, 0);
        }
        if (im == 2)
        {
            m = Math.Min(n + nd, nmax);
        }

        // Set LDA to 1 more than minimum value if room.
        lda = m;
        if (lda < nmax)
        {
            lda = lda + 1;
        }

        // Skip tests if not enough room
        if (lda > nmax)
        {
            goto Loop110;
        }
        laa = lda * n;
        _null = (n <= 0) || (m <= 0);

//100
        for (int ix = 1; ix <= ninc; ix = ix + 1)
        {
            incx = inc[ix - 1];
            lx = Math.Abs(incx) * m;

            // Generate the vector X
            transl = half;
            tmp2d_x = zget_2dfromld(nmax, 0, 1, x);
            ZMAKE("GE", " ", " ", 1, m, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, m - 1, ref reset, transl);
            x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
            if (m > 1)
            {
                x[(m / 2) - 1] = zero;
                xx[(1 + Math.Abs(incx) * (m / 2 - 1)) - 1] = zero;
            }

//90
            for (int iy = 1; iy <= ninc; iy = iy + 1)
            {
                incy = inc[iy - 1];
                ly = Math.Abs(incy) * n;

                // Generate the vector Y
                transl = zero;
                tmp2d_y = zget_2dfromld(nmax, 0, 1, y);
                ZMAKE("GE", " ", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
                y = zget_ldfrom2d(nmax, 0, 1, tmp2d_y);

                if (n > 1)
                {
                    y[(n / 2) - 1] = zero;
                    yy[(1 + Math.Abs(incy) * (n / 2 - 1)) - 1] = zero;
                }

//80
                for (int ia = 1; ia <= nalf; ia = ia + 1)
                {
                    alpha = alf[ia - 1];

                    // Generate the matrix A

                    transl = zero;
                    ZMAKE(sname.Substring(1, 2), " ", " ", m, n, ref a, nmax, ref aa, lda, m - 1, n - 1, ref reset, transl);
                    nc = nc + 1;

                    // Save every datum before calling the subroutine
                    ms = m;
                    ns = n;
                    als = alpha;

                    for (int i = 1; i <= laa; i = i + 1)
                    {
                        _as[i - 1] = aa[i - 1];
                    }
                    ldas = lda;
                    for (int i = 1; i <= lx; i = i + 1)
                    {
                        xs[i - 1] = xx[i - 1];
                    }

                    incxs = incx;
                    for (int i = 1; i <= ly; i = i + 1)
                    {
                        ys[i - 1] = yy[i - 1];
                    }
                }
            }
        }
    }
}

```

```

incys = incy;

// Call the subroutine
if (trace)
{
    ntra.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
        nc, sname, m, n, alpha, incx, incy, lda);
}
if (conj)
{
    if (rewi)
    {
        //REWIND NTRA

        tmp2d_aa = zget_2dfromld(nmax * nmax, 0, lda, aa);
        ZGERC(m, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
        aa = zget_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
        /*tmp2d_aa = get_2dfromld(nmax * nmax, 0, lda, aa);
        // (int m, int n, double alpha, double[] x, int incx, double[] y, int incy, ref double[,] a, int lda
        DGER(m, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
        aa = get_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
        */

    }
    else
    {
        if (rewi)
        {
            //REWIND NTRA

            tmp2d_aa = zget_2dfromld(nmax * nmax, 0, lda, aa);
            ZGERU(m, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
            aa = zget_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
            // public static void ZGERU(int m, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[,]
a, int lda

        }

    }

    // Check if error-exit was taken incorrectly
    if (!ok)
    {
        nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");

        fatal = true;
        //GO TO 140
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
            nc, sname, m, n, alpha, incx, incy, lda);
        return;
    }

    // See what data changed inside subroutine
    isame[0] = (ms == m);
    isame[1] = (ns == n);
    isame[2] = (als == alpha);
    isame[3] = LZE(xs, xx, lx);
    isame[4] = (incxs == incx);
    isame[5] = LZE(ys, yy, ly);
    isame[6] = (incys == incy);
    if (!_null)
    {
        isame[7] = LZE(_as, aa, laa);
    }
    else
    {
        isame[7] = LZERES("GE", " ", m, n, zget_2dfromld(nmax * nmax, 0, lda, _as), zget_2dfromld(nmax * nmax, 0, lda, aa),
lda);
    }
    isame[8] = (ldas == lda);

    // If data was incorrectly changed, report and return
    same = true;
    //40
    for (int i = 1; i <= nargs; i = i + 1)
    {
        same = (same && isame[i - 1]);
        if (!isame[i - 1])
        {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }
    if (!same)
    {
        fatal = true;
        //GO TO 140
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D}, {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
            nc, sname, m, n, alpha, incx, incy, lda);
        return;
    }

    if (!_null)
    {
        // Check the result column by column
        if (incx > 0)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                z[i - 1] = x[i - 1];
            }
        }
        else
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                z[i - 1] = x[(m - i + 1) - 1];
            }
        }
    }
    //70
    for (int j = 1; j <= n; j = j + 1)
    {
        if (incy > 0)

```

```

        {
            w[0] = y[j - 1];
        }
        else
        {
            w[0] = y[(n - j + 1) - 1];
        }

        if (conj)
        {
            w[0] = Complex.Conjugate(w[0]);
        }
        //DMVCH("N", m, 1, alpha, z, nmax, w, 1, one, a[l-1,j-1], 1, yt, g, aa[l+(j-1)*lda-1], eps, err, fatal, nout,
true);
        ZMVCH("N", m, 1, alpha, zget_2dfromld(nmax, 0, nmax, z), nmax, w, 1, one, get_zarr2d(0, nmax - 1, j - 1, a), 1, ref
yt, ref g, get_zarr(1 + (j - 1) * lda - 1, nmax * nmax - 1, aa), eps, ref err, ref fatal, nout, true);
        //CALL ZMVCH( 'N', M, 1, ALPHA, Z, NMAX, W, 1, ONE, A( 1, J ), 1, YT, G, AA( 1 + ( J - 1
)*LDA ), EPS, ERR, FATAL, NOUT, .TRUE. )
        errmax = Math.Max(errmax, err);

        ///ZMVCH("N", lj, 1, alpha, zget_2dfromld(nmax, jj - 1, lj, z), lj, w, 1, one, zget_darr2d(jj - 1, nmax - 1, j -
1, a),
        // 1, ref yt, ref g, zget_darr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);

        // If got really bad answer, report and return
        if (fatal)
        {
            //GO TO 130
            nout.WriteLine(" THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
            //140
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            nout.WriteLine(" {0,6:D}: {1,6} ({2,3:D}, {3,3:D} {4,4:F1}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})
            nc, sname, m, n, alpha, incx, incy, lda);
            return;
        }
        } //70
    }
    else
    {
        // Avoid repeating tests with M.le.0 or N.le.0.
        goto Loop110;
    }
} //80
} //90
} //100
Loop110: ;
} //110
} //120

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^4");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}

static void ZCHK5(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
    int nalf, Complex[] alf, int ninc, int[] inc, int nmax, int incmax, ref Complex[,] a, ref Complex[] aa, Complex[] _as, ref Complex[] x, ref
Complex[] xx, Complex[] xs, ref Complex[] y,
    ref Complex[] yy, Complex[] ys, Complex[] yt, double[] g, Complex[] z)
{
    // Tests ZHER and ZHPR

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex half = new Complex(0.5, 0.0);
    Complex one = new Complex(1.0, 0.0);
    const double rzero = 0.0;

    /*// Scalar Arguments
    double eps, thresh;
    int incmax, nalf, nidim, ninc, nmax; // nout, ntra;
    StreamWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    Complex[,] a = new Complex[nmax,nmax];
    Complex[] aa = new Complex[nmax*nmax];
    Complex[] alf = new Complex[nalf];
    Complex[] _as = new Complex[nmax*nmax];
    Complex[] x = new Complex[nmax];
    Complex[] xs = new Complex[nmax*incmax];
    Complex[] xx = new Complex[nmax*incmax];
    Complex[] y = new Complex[nmax];
    Complex[] ys = new Complex[nmax*incmax];
    Complex[] yt = new Complex[nmax];
    Complex[] yy = new Complex[nmax*incmax];
    Complex[] z = new Complex[nmax];

    double[] g = new double[nmax];
    int[] idim = new int[nidim];
    int[] inc = new int[ninc]; */

    // Local Scalars
    Complex alpha, transl;
    double err, errmax, ralpha, rals;
    int i, ia, ic, _in, incx, incxs, ix, j, ja, jj, laa, lda, ldas, lj, lx, n, nargs, nc, nd, ns;
    //int incx, incxs, ja, jj, laa, lda, ldas, lj, lx, n, nargs, nc, nd, ns;
    bool full, _null, packed, reset, same, upper;
    string uplo, uplos, ich;

```

```

// Local Arrays
Complex[] w = new Complex[1];
bool[] isame = new bool[13];
/*
 * .. Scalars in Common ..
 *   INTEGER          INFOT, NOUTC
 *   LOGICAL          LERR, OK
 * .. Common blocks ..
 *   COMMON           /INFOC/INFOT, NOUTC, OK, LERR */

Complex[,] tmp2d_x, tmp2d_aa, tmp2d;
Complex[] tmp1d;
err = 0.0; // added
ich = "UL";

// Executable Statements
full = (sname.ToUpper().Substring(2, 1) == "E");
packed = (sname.ToUpper().Substring(2, 1) == "P");

// Define the number of arguments
if (full)
{
    nargs = 7;
}
else if (packed)
{
    nargs = 6;
}
else
{
    nargs = 0; // added
}

nc = 0;
reset = true;
errmax = rzero;

//100
for (_in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];

    // Set LDA to 1 more than minimum value if room
    lda = n;
    if (lda < nmax)
    {
        lda = lda + 1;
    }

    // Skip tests if not enough room
    if (lda > nmax)
    {
        //GO TO 100
        goto Loop100; //continue?
    }
    if (packed)
    {
        laa = (n * (n + 1)) / 2;
    }
    else
    {
        laa = lda * n;
    }

    //90
    for (ic = 1; ic <= 2; ic = ic + 1)
    {
        uplo = ich.Substring(ic - 1, 1);
        upper = (uplo == "U");

        //80
        for (ix = 1; ix <= ninc; ix = ix + 1)
        {
            incx = inc[ix - 1];
            lx = Math.Abs(incx) * n;

            // Generate the vector X
            transl = half;
            tmp2d_x = zget_2dfromld(nmax, 0, 1, x);
            ZMAKE("GE", " ", n, n, 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
            //KE("GE", ' ', ' ', 1, N, X, 1, XX, ABS( INCX ), 0, N - 1, RESET, TRANSL )
            x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
            if (n > 1)
            {
                x[(n / 2) - 1] = zero;
                xx[(1 + Math.Abs(incx) * (n / 2 - 1)) - 1] = zero;
            }
        }

        //70
        for (ia = 1; ia <= nalf; ia = ia + 1)
        {
            ralpha = alf[ia - 1].Real;
            alpha = new Complex(ralpha, rzero);
            _null = (n <= 0) || (ralpha == rzero);

            // Generate the matrix A
            transl = zero;

            ZMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, n - 1, n - 1, ref reset, transl);

            nc = nc + 1;

            // Save every datum before calling the subroutine
            uplos = uplo;
            ns = n;
            rals = ralpha;
            for (i = 1; i <= laa; i = i + 1)
            {
                _as[i - 1] = aa[i - 1];
            }

            ldas = lda;

```

```

for (i = 1; i <= lx; i = i + 1)
{
    xs[i - 1] = xx[i - 1];
}
incxs = incx;

// Call the subroutine
if (full)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .", nc,
sname, uplo, n, ralpha, incx, lda);
    }

    if (rewi)
    {
        //REWIND NTRA
    }

    tmp2d_aa = zget_2dfromld(nmax * nmax, 0, lda, aa);
    ZHER(uplo, n, ralpha, xx, incx, ref tmp2d_aa, lda);
    aa = zget_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
}

else if (packed)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc, sname,
uplo, n, ralpha, incx);
    }

    if (rewi)
    {
        //REWIND NTRA
    }

    ZHPR(uplo, n, ralpha, xx, incx, ref aa);
}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .", nc,
sname, uplo, n, ralpha, incx, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc, sname,
uplo, n, ralpha, incx);
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (uplo == uplo);
isame[1] = (ns == n);
isame[2] = (rals == ralpha);
isame[3] = LZE(xs, xx, lx);
isame[4] = (incxs == incx);

if (_null)
{
    isame[5] = LZE(_as, aa, laa);
}
else
{
    isame[5] = LZERES(sname.Substring(1, 2), uplo, n, n, zget_2dfromld(nmax * nmax, 0, lda, _as), zget_2dfromld(nmax * nmax, 0,
lda, aa), lda);
}

if (!packed)
{
    isame[6] = (ldas == lda);
}

// If data was incorrectly changed, report and return
same = true;
for (i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}

if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D})      .", nc,
sname, uplo, n, ralpha, incx, lda);
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)      .", nc, sname,
uplo, n, ralpha, incx);
    }
    return;
}

if (!_null)
{
    // Check the result column by column

```

```

        if (incx > 0)
        {
            for (i = 1; i <= n; i = i + 1)
            {
                z[i - 1] = x[i - 1];
            }
        }
        else
        {
            for (i = 1; i <= n; i = i + 1)
            {
                z[i - 1] = x[(n - i + 1) - 1];
            }
        }

        ja = 1;
        //60
        for (j = 1; j <= n; j = j + 1)
        {
            w[0] = Complex.Conjugate(z[j - 1]);
            if (upper)
            {
                jj = 1;
                lj = j;
            }
            else
            {
                jj = j;
                lj = n - j + 1;
            }

            ZMVCH("N", lj, 1, alpha, zget_2dfromld(nmax, jj - 1, lj, z), lj, w, 1, one, get_zarr2d(jj - 1, nmax - 1, j - 1, a),
                1, ref yt, ref g, get_zarr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);

            if (full)
            {
                if (upper)
                {
                    ja = ja + lda;
                }
                else
                {
                    ja = ja + lda + 1;
                }
            }
            else
            {
                ja = ja + lj;
            }
            errmax = Math.Max(errmax, err);
            //nout.WriteLine("{0} (1,18:F5)", fatal, errmax);
            // If got really bad answer, report and return
            if (fatal)
            {
                //GO TO 110
                nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
                nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
                if (full)
                {
                    nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, A, {6,3:D}}", nc, sname, uplo, n, ralpha, incx, lda);
                }
                else if (packed)
                {
                    nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,3:D}, {4,4:F1}, X, {5,2:D}, AP)", nc, sname, uplo, n, ralpha, incx);
                }
            }
            return;
        }
        //60
    }
    else
    {
        // Avoid repeating tests if N.le.0.
        if (n <= 0)
        {
            //GO TO 100
            goto Loop100;
        }
        /*if (errmax < thresh)
        {
            //nout.WriteLine("^^^5");
            nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
        }
        else
        {
            nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO
(2,8:F2) - SUSPECT *****", sname, nc, errmax);
        }
        return;*/
    }
}
} //70
} //80
} //90
Loop100: ;
} //100

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^6");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
}
}
}

```

```

        static void ZCHK6(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int nidim,
int[] idim,
        int nalf, Complex[] alf, int ninc, int[] inc, int nmax, int incmax, ref Complex[,] a, ref Complex[] aa, Complex[] _as, ref Complex[] x, ref
Complex[] xx, Complex[] xs, ref Complex[] y,
        ref Complex[] yy, Complex[] ys, Complex[] yt, double[] g, Complex[,] z)
    {
        // Tests ZHER2 and ZHPR2

        // Parameters
        Complex zero = new Complex(0.0, 0.0);
        Complex half = new Complex(0.5, 0.0);
        Complex one = new Complex(1.0, 0.0);
        const double rzero = 0.0;
        // Scalar Arguments
        /*double eps, thresh;
        int incmax, nalf, nidim, ninc, nmax; // nout, ntra;
        StreamWriter nout, ntra;
        bool fatal, rewi, trace;
        string sname;*/

        // Array Arguments
        /*Complex[,] a = new Complex[nmax,nmax];
        Complex[] aa = new Complex[nmax*nmax];
        Complex[] alf = new Complex[nalf];
        Complex[] _as = new Complex[nmax*nmax];
        Complex[] x = new Complex[nmax];
        Complex[] xs = new Complex[nmax*incmax];
        Complex[] xx = new Complex[nmax*incmax];
        Complex[] y = new Complex[nmax];
        Complex[] ys = new Complex[nmax*incmax];
        Complex[] yt = new Complex[nmax];
        Complex[] yy = new Complex[nmax*incmax];
        Complex[,] z = new Complex[nmax,2];

        double[] g = new double[nmax];
        int[] idim = new int[nidim];
        int[] inc = new int[ninc];*/

        // Local Scalars
        Complex alpha, als, transl;
        double err, errmax;
        /*int i, ia, ic, _in, incx, incxs, incy, incys, ix, iy, j, ja, jj, laa, lda, ldas, lj, lx, ly, n, nargs, nc, ns;
        int incx, incxs, incy, incys, ja, jj, laa, lda, ldas, lj, lx, ly, n, nargs, nc, ns;
        bool full, _null, packed, reset, same, upper;
        string uplo, uplos, ich;

        // Local Arrays
        Complex[] w = new Complex[2];
        bool[] isame = new bool[13];

        /*
        .. Scalars in Common ..
        INTEGER          INFOT, NOUTC
        LOGICAL          LERR, OK
        .. Common blocks ..
        COMMON            /INFOC/INFOT, NOUTC, OK, LERR */

        Complex[,] tmp2d_x, tmp2d_y, tmp2d_aa;

        err = 0.0; // added

        ich = "UL";

        // Executable Statements
        full = (sname.Substring(2, 1) == "E");
        packed = (sname.Substring(2, 1) == "P");

        // Define the number of arguments
        if (full)
        {
            nargs = 9;
        }
        else if (packed)
        {
            nargs = 8;
        }
        else
        {
            nargs = 0; // added
        }

        nc = 0;
        reset = true;
        errmax = rzero;

        //140
        for (int _in = 1; _in <= nidim; _in = _in + 1)
        {
            n = idim[_in - 1];
            // Set LDA to 1 more than minimum value if room
            lda = n;
            if (lda < nmax)
            {
                lda = lda + 1;
            }

            // Skip tests if not enough room
            if (lda > nmax)
            {
                //GO TO 140
                // Report result
                if (errmax < thresh)
                {
                    //nout.WriteLine("^^^7");
                    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
                }
                else
                {
                    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT
*****", sname, nc, errmax);
                }
                return;
            }
        }
    }

```

```

if (packed)
{
    laa = (n * (n + 1)) / 2;
}
else
{
    laa = lda * n;
}

//130
for (int ic = 1; ic <= 2; ic = ic + 1)
{
    uplo = ich.ToUpper().Substring(ic - 1, 1);
    upper = (uplo == "U");

    //120
    for (int ix = 1; ix <= ninc; ix = ix + 1)
    {
        incx = inc[ix - 1];
        lx = Math.Abs(incx) * n;

        // Generate the vector X
        transl = half;
        tmp2d_x = zget_2dfromld(nmax, 0, 1, x);
        ZMAKE("GE", " ", " ", 1, n, ref tmp2d_x, 1, ref xx, Math.Abs(incx), 0, n - 1, ref reset, transl);
        x = zget_ldfrom2d(nmax, 0, 1, tmp2d_x);
        if (n > 1)
        {
            x[(n / 2) - 1] = zero;
            xx[(1 + Math.Abs(incx) * (n / 2 - 1)) - 1] = zero;
        }

        //110
        for (int iy = 1; iy <= ninc; iy = iy + 1)
        {
            incy = inc[iy - 1];
            ly = Math.Abs(incy) * n;

            // Generate the vector Y
            transl = zero;
            tmp2d_y = zget_2dfromld(nmax, 0, 1, y);
            ZMAKE("GE", " ", " ", 1, n, ref tmp2d_y, 1, ref yy, Math.Abs(incy), 0, n - 1, ref reset, transl);
            y = zget_ldfrom2d(nmax, 0, 1, tmp2d_y);
            if (n > 1)
            {
                y[(n / 2) - 1] = zero;
                yy[(1 + Math.Abs(incy) * (n / 2 - 1)) - 1] = zero;
            }

            //100
            for (int ia = 1; ia <= nalf; ia = ia + 1)
            {
                alpha = alf[ia - 1];
                _null = ((n <= 0) || (alpha == zero));

                // Generate the matrix A
                transl = zero;
                ZMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref a, nmax, ref aa, lda, n - 1, n - 1, ref reset, transl);

                nc = nc + 1;

                // Save every datum before calling the subroutine
                uplos = uplo;
                ns = n;
                als = alpha;
                for (int i = 1; i <= laa; i = i + 1)
                {
                    _as[i - 1] = aa[i - 1];
                }
                ldas = lda;
                for (int i = 1; i <= lx; i = i + 1)
                {
                    xs[i - 1] = xx[i - 1];
                }
                incxs = incx;
                for (int i = 1; i <= ly; i = i + 1)
                {
                    ys[i - 1] = yy[i - 1];
                }
                incys = incy;

                // Call the subroutine
                if (full)
                {
                    if (trace)
                    {
                        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})      .",
                            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy, lda); //4 {,4:F1}
                    }
                    if (rewi)
                    {
                        //REWIND NTRA
                    }

                    tmp2d_aa = zget_2dfromld(nmax * nmax, 0, lda, aa);
                    ZHER2(uplo, n, alpha, xx, incx, yy, incy, ref tmp2d_aa, lda);
                    aa = zget_ldfrom2d(nmax * nmax, 0, lda, tmp2d_aa);
                }
            }
        }
    }
else if (packed)
{
    if (trace)
    {
        ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, AP)      .",
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy); //4 {,4:F1}
    }
    if (rewi)
    {
        //REWIND NTRA
    }
    ZHPR2(uplo, n, alpha, xx, incx, yy, incy, ref aa);
}
}

```

```

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 160
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D}) .",
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy, lda); // 4 {,4:F1}
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, AP) .",
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy); // 4 {,4:F1}
    }
    return;
}

// See what data changed inside subroutines
isame[0] = (uplo == uplos);
isame[1] = (ns == n);
isame[2] = (als == alpha);
isame[3] = LZE(xs, xx, lx);
isame[4] = (incxs == incx);
isame[5] = LZE(ys, yy, ly);
isame[6] = (incys == incy);

if (_null)
{
    isame[7] = LZE(_as, aa, laa);
}
else
{
    isame[7] = LZERES(sname.Substring(1, 2), uplo, n, n, zget_2dfromld(nmax * nmax, 0, lda, _as), zget_2dfromld(nmax *
nmax, 0, lda, aa), lda);
}

if (!packed)
{
    isame[8] = (ldas == lda);
}

// If data was incorrectly changed, report and return
same = true;
//40
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}

if (!same)
{
    fatal = true;
    //GO TO 160
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (full)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D}) .",
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy, lda); // 4 {,4:F1}
    }
    else if (packed)
    {
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, AP) .",
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy); // 4 {,4:F1}
    }
    return;
}

if (!_null)
{
    // Check the result column by column
    if (incx > 0)
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 0] = x[i - 1];
        }
    }
    else
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 0] = x[(n - i + 1) - 1];
        }
    }
    if (incy > 0)
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 1] = y[i - 1];
        }
    }
    else
    {
        for (int i = 1; i <= n; i = i + 1)
        {
            z[i - 1, 1] = y[(n - i + 1) - 1];
        }
    }
    ja = 1;
    //90
    for (int j = 1; j <= n; j = j + 1)
    {
        w[0] = Complex.Multiply(alpha, Complex.Conjugate(z[j - 1, 1]));
        w[1] = Complex.Multiply(Complex.Conjugate(alpha), Complex.Conjugate(z[j - 1, 0]));
        if (upper)
        {

```

```

        jj = 1;
        lj = j;
    }
    else
    {
        jj = j;
        lj = n - j + 1;
    }

    ZMVCH("N", lj, 2, one, zget_2dfrom1d(nmax * 2, 0, nmax, zget_ldfrom2d(nmax * 2, jj - 1, nmax, z)), nmax,
    w, 1, one, get_zarr2d(jj - 1, nmax - 1, j - 1, a), 1,
    ref yt, ref g, get_zarr(ja - 1, (nmax * nmax) - 1, aa), eps, ref err, ref fatal, nout, true);
    //ZMVCH('N', LJ, 2, ONE, Z(JJ, 1), NMAX, W, 1, ONE, A(JJ, J), 1, YT, G, AA(JA), EPS, ERR, FATAL, NOUT, .TRUE.
)

    if (full)
    {
        if (upper)
        {
            ja = ja + lda;
        }
        else
        {
            ja = ja + lda + 1;
        }
    }
    else
    {
        ja = ja + lj;
    }
    errmax = Math.Max(errmax, err);

    // If got really bad answer, report and return
    if (fatal)
    {
        //GO TO 150
        nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (full)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, A, {7,3:D})
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy, lda); // 4 { ,4:F1}

        }
        else if (packed)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,3:D}, {4}, X, {5,2:D}, Y, {6,2:D}, AP
            nc, sname, uplo, n, alpha.ToString("F1"), incx, incy); // 4 { ,4:F1}

        }
        return;
    }
} //90
}
else
{
    // Avoid repeating tests with N.le.0
    if (n <= 0)
    {
        //GO TO 140
        goto Loop140;
        /* // Report result
        if (errmax < thresh)
        {
            //nout.WriteLine("^^^8");
            nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
        }
        else
        {
            nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO
(2,8:F2) - SUSPECT *****", sname, nc, errmax);
        }
        return;*/
    }
}
} //100
} //110
} //120
} //130
Loop140: ;
} //140

// Report result
if (errmax < thresh)
{
    //nout.WriteLine("^^^9");
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO (2,8:F2) - SUSPECT *****",
sname, nc, errmax);
}
return;
}

static void ZCHKE(int isnum, string sramt, StreamWriter nout)
{
    // Tests the error exits from the Level 2 Blas. Requires a special version of the error-handling routine XERBLA. ALPHA, RBETA, A, X and Y
    should not need to be defined.
    // Auxiliary routine for test program for Level 2 Blas.

    // Local Scalars
    Complex alpha, beta;
    double ralpha;

    // Local Arrays
    Complex[,] a = new Complex[1, 1];
    Complex[] x = new Complex[1];
    Complex[] y = new Complex[1];

    Complex[] tmp1d_a;

    alpha = 0.0; //added
    beta = 0.0; // added

```

```

ralpha = 0.0; //added

//COMMON          /INFOC/INFOT, NOUTC, OK, LERR

// Executable Statements
// OK is set to .FALSE. by the special version of XERBLA or by CHKXER if anything is wrong.
ok = true;

// LERR is set to .TRUE. by the special version of XERBLA each time it is called, and is then tested and re-set by CHKXER
lerr = false;
//nout.WriteLine("DCHKE: isnum={0}", isnum);

switch (isnum)
{
    case 1:
        //nout.WriteLine("^^^infot=1");
        infot = 1;
        ZGEMV("/", 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //nout.WriteLine("^^^infot=2");
        infot = 2;
        ZGEMV("N", -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //nout.WriteLine("^^^infot=3");
        infot = 3;
        ZGEMV("N", 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //nout.WriteLine("^^^infot=6");
        infot = 6;
        ZGEMV("N", 2, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //nout.WriteLine("^^^infot=8");
        infot = 8;
        ZGEMV("N", 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //nout.WriteLine("^^^infot=11");
        infot = 11;
        ZGEMV("N", 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 180
        break;

    case 2:
        infot = 1;
        ZGBMV("/", 0, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        // nout.Flush();
        //nout.Close();
        //Environment.Exit(1);
        infot = 2;
        ZGBMV("N", -1, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 3;
        ZGBMV("N", 0, -1, 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 4;
        ZGBMV("N", 0, 0, -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        ZGBMV("N", 2, 0, 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 8;
        ZGBMV("N", 0, 0, 1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 10;
        ZGBMV("N", 0, 0, 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 13;
        ZGBMV("N", 0, 0, 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 180
        break;

    case 3:
        infot = 1;
        ZHEMV("/", 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        ZHEMV("U", -1, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        ZHEMV("U", 2, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 7;
        ZHEMV("U", 0, alpha, a, 1, x, 0, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 10;
        ZHEMV("U", 0, alpha, a, 1, x, 1, beta, ref y, 0);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        //GO TO 180
        break;

    case 4:
        infot = 1;
        ZHBMV("/", 0, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 2;
        ZHBMV("U", -1, 0, alpha, a, 1, x, 1, beta, ref y, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 3;
        ZHBMV("U", 0, -1, alpha, a, 1, x, 1, beta, ref y, 1);

```

```

misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
ZHBMV("U", 0, 1, alpha, a, 1, x, 1, beta, ref y, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 8;
ZHBMV("U", 0, 0, alpha, a, 1, x, 0, beta, ref y, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 11;
ZHBMV("U", 0, 0, alpha, a, 1, x, 1, beta, ref y, 0);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 5:
infot = 1;
ZHPMV("/", 0, alpha, get_zarr2d(0, 0, 0, a), x, 1, beta, ref y, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
ZHPMV("U", -1, alpha, get_zarr2d(0, 0, 0, a), x, 1, beta, ref y, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
ZHPMV("U", 0, alpha, get_zarr2d(0, 0, 0, a), x, 0, beta, ref y, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
ZHPMV("U", 0, alpha, get_zarr2d(0, 0, 0, a), x, 1, beta, ref y, 0);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 6:
infot = 1;
ZTRMV("/", "N", "N", 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
ZTRMV("U", "/", "N", 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
ZTRMV("U", "N", "/", 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
ZTRMV("U", "N", "N", -1, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 6;
ZTRMV("U", "N", "N", 2, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 8;
ZTRMV("U", "N", "N", 0, a, 1, ref x, 0);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 7:
infot = 1;
ZTBMV("/", "N", "N", 0, 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
ZTBMV("U", "/", "N", 0, 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
ZTBMV("U", "N", "/", 0, 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
ZTBMV("U", "N", "N", -1, 0, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
ZTBMV("U", "N", "N", 0, -1, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
ZTBMV("U", "N", "N", 0, 1, a, 1, ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
ZTBMV("U", "N", "N", 0, 0, a, 1, ref x, 0);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 8:
infot = 1;
ZTPMV("/", "N", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
ZTPMV("U", "/", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 3;
ZTPMV("U", "N", "/", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 4;
ZTPMV("U", "N", "N", -1, get_zarr2d(0, 0, 0, a), ref x, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;

```

```

ZTPMV("U", "N", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 0);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 9:
  infot = 1;
  ZTRSV("/", "N", "N", 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 2;
  ZTRSV("U", "/", "N", 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 3;
  ZTRSV("U", "N", "/", 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 4;
  ZTRSV("U", "N", "N", -1, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 6;
  ZTRSV("U", "N", "N", 2, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 8;
  ZTRSV("U", "N", "N", 0, a, 1, ref x, 0);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
  //GO TO 180
  break;

case 10:
  infot = 1;
  ZTBSV("/", "N", "N", 0, 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 2;
  ZTBSV("U", "/", "N", 0, 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 3;
  ZTBSV("U", "N", "/", 0, 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 4;
  ZTBSV("U", "N", "N", -1, 0, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 5;
  ZTBSV("U", "N", "N", 0, -1, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 7;
  ZTBSV("U", "N", "N", 0, 1, a, 1, ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 9;
  ZTBSV("U", "N", "N", 0, 0, a, 1, ref x, 0);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
  //GO TO 180
  break;

case 11:
  infot = 1;
  ZTPSV("/", "N", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 2;
  ZTPSV("U", "/", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 3;
  ZTPSV("U", "N", "/", 0, get_zarr2d(0, 0, 0, a), ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 4;
  ZTPSV("U", "N", "N", -1, get_zarr2d(0, 0, 0, a), ref x, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 7;
  ZTPSV("U", "N", "N", 0, get_zarr2d(0, 0, 0, a), ref x, 0);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
  //GO TO 180
  break;

case 12:
  infot = 1;
  ZGERC(-1, 0, alpha, x, 1, y, 1, ref a, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 2;
  ZGERC(0, -1, alpha, x, 1, y, 1, ref a, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 5;
  ZGERC(0, 0, alpha, x, 0, y, 1, ref a, 1); // x, 0, y NOT x, 1, y
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 7;
  ZGERC(0, 0, alpha, x, 1, y, 0, ref a, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

  infot = 9;
  ZGERC(2, 0, alpha, x, 1, y, 1, ref a, 1);
  misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
  //GO TO 180
  break;

case 13:
  infot = 1;

```

```

ZGERU(-1, 0, alpha, x, 1, y, 1, ref a, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
ZGERU(0, -1, alpha, x, 1, y, 1, ref a, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
ZGERU(0, 0, alpha, x, 0, y, 1, ref a, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
ZGERU(0, 0, alpha, x, 1, y, 0, ref a, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
ZGERU(2, 0, alpha, x, 1, y, 1, ref a, 1);
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 14:
infot = 1;
//tmp2d_a = new Complex[,] { a[0, 0] };
ZHER("/", 0, ralpha, x, 1, ref a, 1);
//a[0, 0] = tmp2d_a[0];
//public static void ZHER(string uplo, int n, double alpha, Complex[] x, int incx, ref Complex[,] a, int lda
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER("U", -1, ralpha, x, 1, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER("U", 0, ralpha, x, 0, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER("U", 2, ralpha, x, 1, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 15:
infot = 1;
tmp1d_a = new Complex[] { a[0, 0] };
ZHPR("/", 0, ralpha, x, 1, ref tmp1d_a);
a[0, 0] = tmp1d_a[0];
//public static void ZHPR(string uplo, int n, double alpha, Complex[] x, int incx, ref Complex[] ap
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
tmp1d_a = new Complex[] { a[0, 0] };
ZHPR("U", -1, ralpha, x, 1, ref tmp1d_a);
a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
tmp1d_a = new Complex[] { a[0, 0] };
ZHPR("U", 0, ralpha, x, 0, ref tmp1d_a);
a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 16:
infot = 1;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER2("/", 0, alpha, x, 1, y, 1, ref a, 1);
//public static void ZHER2(string uplo, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[,] a, int lda
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER2("U", -1, alpha, x, 1, y, 1, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 5;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER2("U", 0, alpha, x, 0, y, 1, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 7;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER2("U", 0, alpha, x, 1, y, 0, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 9;
//tmp1d_a = new Complex[] { a[0, 0] };
ZHER2("U", 2, alpha, x, 1, y, 1, ref a, 1);
//a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
//GO TO 180
break;

case 17:
infot = 1;
tmp1d_a = new Complex[] { a[0, 0] };
ZHPR2("/", 0, alpha, x, 1, y, 1, ref tmp1d_a);
a[0, 0] = tmp1d_a[0];
misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

infot = 2;

```

```

        tmpld_a = new Complex[] { a[0, 0] };
        ZHPR2("U", -1, alpha, x, 1, y, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 5;
        tmpld_a = new Complex[] { a[0, 0] };
        ZHPR2("U", 0, alpha, x, 0, y, 1, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        infot = 7;
        tmpld_a = new Complex[] { a[0, 0] };
        ZHPR2("U", 0, alpha, x, 1, y, 0, ref tmpld_a);
        a[0, 0] = tmpld_a[0];
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        break;
    default:
        break;
}

//180
if (ok)
{
    nout.WriteLine(" {0,6} PASSED THE TESTS OF ERROR-EXITS", srnamt);
}
else
{
    nout.WriteLine("***** {0,6} FAILED THE TESTS OF ERROR-EXITS *****", srnamt);
}
return;
}
public static void XERBLA(string srname, int info)
{
    // This is a special version of XERBLA to be used only as part of the test program for testing error exits from the Level 2 BLAS routines.
    // XERBLA is an error handler for the Level 2 BLAS routines. It is called by the Level 2 BLAS routines if an input parameter is invalid.
    /* .. Scalars in Common ..
        INTEGER          INFOT, NOUT
        LOGICAL          LERR, OK
        CHARACTER*6      SRNAMT*/

    // Executable Statements
    lerr = true;
    //nout.WriteLine("in:srname={0} info={1} :infot={2} srnamt={3}", srname, info, infot, srnamt);
    if (info != infot)
    {
        if (infot != 0)
        {
            nout.WriteLine("***** XERBLA WAS CALLED WITH INFO = {0,6:D} INSTEAD OF {1,2:D} *****", info, infot);
        }
        else
        {
            nout.WriteLine("***** XERBLA WAS CALLED WITH INFO = {0,6:D} *****", info);
        }
        ok = false;
    }

    if (srname != srnamt)
    {
        nout.WriteLine("***** XERBLA WAS CALLED WITH SRNAME = {0,6} INSTEAD OF {1,6} *****", srname, srnamt);
        ok = false;
    }
}

static void ZMAKE(string type, string uplo, string diag, int m, int n, ref Complex[,] a, int nmax, ref Complex[] aa, int lda, int kl, int ku, ref
bool reset, Complex transl)
{
    // Generates values for an M by N matrix A within the bandwidth defined by KL and KU. Stores the values in the array AA in the data structure
    required
    // by the routine, with unwanted elements set to rogue value.
    //
    // TYPE is 'GE', 'GB', 'HE', 'HB', 'HP', 'TR', 'TB' OR 'TP'.

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex one = new Complex(1.0, 0.0);
    Complex rogue = new Complex(-1e10, 1e10);
    const double rzero = 0.0;
    const double rrogue = -1e10;

    /**/ Scalar Arguments
    Complex transl;
    int kl, ku, lda, m, n, nmax;
    bool reset;
    string diag, uplo, type;

    // Array Arguments
    //COMPLEX*16 A( NMAX, * ), AA( * )
    */
    // Local Scalars
    //int i, il, i2, i3, ibeg, iend, ioff, j, kk;
    int il, i2, i3, ibeg, iend, ioff, jj, kk;

    bool gen, lower, sym, tri, unit, upper;

    // Executable Statements
    gen = (type.ToUpper().Substring(0, 1) == "G");
    sym = (type.ToUpper().Substring(0, 1) == "H");
    tri = (type.ToUpper().Substring(0, 1) == "T");
    upper = ((sym || tri) && (uplo.ToUpper().Substring(0, 1) == "U"));
    lower = ((sym || tri) && (uplo.ToUpper().Substring(0, 1) == "L"));
    unit = (tri && (diag == "U"));

    // Generate data in array A
    //20
    for (int j = 1; j <= n; j = j + 1)
    {
        //10
        //Console.WriteLine("j={0} < {1}", j, n);
        for (int i = 1; i <= m; i = i + 1)
        {
            //Console.WriteLine("i={0} < {1}", j, m);
            if (gen || upper && (i <= j)) || (lower && (i >= j)))
            {
                if (((i <= j) && ((j - i) <= ku)) || ((i >= j) && ((i - j) <= kl)))

```

```

        {
            a[i - 1, j - 1] = ZBEG(ref reset) + transl;
        }
        else
        {
            a[i - 1, j - 1] = zero;
        }
    }
    if (i != j)
    {
        if (sym)
        {
            a[j - 1, i - 1] = Complex.Conjugate(a[i - 1, j - 1]);
        }
        else if (tri)
        {
            a[j - 1, i - 1] = zero;
        }
    }
}
//Console.WriteLine("end i={0} < {1}", j, m);
} //10
if (sym)
{
    a[j - 1, j - 1] = new Complex(a[j - 1, j - 1].Real, rzero);
}
if (tri)
{
    a[j - 1, j - 1] = a[j - 1, j - 1] + one;
}
if (unit)
{
    a[j - 1, j - 1] = one;
}
//Console.WriteLine("end j={0} < {1}", j, n);
} //20

//Console.WriteLine("done main j");
// Store elements in array AS in data structure required by routine
if (type == "GE")
{
    //50
    for (int j = 1; j <= n; j = j + 1)
    {
        //Console.WriteLine("j2={0} < {1}", j, n);
        for (int i = 1; i <= m; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
        }
        for (int i = m + 1; i <= lda; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
    }
}
else if (type == "GB")
{
    //Console.WriteLine("nmax={0}", nmax);
    //90
    for (int j = 1; j <= n; j = j + 1)
    {
        //Console.WriteLine("j={0} < {1}", j, n);
        //60
        for (i1 = 1; i1 <= (ku + 1 - j); i1 = i1 + 1)
        {
            //Console.WriteLine("i1={0} < {1}", i1, ku+1-j);
            aa[(i1 + (j - 1) * lda) - 1] = rogue;
        }
        //70
        //for (int i2 = i1; i2 <= Math.Min(kl + ku + 1, ku + 1 + m - j); i2 = i2 + 1) {
        for (i2 = i1; i2 <= Math.Min(kl + ku + 1, ku + 1 + m - j); i2 = i2 + 1)
        {
            //Console.WriteLine("i2={0} < {1}", i2, Math.Min(kl + ku + 1, ku + 1 + m - j));
            aa[(i2 + (j - 1) * lda) - 1] = a[(i2 + j - ku - 1) - 1, j - 1];
        }
        //80
        for (i3 = i2; i3 <= lda; i3 = i3 + 1)
        {
            //Console.WriteLine("i3={0} < {1}", i3, lda);
            aa[(i3 + (j - 1) * lda) - 1] = rogue;
        }
    }
}
}
else if ((type == "HE") || (type == "TR"))
{
    //130
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            {
                ibeg = 1;
                if (unit)
                {
                    iend = j - 1;
                }
                else
                {
                    iend = j;
                }
            }
        }
        else
        {
            {
                if (unit)
                {
                    ibeg = j + 1;
                }
                else
                {
                    ibeg = j;
                }
            }
            iend = n;
        }
        for (int i = 1; i <= (ibeg - 1); i = i + 1)
        {

```

```

        aa[(i + (j - 1) * lda) - 1] = rogue;
    }
    for (int i = ibeg; i <= iend; i = i + 1)
    {
        aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
    }
    for (int i = (iend + 1); i <= lda; i = i + 1)
    {
        aa[(i + (j - 1) * lda) - 1] = rogue;
    }
    if (sym)
    {
        jj = j + (j - 1) * lda;
        aa[jj - 1] = new Complex(aa[jj - 1].Real, rrogue);
    }
} //130

}
else if ((type == "HB") || (type == "TB"))
{
    //170
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            {
                kk = kl + 1;
                ibeg = Math.Max(1, kl + 2 - j);
                if (unit)
                {
                    iend = kl;
                }
                else
                {
                    iend = kl + 1;
                }
            }
        }
        else
        {
            kk = 1;
            if (unit)
            {
                ibeg = 2;
            }
            else
            {
                ibeg = 1;
            }
            iend = Math.Min(kl + 1, 1 + m - j);
        }
        for (int i = 1; i <= (ibeg - 1); i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
        for (int i = ibeg; i <= iend; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = a[(i + j - kk) - 1, j - 1];
        }
        for (int i = (iend + 1); i <= lda; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
        if (sym)
        {
            jj = kk + (j - 1) * lda;
            aa[jj - 1] = new Complex(aa[jj - 1].Real, rrogue);
        }
    }
} //170

}
else if ((type == "HP") || (type == "TP"))
{
    ioff = 0;
    //190
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            {
                ibeg = 1;
                iend = j;
            }
        }
        else
        {
            {
                ibeg = j;
                iend = n;
            }
        }
        //180
        for (int i = ibeg; i <= iend; i = i + 1)
        {
            ioff = ioff + 1;
            aa[ioff - 1] = a[i - 1, j - 1];
            if (i == j)
            {
                {
                    if (unit)
                    {
                        aa[ioff - 1] = rogue;
                    }
                    if (sym)
                    {
                        aa[ioff - 1] = new Complex(aa[ioff - 1].Real, rrogue);
                    }
                }
            }
        }
    }
}

}
static double ABS1(Complex c)
{
    return Math.Abs(c.Real) + Math.Abs(c.Imaginary);
}
static void ZMVCH(string trans, int m, int n, Complex alpha, Complex[,] a, int nmax, Complex[] x, int incx, Complex beta, Complex[] y, int incy,
ref Complex[] yt,
ref double[] g, Complex[] yy, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv)
{
    // Checks the results of the computational tests

```

```

// Parameters
Complex zero = new Complex(0.0, 0.0);
//const Complex one = new Complex(1.0,0.0);
const double rzero = 0.0;
const double rone = 1.0;

// Array Arguments ..
//DOUBLE PRECISION  A( NMAX, * ), G( * ), X( * ), Y( * ), YT( * ), YY( * )

// Local Scalars
Complex c;
double erri;
//int i, incxl, incyl, iy, j, jx, kx, ky, ml, nl;
int incxl, incyl, iy, jx, kx, ky, ml, nl;
bool ctran, tran;

//ABS1( C ) = ABS( DBLE( C ) ) + ABS( DIMAG( C ) )

// Executable Statements
//nout.WriteLine("Rank:{0,4:D} D1:{1,4:D} D2:{2,4:D} NMAX:{3,4:D}", a.Rank, a.GetLength(0), a.GetLength(1), nmax);
/*for (int i = 1; i <= nmax; i = i + 1)
{
    nout.WriteLine("{0,4:D}{1,18:F5}", i, a[i - 1, 0]);
}*/

//tran = (trans.ToUpper().Substring(0,1) == "T") || (trans.ToUpper().Substring(0,1) == "C");
tran = (trans.ToUpper().Substring(0, 1) == "T");
ctran = (trans.ToUpper().Substring(0, 1) == "C");
if (tran || ctran)
{
    ml = n;
    nl = m;
}
else
{
    ml = m;
    nl = n;
}
if (incx < 0)
{
    kx = nl;
    incxl = -1;
}
else
{
    kx = 1;
    incxl = 1;
}
if (incy < 0)
{
    ky = ml;
    incyl = -1;
}
else
{
    ky = 1;
    incyl = 1;
}

// Compute expected result in YT using data in A, X and Y.  Compute gauges in G.
iy = ky;
//30
/*nout.WriteLine("DMVCH: {0}",tran);
nout.WriteLine("DMVCH: {0,4:D}", iy);
*/
for (int i = 1; i <= ml; i = i + 1)
{
    yt[iy - 1] = zero;
    g[iy - 1] = rzero;
    jx = kx;
    if (tran)
    {
        for (int j = 1; j <= nl; j = j + 1)
        {
            yt[iy - 1] = yt[iy - 1] + a[j - 1, i - 1] * x[jx - 1];
            //g[iy - 1] = g[iy - 1] + Math.Abs(a[j - 1, i - 1] * x[jx - 1]);
            g[iy - 1] = g[iy - 1] + ABS1(a[j - 1, i - 1]) * ABS1(x[jx - 1]);
            //nout.WriteLine("DMVCH: {0,4:D}{1,4:D}{2,18:F5}{3,18:F5}{4,18:F5}{5,18:F5}", i, j, yt[iy - 1], g[iy - 1], a[j-1,i-1],x[jx-1]);
            jx = jx + incxl;
        }
    }
    else if (ctran)
    {
        for (int j = 1; j <= nl; j = j + 1)
        {
            yt[iy - 1] = yt[iy - 1] + Complex.Conjugate(a[j - 1, i - 1]) * x[jx - 1];
            g[iy - 1] = g[iy - 1] + ABS1(a[j - 1, i - 1]) * ABS1(x[jx - 1]);
            jx = jx + incxl;
        }
    }
    else
    {
        for (int j = 1; j <= nl; j = j + 1)
        {
            yt[iy - 1] = yt[iy - 1] + a[i - 1, j - 1] * x[jx - 1];
            g[iy - 1] = g[iy - 1] + ABS1(a[i - 1, j - 1]) * ABS1(x[jx - 1]);
            //nout.WriteLine("DMVCH: {0,4:D}{1,4:D}{2,18:F5}{3,18:F5}{4,18:F5}{5,18:F5}", i, j, yt[iy - 1], g[iy - 1], a[i-1,j-1],x[jx-1]);
            jx = jx + incxl;
        }
    }
    yt[iy - 1] = alpha * yt[iy - 1] + beta * y[iy - 1];
    g[iy - 1] = ABS1(alpha) * g[iy - 1] + ABS1(beta) * ABS1(y[iy - 1]);

    iy = iy + incyl;
}
//for (int i = 1; i <= ml; i = i + 1)

```

```

    //{
    //nout.WriteLine("i={0} yt={1} yy={2}", i, yt[i - 1], yy[i - 1]);
    //}
    //nout.Flush();
    //nout.Close();
    // Environment.Exit(1);
    // Compute the error ratio for this result
    err = rzero; // was zero but is complex

    for (int i = 1; i <= ml; i = i + 1)
    {
        //erri = Math.Abs(yt[i - 1] - yy[(1 + (i - 1) * Math.Abs(incy)) - 1]) / eps;
        erri = (yt[i - 1] - yy[(1 + (i - 1) * Math.Abs(incy)) - 1]).Real / eps;
        //nout.WriteLine("DMVCH: {0,4:D}{1,4:D}{2,18:F5}{3,18:F5}{4,18:F5}{5,18:F5}", i, 1 + (i - 1) * Math.Abs(incy), erri, yt[i - 1], yy[1 + (i -
1) * Math.Abs(incy) - 1], g[i - 1]);
        if (misc_z.myflag)
        {
            //nout.WriteLine("DMVCH:{0,7:D} {1,7:D} {2,10:F3} {3,10:F3} {4,10:F3}", i, 1 + (i - 1) * Math.Abs(incy), yt[i - 1], yy[1 + (i - 1) *
Math.Abs(incy) - 1], erri);

            //WRITE(NOUT,*) 'DMVCH',I,1+(I-1)*ABS(INCY),YT(I),YY(1+(I-1)*ABS(INCY)),ERRI
            nout.WriteLine("{0,10:F3} {1,10:F3}", g[i - 1], erri);
        }
        if (g[i - 1] != rzero)
        {
            erri = erri / g[i - 1];
        }
        err = Math.Max(err, erri);
        //if (misc_z.myflag)
        //{
        //nout.WriteLine("{0,7:D} {1,10:F3} {2,10:F3} {3,10:F3}", i, err, eps, err * Math.Sqrt(eps));
        //}
        if ((err * Math.Sqrt(eps)) >= rone)
        {
            //GO TO 60

            // Report fatal error
            fatal = true;
            nout.WriteLine("***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HALF ACCURATE *****\n
            EXPECTED RESULT      COMPUTED

RESULT");

            for (int ii = 1; ii <= ml; ii = ii + 1)
            {
                if (mv)
                {
                    nout.WriteLine(" {0,7:D}, {1,18:G6} {2,18:G6}", ii, yt[ii - 1], yy[(1 + (ii - 1) * Math.Abs(incy)) - 1]);
                }
                else
                {
                    nout.WriteLine(" {0,7:D}, {1,18:G6} {2,18:G6}", ii, yy[(1 + (ii - 1) * Math.Abs(incy)) - 1], yt[ii - 1]);
                }
            }
            return;
        }
    }

    //if (misc_d.myflag)
    {
        nout.Flush();
        nout.Close();
        Environment.Exit(1);
    }
    // If the loop completes, all results are at least half accurate.
    return;
}

static bool LZE(Complex[] ri, Complex[] rj, int lr)
{
    // Tests if two arrays are identical

    // Scalar Arguments
    //int lr;

    // Local Scalars
    //int i;
    bool lze;
    // Executable Statements
    for (int i = 1; i <= lr; i = i + 1)
    {
        if (ri[i - 1] != rj[i - 1])
        {
            //GO TO 20
            lze = false;
            return lze;
        }
    }
    lze = true;
    return lze;
}

static bool LZERS(string type, string uplo, int m, int n, Complex[,] aa, Complex[,] _as, int lda)
{
    // DOUBLE PRECISION AA( LDA, * ), AS( LDA, * )
    // Tests if selected elements in two arrays are equal
    // TYPE is 'GE', 'HE' or 'HP'

    // Local Scalars
    //int i, ibeg, iend, j;
    bool lzeres;
    int ibeg, iend;
    bool upper;
    // COMPLEX*16 AA( LDA, * ), AS( LDA, * )
    // Executable Statements
    upper = (uplo == "U");
    if (type == "GE")
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = m + 1; i <= lda; i = i + 1)
            {
                if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
                {
                    //GO TO 70
                    lzeres = false;
                }
            }
        }
    }
}

```

```

        return lzeres;
    }
}

else if (type == "HE")
{
    //50
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            ibeg = 1;
            iend = j;
        }
        else
        {
            ibeg = j;
            iend = n;
        }
        //30
        for (int i = 1; i <= ibeg - 1; i = i + 1)
        {
            if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
            {
                //GO TO 70
                lzeres = false;
                return lzeres;
            }
        }
        //40
        for (int i = iend + 1; i <= lda; i = i + 1)
        {
            if (aa[i - 1, j - 1] != _as[i - 1, j - 1])
            {
                //GO TO 70
                lzeres = false;
                return lzeres;
            }
        }
    } //50
}

lzeres = true;
return lzeres;
}
// duplicate of one in DBLAT2 , access as misc_d.CHKXER
/*static void CHKXER(string srnamt, int infot, StreamWriter nout, bool lerr, bool ok)
{
    // Tests whether XERBLA has detected an error when it should

    if (!lerr)
    {
        nout.WriteLine("***** ILLEGAL VALUE OF PARAMETER NUMBER {0,2:D} NOT DETECTED BY {1,6} *****", infot, srnamt);
        ok = false;
    }
    lerr = false;
}*/
static Complex ZBEG(ref bool reset)
{
    // Generates complex random numbers uniformly distributed between -0.5 and 0.5

    // Local Scalars
    //static int i, ic, j, mi, mj; see above

    // Executable Statements
    if (reset)
    {
        // Initialize local variables
        mi = 891;
        mj = 457;
        i = 7;
        j = 7;
        ic = 0;
        reset = false;
    }

    // The sequence of values of I or J is bounded between 1 and 999.
    // If initial I or J = 1,2,3,6,7 or 9, the period will be 50.
    // If initial I or J = 4 or 8, the period will be 25.
    // If initial I or J = 5, the period will be 10.
    // IC is used to break up the period by skipping 1 value of I or J in 6.

    ic = ic + 1;

    do
    {
        //Console.WriteLine("in do loop DBEG ic={0}",ic);
        i = i * mi;
        j = j * mj;
        i = i - 1000 * (i / 1000);
        j = j - 1000 * (j / 1000);
        if (ic >= 5)
        {
            ic = 0;
            continue;
        }
        else
        {
            break;
        }
    } while (true);
    return new Complex((i - 500) / 1001.0, (j - 500) / 1001.0);
}
public static void ZGEMV(string trans, int m, int n, Complex alpha, Complex[,] a, int lda, Complex[] x, int incx, Complex beta, ref Complex[] y,
int incy)
{
    // SUBROUTINE ZGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
    // COMPLEX*16 A(LDA,*),X(*),Y(*)

    // ZGEMV performs one of the matrix-vector operations
    //
    // y := alpha*A*x + beta*y, or y := alpha*A**T*x + beta*y, or y := alpha*A**H*x + beta*y,

```

```

//
// where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
//
// TRANS is CHARACTER*1
// On entry, TRANS specifies the operation to be performed as follows:
//   TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
//   TRANS = 'T' or 't'   y := alpha*A**T*x + beta*y.
//   TRANS = 'C' or 'c'   y := alpha*A**H*x + beta*y.
// On entry, M specifies the number of rows of the matrix A. M must be at least zero.
// On entry, N specifies the number of columns of the matrix A. N must be at least zero.

// ALPHA is COMPLEX*16
// On entry, ALPHA specifies the scalar alpha.

// A is COMPLEX*16 array of DIMENSION ( LDA, n ).
// Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, m ).

otherwise. // X is COMPLEX*16 array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( m - 1 ) * abs( INCX ) )
// Before entry, the incremented array X must contain the vector x.

// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
// BETA is COMPLEX*16
// On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

otherwise. // Y is COMPLEX*16 array of DIMENSION at least ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( n - 1 ) * abs( INCY ) )
// Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

// On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

// Level 2 Blas routine.
// The vector and matrix arguments are not referenced when N = 0, or M = 0

// Parameters
Complex one = new Complex(1.0, 0.0);
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp;
//int i, info, ix, iy, j, jx, jy, kx, ky, lenx, leny;
int info, ix, iy, jx, jy, kx, ky, lenx, leny;
bool noconj;

// Test the input parameters
info = 0;
if ((!(trans.ToUpper().Substring(0, 1) == "N")) && (!(trans.ToUpper().Substring(0, 1) == "T")) &&
    (!(trans.ToUpper().Substring(0, 1) == "C"))))
{
    info = 1;
}
else if (m < 0)
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (lda < Math.Max(1, m))
{
    info = 6;
}
else if (incx == 0)
{
    info = 8;
}
else if (incy == 0)
{
    info = 11;
}
if (info != 0)
{
    XERBLA("ZGEMV ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");

// Set LENX and LENY, the lengths of the vectors x and y, and set up the start points in X and Y
if (trans.Substring(0, 1).ToUpper() == "N")
{
    lenx = n;
    leny = m;
}
else
{
    lenx = m;
    leny = n;
}
if (incx > 0)
{
    kx = 1;
}
else
{
    kx = 1 - (lenx - 1) * incx;
}
if (incy > 0)
{
    ky = 1;
}
else

```

```

{
    ky = 1 - (leny - 1) * incy;
}
// Start the operations. In this version the elements of A are accessed sequentially with one pass through the band part of A.
// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = beta * y[iy - 1];
                iy = iy + incy;
            }
        }
    }
}

if (alpha == zero)
{
    return;
}
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form y := alpha*A*x + y.
    jx = kx;
    if (incy == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                for (int i = 1; i <= m; i = i + 1)
                {
                    y[i - 1] = y[i - 1] + temp * a[i - 1, j - 1];
                }
            }
            jx = jx + incx;
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                iy = ky;
                for (int i = 1; i <= m; i = i + 1)
                {
                    y[iy - 1] = y[iy - 1] + temp * a[i - 1, j - 1];
                    iy = iy + incy;
                }
            }
            jx = jx + incx;
        }
    }
}
else
{
    // Form y := alpha*A**T*x + y or y := alpha*A**H*x + y.
    jy = ky;
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = zero;
            if (noconj)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    temp = temp + a[i - 1, j - 1] * x[i - 1];
                }
            }
            else
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
                }
            }
        }
    }
}

```

```

        }
        y[jy - 1] = y[jy - 1] + alpha * temp;
        jy = jy + incy;
    }
}
else
{
    for (int j = 1; j <= n; j = j + 1)
    {
        temp = zero;
        ix = kx;
        if (noconj)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = temp + a[i - 1, j - 1] * x[ix - 1];
                ix = ix + incx;
            }
        }
        else
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
                ix = ix + incx;
            }
        }
        y[jy - 1] = y[jy - 1] + alpha * temp;
        jy = jy + incy;
    }
}
}

public static void ZGBMV(string trans, int m, int n, int kl, int ku, Complex alpha, Complex[,] a, int lda, Complex[] x, int incx, Complex beta, ref
Complex[] y, int incy)
{
    // COMPLEX*16 A(LDA,*),X(*),Y(*)

    // ZGBMV performs one of the matrix-vector operations
    // y := alpha*A*x + beta*y, or y := alpha*A**T*x + beta*y, or
    // y := alpha*A**H*x + beta*y,
    // where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' y := alpha*A*x + beta*y.
    // TRANS = 'T' or 't' y := alpha*A**T*x + beta*y.
    // TRANS = 'C' or 'c' y := alpha*A**H*x + beta*y.

    // M is INTEGER
    // On entry, M specifies the number of rows of the matrix A. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of the matrix A. N must be at least zero.

    // KL is INTEGER
    // On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.

    // KU is INTEGER
    // On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry, the leading ( kl + ku + 1 ) by n part of the array A must contain the matrix of coefficients, supplied column by column, with
the leading diagonal of the matrix in
    // row ( ku + 1 ) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row
( ku + 2 ), and so on.
    // Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced.
    // The following program segment will transfer a band matrix from conventional full matrix storage to band storage:
    //
    //      DO 20, J = 1, N
    //          K = KU + 1 - J
    //          DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
    //              A( K + I, J ) = matrix( I, J )
    //          10 CONTINUE
    //      20 CONTINUE

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( kl + ku + 1 ).

    // X is COMPLEX*16 array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( m - 1 ) * abs( INCX ) )
otherwise.
    // Before entry, the incremented array X must contain the vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

    // Y is COMPLEX*16 array of DIMENSION at least ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n' and at least ( 1 + ( n - 1 ) * abs( INCY ) )
otherwise.
    // Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // Parameters
    Complex one = new Complex(1.0, 0.0);
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    //int i, info, ix, iy, j, jx, jy, k, kupl, kx, ky, lenx, leny;
    int info, ix, iy, jx, jy, k, kupl, kx, ky, lenx, leny;

    bool noconj;

    // Test the input parameters
    info = 0;

```

```

if ((!(trans.Substring(0, 1).ToUpper() == "N")) && (!(trans.Substring(0, 1).ToUpper() == "T")) && (!(trans.Substring(0, 1).ToUpper() == "C")))
{
    info = 1;
}
else if (m < 0)
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (kl < 0)
{
    info = 4;
}
else if (ku < 0)
{
    info = 5;
}
else if (lda < (kl + ku + 1))
{
    info = 8;
}
else if (incx == 0)
{
    info = 10;
}
else if (incy == 0)
{
    info = 13;
}
if (info != 0)
{
    XERBLA("ZGBMV ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");

// Set LENX and LENY, the lengths of the vectors x and y, and set up the start points in X and Y
if (trans.Substring(0, 1).ToUpper() == "N")
{
    lenx = n;
    leny = m;
}
else
{
    lenx = m;
    leny = n;
}
if (incx > 0)
{
    kx = 1;
}
else
{
    kx = 1 - (lenx - 1) * incx;
}
if (incy > 0)
{
    ky = 1;
}
else
{
    ky = 1 - (leny - 1) * incy;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the band part of A.

// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= leny; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
        {
            for (int i = 1; i <= leny; i = i + 1)
            {

```

```

        y[iy - 1] = beta * y[iy - 1];
        iy = iy + incy;
    }
}
}
if (alpha == zero)
{
    return;
}

kupl = ku + 1;
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form y := alpha*A*x + y.
    jx = kx;
    if (incy == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                k = kupl - j;
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    y[i - 1] = y[i - 1] + temp * a[k + i - 1, j - 1];
                }
            }
            jx = jx + incx;
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * x[jx - 1];
                iy = ky;
                k = kupl - j;
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    y[iy - 1] = y[iy - 1] + temp * a[k + i - 1, j - 1];
                    iy = iy + incy;
                }
            }
            jx = jx + incx;
            if (j > ku)
            {
                ky = ky + incy;
            }
        }
    }
}
}
else
{
    // Form y := alpha*A**T*x + y or y := alpha*A**H*x + y.
    jy = ky;
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = zero;
            k = kupl - j;
            if (noconj)
            {
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + a[k + i - 1, j - 1] * x[i - 1];
                }
            }
            else
            {
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[k + i - 1, j - 1]) * x[i - 1];
                }
            }
            y[jy - 1] = y[jy - 1] + alpha * temp;
            jy = jy + incy;
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = zero;
            ix = kx;
            k = kupl - j;
            if (noconj)
            {
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + a[k + i - 1, j - 1] * x[ix - 1];
                    ix = ix + incx;
                }
            }
            else
            {
                for (int i = Math.Max(1, j - ku); i <= Math.Min(m, j + kl); i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[k + i - 1, j - 1]) * x[ix - 1];
                    ix = ix + incx;
                }
            }
            y[jy - 1] = y[jy - 1] + alpha * temp;
            jy = jy + incy;
            if (j > ku)
            {
                kx = kx + incx;
            }
        }
    }
}

```

```

    }
}
}
}
public static void ZHEMV(string uplo, int n, Complex alpha, Complex[,] a, int lda, Complex[] x, int incx, Complex beta, ref Complex[] y, int incy)
{
    // COMPLEX*16 A(LDA,*),X(*),Y(*)
    // ZHEMV performs the matrix-vector operation
    // y := alpha*A*x + beta*y,
    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
    //      UPLO = 'U' or 'u'    Only the upper triangular part of A is to be referenced.
    //      UPLO = 'L' or 'l'    Only the lower triangular part of A is to be referenced.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A.  N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.
    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).  Before entry with  UPLO = 'U' or 'u', the leading n by n upper triangular part of the array
    // A must contain the upper
    // triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced.  Before entry with UPLO = 'L' or 'l',
    // the leading n by n
    // lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part
    // of A is not
    // referenced.
    // Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

    // Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // The vector and matrix arguments are not referenced when N = 0, or M = 0

    // Parameters
    Complex one = new Complex(1.0, 0.0);
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp1, temp2;
    //int i, info, ix, iy, j, jx, jy, kx, ky //, lenx, leny;
    int info, ix, iy, jx, jy, kx, ky; //, lenx, leny;
    bool noconj;

    jx = 0; // added
    // Test the input parameters
    info = 0;
    if ((!(uplo.Substring(0, 1).ToUpper() == "U")) && (!(uplo.Substring(0, 1).ToUpper() == "L")))
    {
        info = 1;
    }
    else if (n < 0)
    {
        info = 2;
    }
    else if (lda < Math.Max(1, n))
    {
        info = 5;
    }
    else if (incx == 0)
    {
        info = 7;
    }
    else if (incy == 0)
    {
        info = 10;
    }
    }
    if (info != 0)
    {
        XERBLA("ZHEMV ", info);
        return;
    }

    // Quick return if possible
    if ((n == 0) || ((alpha == zero) && (beta == one)))
    {
        return;
    }

    // Set up the start points in X and Y.
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (n - 1) * incx; // n was leny
    }
    if (incy > 0)
    {
        ky = 1;
    }
    else
    {
        ky = 1 - (n - 1) * incy; // n was leny
    }
    // Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.

```

```

// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[iy - 1] = beta * y[iy - 1];
                iy = iy + incy;
            }
        }
    }
}

if (alpha == zero)
{
    return;
}
if (uplo.Substring(0, 1).ToUpper() == "U")
{
    // Form y when A is stored in upper triangle.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                y[i - 1] = y[i - 1] + temp1 * a[i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
            }
            y[j - 1] = y[j - 1] + temp1 * a[j - 1, j - 1].Real + alpha * temp2;
            jx = jx + incx;
        }
    }
    else
    {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[jx - 1];
            temp2 = zero;
            ix = kx;
            iy = ky;
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                y[iy - 1] = y[iy - 1] + temp1 * a[i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
                ix = ix + incx;
                iy = iy + incy;
            }
            y[jy - 1] = y[jy - 1] + temp1 * a[j - 1, j - 1].Real + alpha * temp2;
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}
else
{
    // Form y when A is stored in lower triangle.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            y[j - 1] = y[j - 1] + temp1 * a[j - 1, j - 1].Real;
            for (int i = j + 1; i <= n; i = i + 1)
            {
                y[i - 1] = y[i - 1] + temp1 * a[i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
            }
            y[j - 1] = y[j - 1] + alpha * temp2;
        }
    }
    else
    {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1)
        {

```

```

        temp1 = alpha * x[jx - 1];
        temp2 = zero;
        y[jy - 1] = y[jy - 1] + temp1 * a[j - 1, j - 1].Real;
        ix = jx;
        iy = jy;
        for (int i = j + 1; i <= n; i = i + 1)
        {
            ix = ix + incx;
            iy = iy + incy;
            y[iy - 1] = y[iy - 1] + temp1 * a[i - 1, j - 1];
            temp2 = temp2 + Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
        }
        y[jy - 1] = y[jy - 1] + alpha * temp2;
        jx = jx + incx;
        jy = jy + incy;
    }
}

}

public static void ZHBMV(string uplo, int n, int k, Complex alpha, Complex[,] a, int lda, Complex[] x, int incx, Complex beta, ref Complex[] y, int
incy)
{
    // COMPLEX*16 A(LDA,*),X(*),Y(*)
    // ZHBMV performs the matrix-vector operation
    // y := alpha*A*x + beta*y,
    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian band matrix, with k super-diagonals.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:
    //      UPLO = 'U' or 'u' The upper triangular part of A is being supplied.
    //      UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // K is INTEGER
    // On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
hermitian matrix, supplied column by
    // column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
so on. The top left k by k triangle
    // of the array A is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from
conventional
    // full matrix storage to band storage:
    //
    //      DO 20, J = 1, N
    //          M = K + 1 - J
    //          DO 10, I = MAX( 1, J - K ), J
    //              A( M + I, J ) = matrix( I, J )
    //          10 CONTINUE
    //      20 CONTINUE

    // Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
hermitian matrix, supplied column by
    // column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
The bottom right k by k triangle of the
    // array A is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from
conventional full matrix storage to band storage:
    //
    //      DO 20, J = 1, N
    //          M = 1 - J
    //          DO 10, I = J, MIN( N, J + K )
    //              A( M + I, J ) = matrix( I, J )
    //          10 CONTINUE
    //      20 CONTINUE

    // Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).

    // X is COMPLEX*16 array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta.

    // Y is COMPLEX*16 array of DIMENSION at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // Parameters
    Complex one = new Complex(1.0, 0.0);
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp1, temp2;
    //int i, info, ix, iy, j, jx, jy, kplus1, kx, ky, l //, lenx, leny;
    int info, ix, iy, jx, jy, kplus1, kx, ky, l; //, lenx, leny;
    bool noconj;

    // Test the input parameters
    info = 0;
    if ( !(uplo.Substring(0, 1).ToUpper() == "U") ) && !(uplo.Substring(0, 1).ToUpper() == "L") )
    {
        info = 1;
    }
    else if ( n < 0 )
    {
        info = 2;
    }

```

```

    }
    else if (k < 0)
    {
        info = 3;
    }
    else if (lda < (k + 1))
    {
        info = 6;
    }
    else if (incx == 0)
    {
        info = 8;
    }
    else if (incy == 0)
    {
        info = 11;
    }

    if (info != 0)
    {
        XERBLA("ZHBMV ", info);
        return;
    }

    // Quick return if possible
    if ((n == 0) || ((alpha == zero) && (beta == one)))
    {
        return;
    }

    // Set up the start points in X and Y.
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (n - 1) * incx; // n was leny
    }
    if (incy > 0)
    {
        ky = 1;
    }
    else
    {
        ky = 1 - (n - 1) * incy; // n was leny
    }

    // Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.
    // First form y := beta*y.
    if (beta != one)
    {
        if (incy == 1)
        {
            if (beta == zero)
            {
                for (int i = 1; i <= n; i = i + 1)
                {
                    y[i - 1] = zero;
                }
            }
            else
            {
                for (int i = 1; i <= n; i = i + 1)
                {
                    y[i - 1] = beta * y[i - 1];
                }
            }
        }
        else
        {
            iy = ky;
            if (beta == zero)
            {
                for (int i = 1; i <= n; i = i + 1)
                {
                    y[iy - 1] = zero;
                    iy = iy + incy;
                }
            }
            else
            {
                for (int i = 1; i <= n; i = i + 1)
                {
                    y[iy - 1] = beta * y[iy - 1];
                    iy = iy + incy;
                }
            }
        }
    }
}

if (alpha == zero)
{
    return;
}
if (uplo.Substring(0, 1).ToUpper() == "U")
{
    // Form y when upper triangle of A is stored.
    kplus1 = k + 1;
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            l = kplus1 - j;
            for (int i = Math.Max(1, j - k); i <= j - 1; i = i + 1) //DO 50 I = MAX(1,J-K),J - 1
            {
                y[i - 1] = y[i - 1] + temp1 * a[l + i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[l + i - 1, j - 1]) * x[i - 1];
            }
            y[j - 1] = y[j - 1] + temp1 * a[kplus1 - 1, j - 1].Real + alpha * temp2;
        }
    }
}

```

```

    }
    else
    {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[jx - 1];
            temp2 = zero;
            ix = kx;
            iy = ky;
            l = kplus1 - j;
            for (int i = Math.Max(1, j - k); i <= j - 1; i = i + 1) //DO 70 I = MAX(1,J-K),J - 1
            {
                y[iy - 1] = y[iy - 1] + temp1 * a[l + i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[l + i - 1, j - 1]) * x[ix - 1];
                ix = ix + incx;
                iy = iy + incy;
            }
            y[jy - 1] = y[jy - 1] + temp1 * a[kplus1 - 1, j - 1].Real + alpha * temp2;
            jx = jx + incx;
            jy = jy + incy;
            if (j > k)
            {
                kx = kx + incx;
                ky = ky + incy;
            }
        }
    }
}

else
{
    // Form y when lower triangle of A is stored.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            y[j - 1] = y[j - 1] + temp1 * a[0, j - 1].Real;
            l = 1 - j;
            for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
            {
                y[i - 1] = y[i - 1] + temp1 * a[l + i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[l + i - 1, j - 1]) * x[i - 1];
            }
            y[j - 1] = y[j - 1] + alpha * temp2;
        }
    }
    else
    {
        jx = kx;
        jy = ky;

        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[jx - 1];
            temp2 = zero;
            y[jy - 1] = y[jy - 1] + temp1 * a[0, j - 1].Real;
            l = 1 - j;
            ix = jx;
            iy = jy;
            for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
            {
                ix = ix + incx;
                iy = iy + incy;
                y[iy - 1] = y[iy - 1] + temp1 * a[l + i - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(a[l + i - 1, j - 1]) * x[ix - 1];
            }
            y[jy - 1] = y[jy - 1] + alpha * temp2;
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}

}

public static void ZHPMV(string uplo, int n, Complex alpha, Complex[] ap, Complex[] x, int incx, Complex beta, ref Complex[] y, int incy)
{
    // COMPLEX*16 AP(*),X(*),Y(*)

    // ZHPMV performs the matrix-vector operation
    // y := alpha*A*x + beta*y,
    // where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:
    //      UPLO = 'U' or 'u'   The upper triangular part of A is supplied in AP.
    //      UPLO = 'L' or 'l'   The lower triangular part of A is supplied in AP.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // AP is COMPLEX*16 array of DIMENSION at least ( ( n*( n + 1 ) )/2 ).
    // Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column
    // by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. Before entry with UPLO = 'L' or 'l', the
    // array AP must
    // contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2 )
    // and AP( 3 ) contain a( 2, 1 )
    // and a( 3, 1 ) respectively, and so on. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ). Before entry, the incremented array X must contain the n element
    // vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

```

```

// BETA is COMPLEX*16
// On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

// Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
// Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

// INCY is INTEGER
// On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

// Parameters
Complex one = new Complex(1.0, 0.0);
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp1, temp2;
//int i, info, ix, iy, j, jx, jy, k, kk, kx, ky
int info, ix, iy, jx, jy, k, kk, kx, ky;

// Test the input parameters
info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U")) && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (n < 0)
{
    info = 2;
}
else if (incx == 0)
{
    info = 6;
}
else if (incy == 0)
{
    info = 9;
}
if (info != 0)
{
    XERBLA("ZHPMV ", info);
    return;
}

// Quick return if possible
if ((n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

// Set up the start points in X and Y.
if (incx > 0)
{
    kx = 1;
}
else
{
    kx = 1 - (n - 1) * incx; // n was leny
}
if (incy > 0)
{
    ky = 1;
}
else
{
    ky = 1 - (n - 1) * incy; // n was leny
}

// Start the operations. In this version the elements of AP are accessed sequentially with one pass through AP.
// First form y := beta*y.
if (beta != one)
{
    if (incy == 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[i - 1] = zero;
            }
        }
        else
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[i - 1] = beta * y[i - 1];
            }
        }
    }
    else
    {
        iy = ky;
        if (beta == zero)
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[iy - 1] = zero;
                iy = iy + incy;
            }
        }
        else
        {
            for (int i = 1; i <= n; i = i + 1)
            {
                y[iy - 1] = beta * y[iy - 1];
                iy = iy + incy;
            }
        }
    }
}
}

```

```

if (alpha == zero)
{
    return;
}
kk = 1;
if (uplo.Substring(0, 1).ToUpper() == "U")
{
    // Form y when AP contains the upper triangle.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            k = kk;
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                y[i - 1] = y[i - 1] + temp1 * ap[k - 1];
                temp2 = temp2 + Complex.Conjugate(ap[k - 1]) * x[i - 1];
                k = k + 1;
            }
            y[j - 1] = y[j - 1] + temp1 * ap[(kk + j - 1) - 1].Real + alpha * temp2;
            kk = kk + j;
        }
    }
    else
    {
        jx = kx;
        jy = ky;
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[jx - 1];
            temp2 = zero;
            ix = kx;
            iy = ky;
            for (k = kk; k <= kk + j - 2; k = k + 1)
            {
                y[iy - 1] = y[iy - 1] + temp1 * ap[k - 1];
                temp2 = temp2 + Complex.Conjugate(ap[k - 1]) * x[ix - 1];
                ix = ix + incx;
                iy = iy + incy;
            }
            y[jy - 1] = y[jy - 1] + temp1 * ap[(kk + j - 1) - 1].Real + alpha * temp2;
            jx = jx + incx;
            jy = jy + incy;
            kk = kk + j;
        }
    }
}
else
{
    // Form y when AP contains the lower triangle.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[j - 1];
            temp2 = zero;
            y[j - 1] = y[j - 1] + temp1 * ap[kk - 1].Real;
            k = kk + 1;
            for (int i = j + 1; i <= n; i = i + 1)
            {
                y[i - 1] = y[i - 1] + temp1 * ap[k - 1];
                temp2 = temp2 + Complex.Conjugate(ap[k - 1]) * x[i - 1];
                k = k + 1;
            }
            y[j - 1] = y[j - 1] + alpha * temp2;
            kk = kk + (n - j + 1);
        }
    }
    else
    {
        jx = kx;
        jy = ky;

        for (int j = 1; j <= n; j = j + 1)
        {
            temp1 = alpha * x[jx - 1];
            temp2 = zero;
            y[jy - 1] = y[jy - 1] + temp1 * ap[kk - 1].Real;
            ix = jx;
            iy = jy;
            for (k = kk + 1; k <= kk + n - j; k = k + 1)
            {
                ix = ix + incx;
                iy = iy + incy;
                y[iy - 1] = y[iy - 1] + temp1 * ap[k - 1];
                temp2 = temp2 + Complex.Conjugate(ap[k - 1]) * x[ix - 1];
            }
            y[jy - 1] = y[jy - 1] + alpha * temp2;
            jx = jx + incx;
            jy = jy + incy;
            kk = kk + (n - j + 1);
        }
    }
}
}
public static void ZTRMV(string uplo, string trans, string diag, int n, Complex[,] a, int lda, ref Complex[] x, int incx)
{
    // COMPLEX*16 A(LDA,*),X(*)
    // ZTRMV performs one of the matrix-vector operations
    // x := A*x, or x := A**T*x, or x := A**H*x,
    // where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANS is CHARACTER*1

```

```

// On entry, TRANS specifies the operation to be performed as follows:
// TRANS = 'N' or 'n'   x := A*x.
// TRANS = 'T' or 't'   x := A**T*x.
// TRANS = 'C' or 'c'   x := A**H*x.

// DIAG is CHARACTER*1
// On entry, DIAG specifies whether or not A is unit triangular as follows:
// DIAG = 'U' or 'u'   A is assumed to be unit triangular.
// DIAG = 'N' or 'n'   A is not assumed to be unit triangular.

// N is INTEGER
// On entry, N specifies the order of the matrix A.  N must be at least zero.

// A is COMPLEX*16 array of DIMENSION ( LDA, n ).
// Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and
the strictly lower triangular part of
// A is not referenced.  Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower
triangular matrix and the strictly upper triangular part of
// A is not referenced.  Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

// X is (input/output) COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the tranformed vector x.

// INCX is INTEGER
// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

// Parameters
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp;
//int i, info, ix, j, jx, kx;
int info, ix, jx, kx;
bool noconj, nunit;

kx = 0; //added
// Test the input parameters
info = 0;
if ( (! (uplo.Substring(0, 1).ToUpper() == "U")) && (! (uplo.Substring(0, 1).ToUpper() == "L")) ) )
{
    info = 1;
}
else if ( (! (trans.Substring(0, 1).ToUpper() == "N")) && ! (trans.Substring(0, 1).ToUpper() == "T") && ! (trans.Substring(0, 1).ToUpper() == "C")) )
{
    info = 2;
}
else if ( (! (diag.Substring(0, 1).ToUpper() == "U")) && ! (diag.Substring(0, 1).ToUpper() == "N")) )
{
    info = 3;
}
else if ( n < 0 )
{
    info = 4;
}
else if ( lda < Math.Max(1, n) )
{
    info = 6;
}
else if ( incx == 0 )
{
    info = 8;
}

if ( info != 0 )
{
    XERBLA("ZTRMV ", info);
    return;
}

// Quick return if possible
if ( n == 0 )
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");
nunit = (diag.Substring(0, 1).ToUpper() == "N");

// Set up the start point in X if the increment is not unity. This will be ( N - 1 ) * INCX too small for descending loops.
if ( incx <= 0 )
{
    kx = 1 - ( n - 1 ) * incx;
}
else if ( incx != 1 )
{
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if ( trans.Substring(0, 1).ToUpper() == "N" )
{
    // Form x := A*x.
    if ( uplo.Substring(0, 1).ToUpper() == "U" )
    {
        if ( incx == 1 )
        {
            for ( int j = 1; j <= n; j = j + 1 )
            {
                if ( x[j - 1] != zero )
                {
                    temp = x[j - 1];
                    for ( int i = 1; i <= j - 1; i = i + 1 )
                    {
                        x[i - 1] = x[i - 1] + temp * a[i - 1, j - 1];
                    }
                    if ( nunit )
                    {
                        x[j - 1] = x[j - 1] * a[j - 1, j - 1];
                    }
                }
            }
        }
    }
}

```

```

    }
}
else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        if (x[jx - 1] != zero)
        {
            temp = x[jx - 1];
            ix = kx;
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                x[ix - 1] = x[ix - 1] + temp * a[i - 1, j - 1];
                ix = ix + incx;
            }
            if (nounit)
            {
                x[jx - 1] = x[jx - 1] * a[j - 1, j - 1];
            }
            jx = jx + incx;
        }
    }
}
else
{
    if (incx == 1)
    {
        for (int j = n; j >= 1; j = j - 1)
        {
            if (x[j - 1] != zero)
            {
                temp = x[j - 1];
                for (int i = n; i >= j + 1; i = i - 1)
                {
                    x[i - 1] = x[i - 1] + temp * a[i - 1, j - 1];
                }
                if (nounit)
                {
                    x[j - 1] = x[j - 1] * a[j - 1, j - 1];
                }
            }
        }
    }
    else
    {
        kx = kx + (n - 1) * incx;
        jx = kx;
        for (int j = n; j >= 1; j = j - 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = x[jx - 1];
                ix = kx;
                for (int i = n; i >= j + 1; i = i - 1)
                {
                    x[ix - 1] = x[ix - 1] + temp * a[i - 1, j - 1];
                    ix = ix - incx;
                }
                if (nounit)
                {
                    x[jx - 1] = x[jx - 1] * a[j - 1, j - 1];
                }
            }
            jx = jx - incx;
        }
    }
}
}
else
{
    // Form x := A**T*x or x := A**H*x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                temp = x[j - 1];
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * a[j - 1, j - 1];
                    }
                    for (int i = j - 1; i >= 1; i = i - 1)
                    {
                        temp = temp + a[i - 1, j - 1] * x[i - 1];
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(a[j - 1, j - 1]);
                    }
                    for (int i = j - 1; i >= 1; i = i - 1)
                    {
                        temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
                    }
                }
                x[j - 1] = temp;
            }
        }
        else
        {
            jx = kx + (n - 1) * incx;
            for (int j = n; j >= 1; j = j - 1)
            {
                temp = x[jx - 1];
                ix = jx;
                if (noconj)
                {

```

```

        if (nounit)
        {
            temp = temp * a[j - 1, j - 1];
        }
        for (int i = j - 1; i >= 1; i = i - 1)
        {
            ix = ix - incx;
            temp = temp + a[i - 1, j - 1] * x[ix - 1];
        }
    }
    else
    {
        if (nounit)
        {
            temp = temp * Complex.Conjugate(a[j - 1, j - 1]);
        }
        for (int i = j - 1; i >= 1; i = i - 1)
        {
            ix = ix - incx;
            temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
        }
        x[jx - 1] = temp;
        jx = jx - incx;
    }
}
}
else
{
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = x[j - 1];
            if (noconj)
            {
                if (nounit)
                {
                    temp = temp * a[j - 1, j - 1];
                }
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    temp = temp + a[i - 1, j - 1] * x[i - 1];
                }
            }
            else
            {
                if (nounit)
                {
                    temp = temp * Complex.Conjugate(a[j - 1, j - 1]);
                }
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
                }
            }
            x[j - 1] = temp;
        }
    }
    else
    {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = x[jx - 1];
            ix = jx;
            if (noconj)
            {
                if (nounit)
                {
                    temp = temp * a[j - 1, j - 1];
                }
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    ix = ix + incx;
                    temp = temp + a[i - 1, j - 1] * x[ix - 1];
                }
            }
            else
            {
                if (nounit)
                {
                    temp = temp * Complex.Conjugate(a[j - 1, j - 1]);
                }
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    ix = ix + incx;
                    temp = temp + Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
                }
            }
            x[jx - 1] = temp;
            jx = jx + incx;
        }
    }
}
}

public static void ZTBMV(string uplo, string trans, string diag, int n, int k, Complex[,] a, int lda, ref Complex[] x, int incx)
{
    // COMPLEX*16 A(LDA,*),X(*)
    // ZTBMV performs one of the matrix-vector operations
    // x := A*x, or x := A**T*x, or x := A**H*x,
    // where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with ( k + 1 ) diagonals.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' x := A*x.
    // TRANS = 'T' or 't' x := A**T*x.
    // TRANS = 'C' or 'c' x := A**H*x.

```

```

// DIAG is CHARACTER*1
// On entry, DIAG specifies whether or not A is unit triangular as follows:
//      DIAG = 'U' or 'u'   A is assumed to be unit triangular.
//      DIAG = 'N' or 'n'   A is not assumed to be unit triangular.

// N is INTEGER
// On entry, N specifies the order of the matrix A.  N must be at least zero.

// K is INTEGER
// On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.  On entry with UPLO = 'L' or 'l', K specifies
the number of
// sub-diagonals of the matrix A.  K must satisfy 0 .le. K.

// A is COMPLEX*16 array of DIMENSION ( LDA, n ).
// Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
matrix of coefficients, supplied column by
// column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
so on. The top left k by k triangle
// of the array A is not referenced.  The following program segment will transfer an upper triangular band matrix from conventional full matrix
storage to band storage:
//
//      DO 20, J = 1, N
//          M = K + 1 - J
//          DO 10, I = MAX( 1, J - K ), J
//              A( M + I, J ) = matrix( I, J )
//          10 CONTINUE
//      20 CONTINUE

// Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
matrix of coefficients, supplied column by
// column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
The bottom right k by k triangle of the
// array A is not referenced.  The following program segment will transfer a lower triangular band matrix from conventional full matrix storage
to band storage:
//
//      DO 20, J = 1, N
//          M = 1 - J
//          DO 10, I = J, MIN( N, J + K )
//              A( M + I, J ) = matrix( I, J )
//          10 CONTINUE
//      20 CONTINUE

// Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but
are assumed
to be unity.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).

// X is (input/output) COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

// INCX is INTEGER
// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

// The vector and matrix arguments are not referenced when N = 0, or M = 0
// Parameters
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp;
int info, ix, jx, kplus1, kx, l;
bool noconj, nunit;

kx = 0; // added
// Test the input parameters
info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T") && !(trans.Substring(0, 1).ToUpper() == "C"))
{
    info = 2;
}
else if (!(diag.Substring(0, 1).ToUpper() == "U") && !(diag.Substring(0, 1).ToUpper() == "N"))
{
    info = 3;
}
else if (n < 0)
{
    info = 4;
}
else if (k < 0)
{
    info = 5;
}
else if (lda < (k + 1))
{
    info = 7;
}
else if (incx == 0)
{
    info = 9;
}
if (info != 0)
{
    XERBLA("ZTBMV ", info);
    return;
}

// Quick return if possible
if (n == 0)
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");
nunit = (diag.ToUpper().Substring(0, 1) == "N");

// Set up the start point in X if the increment is not unity. This will be ( N - 1 ) * INCX too small for descending loops.
if (incx <= 0)
{
    kx = 1 - (n - 1) * incx;

```

```

}
else if (incx != 1)
{
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form x := A*x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kplus1 = k + 1;
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[j - 1] != zero)
                {
                    temp = x[j - 1];
                    l = kplus1 - j;
                    for (int i = Math.Max(1, j - k); i <= j - 1; i = i + 1)
                    {
                        x[i - 1] = x[i - 1] + temp * a[l + i - 1, j - 1];
                    }
                    if (nounit)
                    {
                        x[j - 1] = x[j - 1] * a[kplus1 - 1, j - 1];
                    }
                }
            }
        }
        else
        {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[jx - 1] != zero)
                {
                    temp = x[jx - 1];
                    ix = kx;
                    l = kplus1 - j;
                    for (int i = Math.Max(1, j - k); i <= j - 1; i = i + 1)
                    {
                        x[ix - 1] = x[ix - 1] + temp * a[l + i - 1, j - 1];
                        ix = ix + incx;
                    }
                    if (nounit)
                    {
                        x[jx - 1] = x[jx - 1] * a[kplus1 - 1, j - 1];
                    }
                    jx = jx + incx;
                    if (j > k)
                    {
                        kx = kx + incx;
                    }
                }
            }
        }
    }
    else
    {
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 60 J = N,1,-1
                if (x[j - 1] != zero)
                {
                    temp = x[j - 1];
                    l = 1 - j;
                    for (int i = Math.Min(n, j + k); i >= j + 1; i = i - 1)
                    {
                        //DO 50 I = MIN(N,J+K),J + 1,-1
                        x[i - 1] = x[i - 1] + temp * a[l + i - 1, j - 1];
                    }
                    if (nounit)
                    {
                        x[j - 1] = x[j - 1] * a[0, j - 1];
                    }
                }
            }
        }
        else
        {
            kx = kx + (n - 1) * incx;
            jx = kx;
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 80 J = N,1,-1
                if (x[jx - 1] != zero)
                {
                    temp = x[jx - 1];
                    ix = kx;
                    l = 1 - j;
                    for (int i = Math.Min(n, j + k); i >= j + 1; i = i - 1)
                    {
                        //DO 70 I = MIN(N,J+K),J + 1,-1
                        x[ix - 1] = x[ix - 1] + temp * a[l + i - 1, j - 1];
                        ix = ix - incx;
                    }
                    if (nounit)
                    {
                        x[jx - 1] = x[jx - 1] * a[0, j - 1];
                    }
                    jx = jx - incx;
                    if ((n - j) >= k)
                    {
                        kx = kx - incx;
                    }
                }
            }
        }
    }
}
else

```

```

{
    // Form x := A**T*x or x := A**H*x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kplus1 = k + 1;
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 100 J = N,1,-1
                temp = x[j - 1];
                l = kplus1 - j;
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * a[kplus1 - 1, j - 1];
                    }
                    for (int i = j - 1; i >= Math.Max(1, j - k); i = i - 1)
                    {
                        temp = temp + a[l + i - 1, j - 1] * x[i - 1];
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(a[kplus1 - 1, j - 1]);
                    }
                    for (int i = j - 1; i >= Math.Max(1, j - k); i = i - 1)
                    {
                        //DO 90 I = J - 1, MAX(1, J-K), -1
                        temp = temp + Complex.Conjugate(a[l + i - 1, j - 1]) * x[i - 1];
                    }
                }
                x[j - 1] = temp;
            }
        }
        else
        {
            kx = kx + (n - 1) * incx;
            jx = kx;
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 120 J = N,1,-1
                temp = x[jx - 1];
                kx = kx - incx;
                ix = kx;
                l = kplus1 - j;
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * a[kplus1 - 1, j - 1];
                    }
                    for (int i = j - 1; i >= Math.Max(1, j - k); i = i - 1)
                    {
                        temp = temp + a[l + i - 1, j - 1] * x[ix - 1];
                        ix = ix - incx;
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(a[kplus1 - 1, j - 1]);
                    }
                    for (int i = j - 1; i >= Math.Max(1, j - k); i = i - 1)
                    {
                        //DO 110 I = J - 1, MAX(1, J-K), -1
                        temp = temp + Complex.Conjugate(a[l + i - 1, j - 1]) * x[ix - 1];
                        ix = ix - incx;
                    }
                }
                x[jx - 1] = temp;
                jx = jx - incx;
            }
        }
    }
}

else
{
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = x[j - 1];
            l = 1 - j;
            if (noconj)
            {
                if (nounit)
                {
                    temp = temp * a[0, j - 1];
                }
                for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
                {
                    temp = temp + a[l + i - 1, j - 1] * x[i - 1];
                }
            }
            else
            {
                if (nounit)
                {
                    temp = temp * Complex.Conjugate(a[0, j - 1]);
                }
                for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[l + i - 1, j - 1]) * x[i - 1];
                }
            }
            x[j - 1] = temp;
        }
    }
    else
    {
        jx = kx;
    }
}

```

```

        for (int j = 1; j <= n; j = j + 1)
        {
            temp = x[jx - 1];
            kx = kx + incx;
            ix = kx;
            l = l - j;
            if (noconj)
            {
                if (nunit)
                {
                    temp = temp * a[0, j - 1];
                }
                for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
                {
                    temp = temp + a[l + i - 1, j - 1] * x[ix - 1];
                    ix = ix + incx;
                }
            }
            else
            {
                if (nunit)
                {
                    temp = temp * Complex.Conjugate(a[0, j - 1]);
                }
                for (int i = j + 1; i <= Math.Min(n, j + k); i = i + 1)
                {
                    temp = temp + Complex.Conjugate(a[l + i - 1, j - 1]) * x[ix - 1];
                    ix = ix + incx;
                }
            }
            x[jx - 1] = temp;
            jx = jx + incx;
        }
    }
}

public static void ZTPMV(string uplo, string trans, string diag, int n, Complex[] ap, ref Complex[] x, int incx)
{
    // COMPLEX*16 AP(*),X(*)
    // ZTPMV performs one of the matrix-vector operations
    // x := A*x, or x := A**T*x, or x := A**H*x,
    // where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' x := A*x.
    // TRANS = 'T' or 't' x := A**T*x.
    // TRANS = 'C' or 'c' x := A**H*x.

    // DIAG is CHARACTER*1
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    // DIAG = 'U' or 'u' A is assumed to be unit triangular.
    // DIAG = 'N' or 'n' A is not assumed to be unit triangular.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // AP is COMPLEX*16 array of DIMENSION at least ( ( n*( n + 1 ) )/2 ).
    // Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that
    AP( 1 ) contains a( 1, 1 ),
    // AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must
    // contain the lower triangular matrix packed sequentially, column by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain
    a( 2, 1 ) and a( 3, 1 )
    // respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

    // X is (input/output) COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) *abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the tranformed vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // The vector and matrix arguments are not referenced when N = 0, or M = 0

    // Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jx, k, kk, kx;
    bool noconj, nunit;

    kx = 0; //added
    // Test the input parameters
    info = 0;
    if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
    {
        info = 1;
    }
    else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T") && !(trans.Substring(0, 1).ToUpper() == "C"))
    {
        info = 2;
    }
    else if (!(diag.ToUpper().Substring(0, 1) == "U") && !(diag.ToUpper().Substring(0, 1) == "N"))
    {
        info = 3;
    }
    else if (n < 0)
    {
        info = 4;
    }
    else if (incx == 0)
    {
        info = 7;
    }
    if (info != 0)
    {
        XERBLA("ZTPMV ", info);
    }
}

```

```

        return;
    }

    // Quick return if possible
    if (n == 0)
    {
        return;
    }

    noconj = (trans.Substring(0, 1).ToUpper() == "T");
    nunit = (diag.Substring(0, 1).ToUpper() == "N");

    // Set up the start point in X if the increment is not unity. This will be (N - 1)*INCX too small for descending loops.
    if (incx <= 0)
    {
        kx = 1 - (n - 1) * incx;
    }
    else if (incx != 1)
    {
        kx = 1;
    }

    // Start the operations. In this version the elements of AP are accessed sequentially with one pass through AP.
    if (trans.Substring(0, 1).ToUpper() == "N")
    {
        // Form x:=A*x.
        if (uplo.Substring(0, 1).ToUpper() == "U")
        {
            kk = 1;
            if (incx == 1)
            {
                for (int j = 1; j <= n; j = j + 1)
                {
                    if (x[j - 1] != zero)
                    {
                        temp = x[j - 1];
                        k = kk;
                        for (int i = 1; i <= (j - 1); i = i + 1)
                        {
                            x[i - 1] = x[i - 1] + temp * ap[k - 1];
                            k = k + 1;
                        }
                        if (nunit)
                        {
                            x[j - 1] = x[j - 1] * ap[(kk + j - 1) - 1];
                        }
                    }
                    kk = kk + j;
                }
            }
            else
            {
                jx = kx;
                for (int j = 1; j <= n; j = j + 1)
                {
                    if (x[jx - 1] != zero)
                    {
                        temp = x[jx - 1];
                        ix = kx;
                        for (k = kk; k <= (kk + j - 2); k = k + 1)
                        {
                            x[ix - 1] = x[ix - 1] + temp * ap[k - 1];
                            ix = ix + incx;
                        }
                        if (nunit)
                        {
                            x[jx - 1] = x[jx - 1] * ap[(kk + j - 1) - 1];
                        }
                    }
                    jx = jx + incx;
                    kk = kk + j;
                }
            }
        }
        else
        {
            kk = (n * (n + 1)) / 2;
            if (incx == 1)
            {
                for (int j = n; j >= 1; j = j - 1)
                {
                    //DO 60 J = N,1,-1
                    if (x[j - 1] != zero)
                    {
                        temp = x[j - 1];
                        k = kk;
                        for (int i = n; i >= (j + 1); i = i - 1)
                        {
                            //DO 50 I = N,J + 1,-1
                            x[i - 1] = x[i - 1] + temp * ap[k - 1];
                            k = k - 1;
                        }
                        if (nunit)
                        {
                            x[j - 1] = x[j - 1] * ap[(kk - n + j) - 1];
                        }
                    }
                    kk = kk - (n - j + 1);
                }
            }
            else
            {
                kx = kx + (n - 1) * incx;
                jx = kx;
                for (int j = n; j >= 1; j = j - 1)
                {
                    if (x[jx - 1] != zero)
                    {
                        temp = x[jx - 1];
                        ix = kx;
                        for (k = kk; k >= (kk - (n - (j + 1)))); k = k - 1)
                        {
                            //DO 70 K = KK, KK - (N - (J + 1)), -1
                            x[ix - 1] = x[ix - 1] + temp * ap[k - 1];
                            ix = ix - incx;
                        }
                    }
                }
            }
        }
    }

```

```

        }
        if (nounit)
        {
            x[jx - 1] = x[jx - 1] * ap[(kk - n + j) - 1];
        }
    }
    jx = jx - incx;
    kk = kk - (n - j + 1);
}
}
}

else
{
    // Form x := A**T*x or x := A**H*x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kk = (n * (n + 1)) / 2;
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 100 J = N,1,-1
                temp = x[j - 1];
                k = kk - 1;
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * ap[kk - 1];
                    }
                    //k = kk - 1;
                    for (int i = (j - 1); i >= 1; i = i - 1)
                    {
                        //DO 90 I = J - 1,1,-1
                        temp = temp + ap[k - 1] * x[i - 1];
                        k = k - 1;
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(ap[kk - 1]);
                    }
                    //k = kk - 1;
                    for (int i = (j - 1); i >= 1; i = i - 1)
                    {
                        //DO 90 I = J - 1,1,-1
                        temp = temp + Complex.Conjugate(ap[k - 1]) * x[i - 1];
                        k = k - 1;
                    }
                }
                x[j - 1] = temp;
                kk = kk - j;
            }
        }
        else
        {
            jx = kx + (n - 1) * incx;
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 120 J = N,1,-1
                temp = x[jx - 1];
                ix = jx;
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * ap[kk - 1];
                    }
                    for (k = (kk - 1); k >= (kk - j + 1); k = k - 1)
                    {
                        //DO 110 K = KK - 1, KK - J + 1, -1
                        ix = ix - incx;
                        temp = temp + ap[k - 1] * x[ix - 1];
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(ap[kk - 1]);
                    }
                    for (k = (kk - 1); k >= (kk - j + 1); k = k - 1)
                    {
                        //DO 110 K = KK - 1, KK - J + 1, -1
                        ix = ix - incx;
                        temp = temp + Complex.Conjugate(ap[k - 1]) * x[ix - 1];
                    }
                }
                x[jx - 1] = temp;
                jx = jx - incx;
                kk = kk - j;
            }
        }
    }
}

else
{
    kk = 1;
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            temp = x[j - 1];
            k = kk + 1;
            if (noconj)
            {
                if (nounit)
                {
                    temp = temp * ap[kk - 1];
                }

                for (int i = (j + 1); i <= n; i = i + 1)
                {
                    temp = temp + ap[k - 1] * x[i - 1];
                    k = k + 1;
                }
            }
            else
            {
                if (nounit)
                {
                    temp = temp * Complex.Conjugate(ap[kk - 1]);
                }

                for (int i = (j + 1); i <= n; i = i + 1)
                {
                    temp = temp + Complex.Conjugate(ap[k - 1]) * x[i - 1];
                    k = k + 1;
                }
            }
            x[j] = temp;
            kk = kk + j;
        }
    }
}

```

```

        }
    }
    else
    {
        if (nunit)
        {
            temp = temp * Complex.Conjugate(ap[kk - 1]);
        }
        for (int i = (j + 1); i <= n; i = i + 1)
        {
            temp = temp + Complex.Conjugate(ap[k - 1]) * x[i - 1];
            k = k + 1;
        }
        x[j - 1] = temp;
        kk = kk + (n - j + 1);
    }
}
else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        temp = x[jx - 1];
        ix = jx;
        if (noconj)
        {
            if (nunit)
            {
                temp = temp * ap[kk - 1];
            }
            for (k = (kk + 1); k <= (kk + n - j); k = k + 1)
            {
                ix = ix + incx;
                temp = temp + ap[k - 1] * x[ix - 1];
            }
        }
        else
        {
            if (nunit)
            {
                temp = temp * Complex.Conjugate(ap[kk - 1]);
            }
            for (k = (kk + 1); k <= (kk + n - j); k = k + 1)
            {
                ix = ix + incx;
                temp = temp + Complex.Conjugate(ap[k - 1]) * x[ix - 1];
            }
        }
        x[jx - 1] = temp;
        jx = jx + incx;
        kk = kk + (n - j + 1);
    }
}
}

}

public static void ZTRSV(string uplo, string trans, string diag, int n, Complex[,] a, int lda, ref Complex[] x, int incx)
{
    //COMPLEX*16 A(LDA,*),X(*)
    // ZTRSV solves one of the systems of equations
    // A*x = b, or A**T*x = b, or A**H*x = b,
    // where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.
    // No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    //      UPLO = 'U' or 'u' A is an upper triangular matrix.
    //      UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the equations to be solved as follows:
    //      TRANS = 'N' or 'n' A*x = b.
    //      TRANS = 'T' or 't' A**T*x = b.
    //      TRANS = 'C' or 'c' A**H*x = b.

    // DIAG is CHARACTER*1
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    //      DIAG = 'U' or 'u' A is assumed to be unit triangular.
    //      DIAG = 'N' or 'n' A is not assumed to be unit triangular.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and
    the strictly lower triangular part of
    // A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower
    // triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements
    of
    // A are not referenced either, but are assumed to be unity.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution
    vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    //Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jx, kx;
    bool noconj, nunit;

    kx = 0; //added
    // Test the input parameters

```

```

info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T") && !(trans.Substring(0, 1).ToUpper() == "C"))
{
    info = 2;
}
else if (!(diag.Substring(0, 1).ToUpper() == "U") && !(diag.Substring(0, 1).ToUpper() == "N"))
{
    info = 3;
}
else if (n < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, n))
{
    info = 6;
}
else if (incx == 0)
{
    info = 8;
}
if (info != 0)
{
    XERBLA("ZTRSV ", info);
    return;
}

// Quick return if possible.
if (n == 0)
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");
nounit = (diag.Substring(0, 1).ToUpper() == "N");

// Set up the start point in X if the increment is not unity. This will be (N - 1)*INCX too small for descending loops.
if (incx <= 0)
{
    kx = 1 - (n - 1) * incx;
}
else if (incx != 1)
{
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form x := inv(A)*x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            { //DO 20 J = N,1,-1
                if (x[j - 1] != zero)
                {
                    if (nounit)
                    {
                        x[j - 1] = x[j - 1] / a[j - 1, j - 1];
                    }
                    temp = x[j - 1];
                    for (int i = j - 1; i >= 1; i = i - 1)
                    { //DO 10 I = J - 1,1,-1
                        x[i - 1] = x[i - 1] - temp * a[i - 1, j - 1];
                    }
                }
            }
        }
        else
        {
            jx = kx + (n - 1) * incx;
            for (int j = n; j >= 1; j = j - 1)
            { //DO 40 J = N,1,-1
                if (x[jx - 1] != zero)
                {
                    if (nounit)
                    {
                        x[jx - 1] = x[jx - 1] / a[j - 1, j - 1];
                    }
                    temp = x[jx - 1];
                    ix = jx;
                    for (int i = j - 1; i >= 1; i = i - 1)
                    { //DO 30 I = J - 1,1,-1
                        ix = ix - incx;
                        x[ix - 1] = x[ix - 1] - temp * a[i - 1, j - 1];
                    }
                }
                jx = jx - incx;
            }
        }
    }
}
else
{
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[j - 1] != zero)
            {
                if (nounit)
                {
                    x[j - 1] = x[j - 1] / a[j - 1, j - 1];
                }
                temp = x[j - 1];
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    x[i - 1] = x[i - 1] - temp * a[i - 1, j - 1];
                }
            }
        }
    }
}

```

```

    }
}
else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        if (x[jx - 1] != zero)
        {
            if (nounit)
            {
                x[jx - 1] = x[jx - 1] / a[j - 1, j - 1];
            }
            temp = x[jx - 1];
            ix = jx;
            for (int i = j + 1; i <= n; i = i + 1)
            {
                ix = ix + incx;
                x[ix - 1] = x[ix - 1] - temp * a[i - 1, j - 1];
            }
            jx = jx + incx;
        }
    }
}
}
else
{
    // Form x := inv( A**T ) * x or x := inv( A**H ) * x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = x[j - 1];
                if (noconj)
                {
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        temp = temp - a[i - 1, j - 1] * x[i - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / a[j - 1, j - 1];
                    }
                }
                else
                {
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        temp = temp - Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / Complex.Conjugate(a[j - 1, j - 1]);
                    }
                }
                x[j - 1] = temp;
            }
        }
        else
        {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = x[jx - 1];
                ix = kx;
                if (noconj)
                {
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        temp = temp - a[i - 1, j - 1] * x[ix - 1];
                        ix = ix + incx;
                    }
                    if (nounit)
                    {
                        temp = temp / a[j - 1, j - 1];
                    }
                }
                else
                {
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        temp = temp - Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
                        ix = ix + incx;
                    }
                    if (nounit)
                    {
                        temp = temp / Complex.Conjugate(a[j - 1, j - 1]);
                    }
                }
                x[jx - 1] = temp;
                jx = jx + incx;
            }
        }
    }
}
else
{
    if (incx == 1)
    {
        for (int j = n; j >= 1; j = j - 1)
        { //DO 140 J = N,1,-1
            temp = x[j - 1];
            if (noconj)
            {
                for (int i = n; i >= j + 1; i = i - 1)
                { //DO 130 I = N,J+1,-1
                    temp = temp - a[i - 1, j - 1] * x[i - 1];
                }
            }
        }
    }
}

```

```

        }
        if (nounit)
        {
            temp = temp / a[j - 1, j - 1];
        }
    }
    else
    {
        for (int i = n; i >= j + 1; i = i - 1)
        { //DO 130 I = N,J + 1,-1
            temp = temp - Complex.Conjugate(a[i - 1, j - 1]) * x[i - 1];
        }
        if (nounit)
        {
            temp = temp / Complex.Conjugate(a[j - 1, j - 1]);
        }
    }
    x[j - 1] = temp;
}
}
else
{
    kx = kx + (n - 1) * incx;
    jx = kx;
    for (int j = n; j >= 1; j = j - 1)
    { //DO 160 J = N,1,-1
        temp = x[jx - 1];
        ix = kx;
        if (noconj)
        {
            for (int i = n; i >= j + 1; i = i - 1)
            { //DO 150 I = N,J + 1,-1
                temp = temp - a[i - 1, j - 1] * x[ix - 1];
                ix = ix - incx;
            }
        }
        if (nounit)
        {
            temp = temp / a[j - 1, j - 1];
        }
    }
    else
    {
        for (int i = n; i >= j + 1; i = i - 1)
        { //DO 150 I = N,J + 1,-1
            temp = temp - Complex.Conjugate(a[i - 1, j - 1]) * x[ix - 1];
            ix = ix - incx;
        }
        if (nounit)
        {
            temp = temp / Complex.Conjugate(a[j - 1, j - 1]);
        }
    }
    x[jx - 1] = temp;
    jx = jx - incx;
}
}
}

}

public static void ZTBSV(string uplo, string trans, string diag, int n, int k, Complex[,] a, int lda, ref Complex[] x, int incx)
{
    //COMPLEX*16 A(LDA,*),X(*)
    // ZTBSV solves one of the systems of equations
    // A*x = b, or A**T*x = b, or A**H*x = b,
    // where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with ( k + 1 ) diagonals.
    // No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:
    //      UPLO = 'U' or 'u'  A is an upper triangular matrix.
    //      UPLO = 'L' or 'l'  A is a lower triangular matrix.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the equations to be solved as follows:
    //      TRANS = 'N' or 'n'  A*x = b.
    //      TRANS = 'T' or 't'  A**T*x = b.
    //      TRANS = 'C' or 'c'  A**H*x = b.

    // DIAG is CHARACTER*1
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    //      DIAG = 'U' or 'u'  A is assumed to be unit triangular.
    //      DIAG = 'N' or 'n'  A is not assumed to be unit triangular.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // K is INTEGER
    // On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.
    // On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading ( k + 1 ) by n part of the array A must contain the upper triangular band part of the
    matrix of coefficients, supplied column by
    // column, with the leading diagonal of the matrix in row ( k + 1 ) of the array, the first super-diagonal starting at position 2 in row k, and
    so on. The top left k by k triangle
    // of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix
    storage to band storage:
    //
    //      DO 20, J = 1, N
    //          M = K + 1 - J
    //          DO 10, I = MAX( 1, J - K ), J
    //              A( M + I, J ) = matrix( I, J )
    //          10 CONTINUE
    //      20 CONTINUE
    //
    // Before entry with UPLO = 'L' or 'l', the leading ( k + 1 ) by n part of the array A must contain the lower triangular band part of the
    matrix of coefficients, supplied column by
    // column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on.
    The bottom right k by k triangle of the
    // array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage
    to band storage:

```

```

//          DO 20, J = 1, N
//          M = 1 - J
//          DO 10, I = J, MIN( N, J + K )
//          A( M + I, J ) = matrix( I, J )
//          10 CONTINUE
//          20 CONTINUE

// Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but
are assumed to be unity.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ( k + 1 ).

// X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
// Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution
vector x.

// INCX is INTEGER
// On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

// Parameters
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp;
int info, ix, jx, kplus1, kx, l;
bool noconj, nounit;

kx = 0; //added
// Test the input parameters
info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T") && !(trans.Substring(0, 1).ToUpper() == "C"))
{
    info = 2;
}
else if !(diag.Substring(0, 1).ToUpper() == "U") && !(diag.Substring(0, 1).ToUpper() == "N"))
{
    info = 3;
}
else if (n < 0)
{
    info = 4;
}
else if (k < 0)
{
    info = 5;
}
else if (lda < (k + 1))
{
    info = 7;
}
else if (incx == 0)
{
    info = 9;
}
if (info != 0)
{
    XERBLA("ZTBSV ", info);
    return;
}

// Quick return if possible
if (n == 0)
{
    return;
}

noconj = (trans.Substring(0, 1).ToUpper() == "T");
nounit = (diag.Substring(0, 1).ToUpper() == "N");

// Set up the start point in X if the increment is not unity. This will be ( N - 1 ) * INCX too small for descending loops.
if (incx <= 0)
{
    kx = 1 - (n - 1) * incx;
}
else if (incx != 1)
{
    kx = 1;
}

// Start the operations. In this version the elements of A are accessed by sequentially with one pass through A.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form x := inv( A ) * x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kplus1 = k + 1;
        if (incx == 1)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                //DO 20 J = N,1,-1
                if (x[j - 1] != zero)
                {
                    l = kplus1 - j;
                    if (nounit)
                    {
                        x[j - 1] = x[j - 1] / a[kplus1 - 1, j - 1];
                    }
                    temp = x[j - 1];
                    for (int i = (j - 1); i >= Math.Max(1, j - k); i = i - 1)
                    {
                        x[i - 1] = x[i - 1] - temp * a[(1 + i) - 1, j - 1];
                    }
                }
            }
        }
        else
    }

```

```

{
    kx = kx + (n - 1) * incx;
    jx = kx;
    for (int j = n; j >= 1; j = j - 1)
    { //DO 40 J = N,1,-1
        kx = kx - incx;
        if (x[jx - 1] != zero)
        {
            ix = kx;
            l = kplus1 - j;
            if (nounit)
            {
                x[jx - 1] = x[jx - 1] / a[kplus1 - 1, j - 1];
            }
            temp = x[jx - 1];
            for (int i = (j - 1); i >= Math.Max(1, j - k); i = i - 1)
            { //DO 30 I = J - 1,MAX(1,J-K),-1
                x[ix - 1] = x[ix - 1] - temp * a[(l + i) - 1, j - 1];
                ix = ix - incx;
            }
            jx = jx - incx;
        }
    }
}

else
{
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[j - 1] != zero)
            {
                l = 1 - j;
                if (nounit)
                {
                    x[j - 1] = x[j - 1] / a[0, j - 1];
                }
                temp = x[j - 1];
                for (int i = (j + 1); i <= Math.Min(n, j + k); i = i + 1)
                {
                    x[i - 1] = x[i - 1] - temp * a[(l + i) - 1, j - 1];
                }
            }
        }
    }
    else
    {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1)
        {
            kx = kx + incx;
            if (x[jx - 1] != zero)
            {
                ix = kx;
                l = 1 - j;
                if (nounit)
                {
                    x[jx - 1] = x[jx - 1] / a[0, j - 1];
                }
                temp = x[jx - 1];
                for (int i = (j + 1); i <= Math.Min(n, j + k); i = i + 1)
                {
                    x[ix - 1] = x[ix - 1] - temp * a[(l + i) - 1, j - 1];
                    ix = ix + incx;
                }
            }
            jx = jx + incx;
        }
    }
}

else
{
    // Form x := inv( A**T ) * x or x := inv( A**H ) * x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kplus1 = k + 1;
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = x[j - 1];
                l = kplus1 - j;
                if (noconj)
                {
                    for (int i = Math.Max(1, j - k); i <= (j - 1); i = i + 1)
                    {
                        temp = temp - a[(l + i) - 1, j - 1] * x[i - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / a[kplus1 - 1, j - 1];
                    }
                }
            }
        }
        else
        {
            for (int i = Math.Max(1, j - k); i <= (j - 1); i = i + 1)
            {
                temp = temp - Complex.Conjugate(a[(l + i) - 1, j - 1]) * x[i - 1];
            }
            if (nounit)
            {
                temp = temp / Complex.Conjugate(a[kplus1 - 1, j - 1]);
            }
        }
    }
}
}

```

```

        x[j - 1] = temp;
    }
}
else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        temp = x[jx - 1];
        ix = kx;
        l = kplus1 - j;
        if (noconj)
        {
            for (int i = Math.Max(1, j - k); i <= (j - 1); i = i + 1)
            {
                temp = temp - a[(l + i) - 1, j - 1] * x[ix - 1];
                ix = ix + incx;
            }
            if (nounit)
            {
                temp = temp / a[kplus1 - 1, j - 1];
            }
        }
        else
        {
            for (int i = Math.Max(1, j - k); i <= (j - 1); i = i + 1)
            {
                temp = temp - Complex.Conjugate(a[(l + i) - 1, j - 1]) * x[ix - 1];
                ix = ix + incx;
            }
            if (nounit)
            {
                temp = temp / Complex.Conjugate(a[kplus1 - 1, j - 1]);
            }
        }
        x[jx - 1] = temp;
        jx = jx + incx;
        if (j > k)
        {
            kx = kx + incx;
        }
    }
}

else
{
    if (incx == 1)
    {
        for (int j = n; j >= 1; j = j - 1)
        { //DO 140 J = N,1,-1
            temp = x[j - 1];
            l = 1 - j;
            if (noconj)
            {
                for (int i = Math.Min(n, j + k); i >= (j + 1); i = i - 1)
                { //DO 130 I = MIN(N,J+K),J + 1,-1
                    temp = temp - a[(l + i) - 1, j - 1] * x[i - 1];
                }
                if (nounit)
                {
                    temp = temp / a[0, j - 1];
                }
            }
            else
            {
                for (int i = Math.Min(n, j + k); i >= (j + 1); i = i - 1)
                { //DO 130 I = MIN(N,J+K),J + 1,-1
                    temp = temp - Complex.Conjugate(a[(l + i) - 1, j - 1]) * x[i - 1];
                }
                if (nounit)
                {
                    temp = temp / Complex.Conjugate(a[0, j - 1]);
                }
            }
            x[j - 1] = temp;
        }
    }
    else
    {
        kx = kx + (n - 1) * incx;
        jx = kx;
        for (int j = n; j >= 1; j = j - 1)
        { //DO 160 J = N,1,-1
            temp = x[jx - 1];
            ix = kx;
            l = 1 - j;
            if (noconj)
            {
                for (int i = Math.Min(n, j + k); i >= (j + 1); i = i - 1)
                { //DO 150 I = MIN(N,J+K),J + 1,-1
                    temp = temp - a[(l + i) - 1, j - 1] * x[ix - 1];
                    ix = ix - incx;
                }
                if (nounit)
                {
                    temp = temp / a[0, j - 1];
                }
            }
            else
            {
                for (int i = Math.Min(n, j + k); i >= (j + 1); i = i - 1)
                { //DO 150 I = MIN(N,J+K),J + 1,-1
                    temp = temp - Complex.Conjugate(a[(l + i) - 1, j - 1]) * x[ix - 1];
                    ix = ix - incx;
                }
                if (nounit)
                {
                    temp = temp / Complex.Conjugate(a[0, j - 1]);
                }
            }
        }
    }
}

```



```

// Form x := inv( A ) * x.
if (uplo.Substring(0, 1).ToUpper() == "U")
{
    kk = (n * (n + 1)) / 2;
    if (incx == 1)
    {
        for (int j = n; j >= 1; j = j - 1)
        {
            //DO 20 J = N,1,-1
            if (x[j - 1] != zero)
            {
                if (nounit)
                {
                    x[j - 1] = x[j - 1] / ap[kk - 1];
                }
                temp = x[j - 1];
                k = kk - 1;
                for (int i = j - 1; i >= 1; i = i - 1)
                {
                    //DO 10 I = J - 1,1,-1
                    x[i - 1] = x[i - 1] - temp * ap[k - 1];
                    k = k - 1;
                }
            }
            kk = kk - j;
        }
    }
    else
    {
        jx = kx + (n - 1) * incx;
        for (int j = n; j >= 1; j = j - 1)
        {
            //DO 40 J = N,1,-1
            if (x[jx - 1] != zero)
            {
                if (nounit)
                {
                    x[jx - 1] = x[jx - 1] / ap[kk - 1];
                }
                temp = x[jx - 1];
                ix = jx;
                for (k = kk - 1; k >= kk - j + 1; k = k - 1)
                {
                    //DO 30 K = KK - 1, KK - J + 1, -1
                    ix = ix - incx;
                    x[ix - 1] = x[ix - 1] - temp * ap[k - 1];
                }
            }
            jx = jx - incx;
            kk = kk - j;
        }
    }
}

else
{
    kk = 1;
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[j - 1] != zero)
            {
                if (nounit)
                {
                    x[j - 1] = x[j - 1] / ap[kk - 1];
                }
                temp = x[j - 1];
                k = kk + 1;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    x[i - 1] = x[i - 1] - temp * ap[k - 1];
                    k = k + 1;
                }
            }
            kk = kk + (n - j + 1);
        }
    }
    else
    {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                if (nounit)
                {
                    x[jx - 1] = x[jx - 1] / ap[kk - 1];
                }
                temp = x[jx - 1];
                ix = jx;
                for (k = kk + 1; k <= kk + n - j; k = k + 1)
                {
                    ix = ix + incx;
                    x[ix - 1] = x[ix - 1] - temp * ap[k - 1];
                }
            }
            jx = jx + incx;
            kk = kk + (n - j + 1);
        }
    }
}

else
{
    // Form x := inv( A**T ) * x or x := inv( A**H ) * x.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        kk = 1;
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = x[j - 1];
                k = kk;
                if (noconj)
                {

```

```

        for (int i = 1; i <= j - 1; i = i + 1)
        {
            temp = temp - ap[k - 1] * x[i - 1];
            k = k + 1;
        }
        if (nounit)
        {
            temp = temp / ap[kk + j - 1 - 1];
        }
    }
    else
    {
        for (int i = 1; i <= j - 1; i = i + 1)
        {
            temp = temp - Complex.Conjugate(ap[k - 1]) * x[i - 1];
            k = k + 1;
        }
        if (nounit)
        {
            temp = temp / Complex.Conjugate(ap[kk + j - 1 - 1]);
        }
    }
    x[j - 1] = temp;
    kk = kk + j;
}
}
}

else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        temp = x[jx - 1];
        ix = kx;
        if (noconj)
        {
            for (k = kk; k <= kk + j - 2; k = k + 1)
            {
                temp = temp - ap[k - 1] * x[ix - 1];
                ix = ix + incx;
            }
            if (nounit)
            {
                temp = temp / ap[kk + j - 1 - 1];
            }
        }
        else
        {
            for (k = kk; k <= kk + j - 2; k = k + 1)
            {
                temp = temp - Complex.Conjugate(ap[k - 1]) * x[ix - 1];
                ix = ix + incx;
            }
            if (nounit)
            {
                temp = temp / Complex.Conjugate(ap[kk + j - 1 - 1]);
            }
        }
        x[jx - 1] = temp;
        jx = jx + incx;
        kk = kk + j;
    }
}

}

else
{
    kk = (n * (n + 1)) / 2;
    if (incx == 1)
    {
        for (int j = n; j >= 1; j = j - 1)
        { //DO 140 J = N,1,-1
            temp = x[j - 1];
            k = kk;
            if (noconj)
            {
                for (int i = n; i >= j + 1; i = i - 1)
                {
                    temp = temp - ap[k - 1] * x[i - 1];
                    k = k - 1;
                }
                if (nounit)
                {
                    temp = temp / ap[kk - n + j - 1];
                }
            }
            else
            {
                for (int i = n; i >= j + 1; i = i - 1)
                {
                    temp = temp - Complex.Conjugate(ap[k - 1]) * x[i - 1];
                    k = k - 1;
                }
                if (nounit)
                {
                    temp = temp / Complex.Conjugate(ap[kk - n + j - 1]);
                }
            }
            x[j - 1] = temp;
            kk = kk - (n - j + 1);
        }
    }
}

else
{
    kx = kx + (n - 1) * incx;
    jx = kx;
    for (int j = n; j >= 1; j = j - 1)
    { //DO 160 J = N,1,-1
        temp = x[jx - 1];
        ix = kx;
    }
}
}

```

```

        if (noconj)
        {
            for (k = kk; k >= kk - (n - (j + 1)); k = k - 1)
            {
                //DO 150 K = KK, KK - (N- (J+1)), -1
                temp = temp - ap[k - 1] * x[ix - 1];
                ix = ix - incx;
            }
            if (nounit)
            {
                temp = temp / ap[kk - n + j - 1];
            }
        }
        else
        {
            for (k = kk; k >= kk - (n - (j + 1)); k = k - 1)
            {
                //DO 150 K = KK, KK - (N- (J+1)), -1
                temp = temp - Complex.Conjugate(ap[k - 1]) * x[ix - 1];
                ix = ix - incx;
            }
            if (nounit)
            {
                temp = temp / Complex.Conjugate(ap[kk - n + j - 1]);
            }
        }
        x[jx - 1] = temp;
        jx = jx - incx;
        kk = kk - (n - j + 1);
    }
}

}

}

public static void ZGERC(int m, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[,] a, int lda)
{
    //COMPLEX*16 A(LDA,*),X(*),Y(*)

    // ZGERC performs the rank 1 operation
    // A := alpha*x*y**H + A,
    // where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

    // M is INTEGER
    // On entry, M specifies the number of rows of the matrix A. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of the matrix A. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( m - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the m element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated
matrix.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, m ).

    // Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jy, kx;

    // Test the input parameters
    info = 0;
    if (m < 0)
    {
        info = 1;
    }
    else if (n < 0)
    {
        info = 2;
    }
    else if (incx == 0)
    {
        info = 5;
    }
    else if (incy == 0)
    {
        info = 7;
    }
    else if (lda < Math.Max(1, m))
    {
        info = 9;
    }
    if (info != 0)
    {
        XERBLA("ZGERC ", info);
        return;
    }

    // Quick return if possible
    if ((m == 0) || (n == 0) || (alpha == zero))
    {
        return;
    }

    // Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
    if (incy > 0)
    {
        jy = 1;
    }

```

```

else
{
    jy = 1 - (n - 1) * incy;
}
if (incx == 1)
{
    for (int j = 1; j <= n; j = j + 1)
    {
        if (y[jy - 1] != zero)
        {
            temp = alpha * Complex.Conjugate(y[jy - 1]);
            for (int i = 1; i <= m; i = i + 1)
            {
                a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp;
            }
            jy = jy + incy;
        }
    }
}
else
{
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (m - 1) * incx;
    }
    for (int j = 1; j <= n; j = j + 1)
    {
        if (y[jy - 1] != zero)
        {
            temp = alpha * Complex.Conjugate(y[jy - 1]);
            ix = kx;
            for (int i = 1; i <= m; i = i + 1)
            {
                a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp;
                ix = ix + incx;
            }
            jy = jy + incy;
        }
    }
}
}
public static void ZGERU(int m, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[, ] a, int lda)
{
    // COMPLEX*16 A(LDA,*),X(*),Y(*)
    // ZGERU performs the rank 1 operation
    // A := alpha*x*y**T + A,
    // where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

    // M is INTEGER
    // On entry, M specifies the number of rows of the matrix A. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of the matrix A. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( m - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the m element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated
matrix.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, m ).

    // Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jy, kx;

    // Test the input parameters
    info = 0;
    if (m < 0)
    {
        info = 1;
    }
    else if (n < 0)
    {
        info = 2;
    }
    else if (incx == 0)
    {
        info = 5;
    }
    else if (incy == 0)
    {
        info = 7;
    }
    else if (lda < Math.Max(1, m))
    {
        info = 9;
    }
    if (info != 0)
    {
        XERBLA("ZGERU ", info);
        return;
    }
}

```

```

// Quick return if possible
if ((m == 0) || (n == 0) || (alpha == zero))
{
    return;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through A.
if (incy > 0)
{
    jy = 1;
}
else
{
    jy = 1 - (n - 1) * incy;
}
if (incx == 1)
{
    for (int j = 1; j <= n; j = j + 1)
    {
        if (y[jy - 1] != zero)
        {
            temp = alpha * y[jy - 1];
            for (int i = 1; i <= m; i = i + 1)
            {
                a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp;
            }
            jy = jy + incy;
        }
    }
}
else
{
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (m - 1) * incx;
    }
    for (int j = 1; j <= n; j = j + 1)
    {
        if (y[jy - 1] != zero)
        {
            temp = alpha * y[jy - 1];
            ix = kx;
            for (int i = 1; i <= m; i = i + 1)
            {
                a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp;
                ix = ix + incx;
            }
            jy = jy + incy;
        }
    }
}
}

public static void ZHER(string uplo, int n, double alpha, Complex[] x, int incx, ref Complex[,] a, int lda)
{
    //COMPLEX*16 A(LDA,*),X(*)
    // ZHER performs the hermitian rank 1 operation
    // A := alpha*x*x**H + A,
    // where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // ALPHA is DOUBLE PRECISION.
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // A is COMPLEX*16 array of DIMENSION ( LDA, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the
    hermitian matrix and the strictly
    // lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part
    of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the
    hermitian matrix and the strictly
    // upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part
    of the updated matrix.
    // Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

    // Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jx, kx;
    kx = 0; // added
    // Test the input parameters
    info = 0;
    if (! (uplo.Substring(0, 1).ToUpper() == "U") && ! (uplo.Substring(0, 1).ToUpper() == "L"))
    {
        info = 1;
    }
    else if (n < 0)
    {
        info = 2;
    }
    else if (incx == 0)
    {
        info = 3;
    }
}

```

```

        info = 5;
    }
    else if (lda < Math.Max(1, n))
    {
        info = 7;
    }

    if (info != 0)
    {
        XERBLA("ZHER ", info);
        return;
    }

    // Quick return if possible
    if ((n == 0) || (alpha == zero.Real))
    {
        return;
    }

    // Set the start point in X if the increment is not unity.
    if (incx <= 0)
    {
        kx = 1 - (n - 1) * incx;
    }
    else if (incx != 1)
    {
        kx = 1;
    }

    // Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        // Form A when A is stored in upper triangle.
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[j - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(x[j - 1]);
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp;
                    }
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real + Complex.Multiply(x[j - 1], temp).Real;
                }
                else
                {
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real;
                }
            }
        }
        else
        {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[jx - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(x[jx - 1]);
                    ix = kx;
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp;
                        ix = ix + incx;
                    }
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real + Complex.Multiply(x[jx - 1], temp).Real;
                }
                else
                {
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real;
                }
                jx = jx + incx;
            }
        }
    }
    else
    {
        // Form A when A is stored in lower triangle.
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[j - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(x[j - 1]);
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real + (temp * x[j - 1]).Real;
                    for (int i = j + 1; i <= n; i = i + 1)
                    {
                        a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp;
                    }
                }
                else
                {
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real;
                }
            }
        }
        else
        {
            jx = kx;
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[jx - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(x[jx - 1]);
                    a[j - 1, j - 1] = a[j - 1, j - 1].Real + (temp * x[jx - 1]).Real;
                    ix = jx;
                    for (int i = j + 1; i <= n; i = i + 1)
                    {

```

```

        ix = ix + incx;
        a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp;
    }
}
else
{
    a[j - 1, j - 1] = a[j - 1, j - 1].Real;
}
jx = jx + incx;
}
}
}

public static void ZHPR(string uplo, int n, double alpha, Complex[] x, int incx, ref Complex[] ap)
{
    // COMPLEX*16 AP(*),X(*)
    // ZHPR performs the hermitian rank 1 operation
    // A := alpha*x*x**H + A,
    // where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix, supplied in packed form.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:
    //     UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.
    //     UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // ALPHA is DOUBLE PRECISION.
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // AP is COMPLEX*16 array of DIMENSION at least ( ( n * ( n + 1 ) ) / 2 ).
    // Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially,
    // column by column, so that AP( 1 )
    // contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. On exit, the array AP is overwritten by the
    // upper triangular part of the
    // updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix
    // packed sequentially, column by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respectively,
    // and so on. On exit, the array
    // AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be
    // set, they are assumed to be zero,
    // and on exit they are set to zero.

    // Parameters
    Complex zero = new Complex(0.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, ix, jx, k, kk, kx;
    kx = 0; // added

    // Test the input parameters
    info = 0;
    if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
    {
        info = 1;
    }
    else if (n < 0)
    {
        info = 2;
    }
    else if (incx == 0)
    {
        info = 5;
    }
    if (info != 0)
    {
        XERBLA("ZHPR ", info);
        return;
    }

    // Quick return if possible
    if ((n == 0) || (alpha == zero.Real))
    {
        return;
    }

    // Set the start point in X if the increment is not unity
    if (incx <= 0)
    {
        kx = 1 - (n - 1) * incx;
    }
    else if (incx != 1)
    {
        kx = 1;
    }

    // Start the operations. In this version the elements of the array AP are accessed sequentially with one pass through AP.
    kk = 1;
    if (uplo.Substring(0, 1).ToUpper() == "U")
    {
        // Form A when upper triangle is stored in AP.
        if (incx == 1)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (x[j - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(x[j - 1]);
                    k = kk;
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        ap[k - 1] = ap[k - 1] + x[i - 1] * temp;
                        k = k + 1;
                    }
                }
            }
        }
    }
}

```

```

        ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real + Complex.Multiply(x[j - 1], temp).Real;
    }
    else
    {
        ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real;
    }
    kk = kk + j;
}
}
else
{
    jx = kx;
    for (int j = 1; j <= n; j = j + 1)
    {
        if (x[jx - 1] != zero)
        {
            temp = alpha * Complex.Conjugate(x[jx - 1]);
            ix = kx;
            for (k = kk; k <= (kk + j - 2); k = k + 1)
            {
                ap[k - 1] = ap[k - 1] + x[ix - 1] * temp;
                ix = ix + incx;
            }
            ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real + Complex.Multiply(x[jx - 1], temp).Real;
        }
        else
        {
            ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real;
        }
        jx = jx + incx;
        kk = kk + j;
    }
}

else
{
    // Form A when lower triangle is stored in AP.
    if (incx == 1)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[j - 1] != zero)
            {
                temp = alpha * Complex.Conjugate(x[j - 1]);
                ap[kk - 1] = ap[kk - 1].Real + Complex.Multiply(temp, x[j - 1]).Real;

                k = kk + 1;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    ap[k - 1] = ap[k - 1] + x[i - 1] * temp;
                    k = k + 1;
                }
            }
            else
            {
                ap[kk - 1] = ap[kk - 1].Real;
            }
            kk = kk + n - j + 1;
        }
    }
    else
    {
        jx = kx;
        for (int j = 1; j <= n; j = j + 1)
        {
            if (x[jx - 1] != zero)
            {
                temp = alpha * Complex.Conjugate(x[jx - 1]);
                ap[kk - 1] = ap[kk - 1].Real + Complex.Multiply(temp, x[jx - 1]).Real;
                ix = jx;
                for (k = kk + 1; k <= (kk + n - j); k = k + 1)
                {
                    ix = ix + incx;
                    ap[k - 1] = ap[k - 1] + x[ix - 1] * temp;
                }
            }
            else
            {
                ap[kk - 1] = ap[kk - 1].Real;
            }
            jx = jx + incx;
            kk = kk + n - j + 1;
        }
    }
}

}

public static void ZHPR2(string uplo, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[] ap)
{
    // COMPLEX*16 AP(*),X(*),Y(*)
    // ZHPR2 performs the hermitian rank 2 operation
    // A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
    // where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:
    // UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.
    // UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

```

```

// Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
// Before entry, the incremented array Y must contain the n element vector y.

// INCY is INTEGER
// On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

// AP is COMPLEX*16 array of DIMENSION at least ( ( n * ( n + 1 ) ) / 2 ).
// Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially,
column by column, so that AP( 1 )
// contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 ) respectively, and so on. On exit, the array AP is overwritten by the
upper triangular part of the
// updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix
// packed sequentially, column by column, so that AP( 1 ) contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 ) respectively,
and so on. On exit, the array
// AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need
// not be set, they are assumed to be zero, and on exit they are set to zero.

// Parameters
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp1, temp2;
int info, ix, iy, jx, jy, k, kk, kx, ky;
kx = 0; // added
ky = 0; // added
jx = 0; // added
jy = 0; // added

// Test the input parameters
info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (n < 0)
{
    info = 2;
}
else if (incx == 0)
{
    info = 5;
}
else if (incy == 0)
{
    info = 7;
}
if (info != 0)
{
    XERBLA("ZHPR2 ", info);
    return;
}

// Quick return if possible
if ((n == 0) || (alpha == zero))
{
    return;
}

// Set up the start points in X and Y if the increments are not both unity
if ((incx != 1) || (incy != 1))
{
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (n - 1) * incx;
    }

    if (incy > 0)
    {
        ky = 1;
    }
    else
    {
        ky = 1 - (n - 1) * incy;
    }
    jx = kx;
    jy = ky;
}

// Start the operations. In this version the elements of the array AP are accessed sequentially with one pass through AP.
kk = 1;
if (uplo.Substring(0, 1).ToUpper() == "U")
{
    // Form A when upper triangle is stored in AP
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[j - 1] != zero) || (y[j - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[j - 1]);
                temp2 = Complex.Conjugate(alpha * x[j - 1]);
                k = kk;
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    ap[k - 1] = ap[k - 1] + x[i - 1] * temp1 + y[i - 1] * temp2;
                    k = k + 1;
                }
                ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real + (x[j - 1] * temp1 + y[j - 1] * temp2).Real;
            }
            else
            {
                ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real;
            }
            kk = kk + j;
        }
    }
    else

```

```

    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[jx - 1] != zero) || (y[jy - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[jy - 1]);
                temp2 = Complex.Conjugate(alpha * x[jx - 1]);
                ix = kx;
                iy = ky;
                for (k = kk; k <= (kk + j - 2); k = k + 1)
                {
                    ap[k - 1] = ap[k - 1] + x[ix - 1] * temp1 + y[iy - 1] * temp2;
                    ix = ix + incx;
                    iy = iy + incy;
                }
                ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real + (x[jx - 1] * temp1 + y[jy - 1] * temp2).Real;
            }
            else
            {
                ap[kk + j - 1 - 1] = ap[kk + j - 1 - 1].Real;
            }
            jx = jx + incx;
            jy = jy + incy;
            kk = kk + j;
        }
    }
}

else
{
    // Form A when lower triangle is stored in AP
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[j - 1] != zero) || (y[j - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[j - 1]);
                temp2 = Complex.Conjugate(alpha * x[j - 1]);
                ap[kk - 1] = ap[kk - 1].Real + (x[j - 1] * temp1 + y[j - 1] * temp2).Real;
                k = kk + 1;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    ap[k - 1] = ap[k - 1] + x[i - 1] * temp1 + y[i - 1] * temp2;
                    k = k + 1;
                }
            }
            else
            {
                ap[kk - 1] = ap[kk - 1].Real;
            }
            kk = kk + n - j + 1;
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[jx - 1] != zero) || (y[jy - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[jy - 1]);
                temp2 = Complex.Conjugate(alpha * x[jx - 1]);
                ap[kk - 1] = ap[kk - 1].Real + (x[jx - 1] * temp1 + y[jy - 1] * temp2).Real;
                ix = jx;
                iy = jy;
                for (k = kk + 1; k <= (kk + n - j); k = k + 1)
                {
                    ix = ix + incx;
                    iy = iy + incy;
                    ap[k - 1] = ap[k - 1] + x[ix - 1] * temp1 + y[iy - 1] * temp2;
                }
            }
            else
            {
                ap[kk - 1] = ap[kk - 1].Real;
            }
            jx = jx + incx;
            jy = jy + incy;
            kk = kk + n - j + 1;
        }
    }
}
}

public static void ZHER2(string uplo, int n, Complex alpha, Complex[] x, int incx, Complex[] y, int incy, ref Complex[,] a, int lda)
{
    // COMPLEX*16 A(LDA,*),X(*),Y(*)
    // ZHER2 performs the hermitian rank 2 operation
    // A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
    // where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:
    //      UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.
    //      UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

    // N is INTEGER
    // On entry, N specifies the order of the matrix A. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // X is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCX ) ).
    // Before entry, the incremented array X must contain the n element vector x.

    // INCX is INTEGER
    // On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

    // Y is COMPLEX*16 array of dimension at least ( 1 + ( n - 1 ) * abs( INCY ) ).
    // Before entry, the incremented array Y must contain the n element vector y.

    // INCY is INTEGER
    // On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

```

```

// A is COMPLEX*16 array of DIMENSION ( LDA, n ).
// Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the
hermitian matrix and the strictly
// lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part
of the updated matrix.
// Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the
hermitian matrix and the strictly
// upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part
of the updated matrix.
// Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max( 1, n ).

// Parameters
Complex zero = new Complex(0.0, 0.0);

// Local Scalars
Complex temp1, temp2;
int info, ix, iy, jx, jy, kx, ky;
jx = jy = kx = ky = 0; // added
// Test the input parameters
info = 0;
if (!(uplo.Substring(0, 1).ToUpper() == "U") && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (n < 0)
{
    info = 2;
}
else if (incx == 0)
{
    info = 5;
}
else if (incy == 0)
{
    info = 7;
}
else if (lda < Math.Max(1, n))
{
    info = 9;
}

if (info != 0)
{
    XERBLA("ZHER2 ", info);
    return;
}

// Quick return if possible
if ((n == 0) || (alpha == zero))
{
    return;
}

// Set the start point in X and Y if the increment is not both unity.

if ((incx != 1) || (incy != 1))
{
    if (incx > 0)
    {
        kx = 1;
    }
    else
    {
        kx = 1 - (n - 1) * incx;
    }
    if (incy > 0)
    {
        ky = 1;
    }
    else
    {
        ky = 1 - (n - 1) * incy;
    }
    jx = kx;
    jy = ky;
}

// Start the operations. In this version the elements of A are accessed sequentially with one pass through the triangular part of A.

if (uplo.Substring(0, 1).ToUpper() == "U")
{
    // Form A when A is stored in upper triangle.

    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[j - 1] != zero) || (y[j - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[j - 1]);
                temp2 = Complex.Conjugate(alpha * x[j - 1]);
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp1 + y[i - 1] * temp2;
                }
                a[j - 1, j - 1] = a[j - 1, j - 1].Real + (x[j - 1] * temp1 + y[j - 1] * temp2).Real;
            }
            else
            {
                a[j - 1, j - 1] = a[j - 1, j - 1].Real;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[jx - 1] != zero) || (y[jy - 1] != zero))

```

```

        {
            temp1 = alpha * Complex.Conjugate(y[jy - 1]);
            temp2 = Complex.Conjugate(alpha * x[jx - 1]);
            ix = kx;
            iy = ky;
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp1 + y[iy - 1] * temp2;
                ix = ix + incx;
                iy = iy + incy;
            }
            a[j - 1, j - 1] = a[j - 1, j - 1].Real + (x[jx - 1] * temp1 + y[jy - 1] * temp2).Real;
        }
        else
        {
            a[j - 1, j - 1] = a[j - 1, j - 1].Real;
        }
        jx = jx + incx;
        jy = jy + incy;
    }
}

else
{
    // Form A when A is stored in lower triangle.
    if ((incx == 1) && (incy == 1))
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[j - 1] != zero) || (y[j - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[j - 1]);
                temp2 = Complex.Conjugate(alpha * x[j - 1]);
                a[j - 1, j - 1] = a[j - 1, j - 1].Real + (x[j - 1] * temp1 + y[j - 1] * temp2).Real;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    a[i - 1, j - 1] = a[i - 1, j - 1] + x[i - 1] * temp1 + y[i - 1] * temp2;
                }
            }
            else
            {
                a[j - 1, j - 1] = a[j - 1, j - 1].Real;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if ((x[jx - 1] != zero) || (y[jy - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(y[jy - 1]);
                temp2 = Complex.Conjugate(alpha * x[jx - 1]);
                a[j - 1, j - 1] = a[j - 1, j - 1].Real + (x[jx - 1] * temp1 + y[jy - 1] * temp2).Real;
                ix = jx;
                iy = jy;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    ix = ix + incx;
                    iy = iy + incy;
                    a[i - 1, j - 1] = a[i - 1, j - 1] + x[ix - 1] * temp1 + y[iy - 1] * temp2;
                }
            }
            else
            {
                a[j - 1, j - 1] = a[j - 1, j - 1].Real;
            }
            jx = jx + incx;
            jy = jy + incy;
        }
    }
}

}

public static void ZBLAT3() //???TEST uses misc_d.eps, DLAMCH/ first
{
    /*
    // C# version of Test program for the COMPLEX*16 Level 3 Blas.

    /* The program must be driven by a short data file. The first 14 records
    of the file are read using list-directed input, the last 9 records
    are read using the format ( A6, L2 ). An annotated example of a data
    file can be obtained by deleting the first 3 characters from the
    following 23 lines:
    'zblat3.out'      NAME OF SUMMARY OUTPUT FILE
    6                 UNIT NUMBER OF SUMMARY FILE
    'ZBLAT3.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
    -1                UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
    F                 LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
    F                 LOGICAL FLAG, T TO STOP ON FAILURES.
    T                 LOGICAL FLAG, T TO TEST ERROR EXITS.
    16.0              THRESHOLD VALUE OF TEST RATIO
    6                 NUMBER OF VALUES OF N
    0 1 2 3 5 9      VALUES OF N
    3                 NUMBER OF VALUES OF ALPHA
    (0.0,0.0) (1.0,0.0) (0.7,-0.9)      VALUES OF ALPHA
    3                 NUMBER OF VALUES OF BETA
    (0.0,0.0) (1.0,0.0) (1.3,-1.1)      VALUES OF BETA
    ZGEMM T PUT F FOR NO TEST. SAME COLUMNS.
    ZHEMM T PUT F FOR NO TEST. SAME COLUMNS.
    ZSYMM T PUT F FOR NO TEST. SAME COLUMNS.
    ZTRMM T PUT F FOR NO TEST. SAME COLUMNS.
    ZTRSM T PUT F FOR NO TEST. SAME COLUMNS.
    ZHERK T PUT F FOR NO TEST. SAME COLUMNS.
    ZSYRK T PUT F FOR NO TEST. SAME COLUMNS.
    ZHER2K T PUT F FOR NO TEST. SAME COLUMNS.

```

```

ZSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
*/

// Parameters
//const int nin = 5;
const int nsubs = 9;
Complex zero = new Complex(0.0, 0.0);
Complex one = new Complex(1.0, 0.0);
const double rzero = 0.0;

const int nmax = 65;
//const int incmax = 2;

//const int ninmax = 7;
const int nidmax = 9;
//const int nkbmax = 7;
const int nalmax = 7;
const int nbemax = 7;

// Local Scalars
double eps, err, thresh;
int n, nalf, nbet, nidim, nout, ntra;
//i, isnum, j, n, nalf, nbet, nidim, nout, ntra;
bool fatal = false, ltestt, rew1, same, sfatal, trace, tsterr;

string transa, transb, snamet, snaps, summy;

// Local Arrays
//Complex[,] a = new Complex[nmax,nmax];
Complex[,] aa = new Complex[nmax * nmax];
Complex[,] ab = new Complex[nmax, 2 * nmax];
Complex[] alf = new Complex[nalmax];
Complex[] _as = new Complex[nmax * nmax];
Complex[] bb = new Complex[nmax * nmax];
Complex[] bet = new Complex[nbemax];
Complex[] bs = new Complex[nmax * nmax];
Complex[,] c = new Complex[nmax, nmax];
Complex[] cc = new Complex[nmax * nmax];
Complex[] cs = new Complex[nmax * nmax];
Complex[] ct = new Complex[nmax];
Complex[] w = new Complex[2 * nmax];

double[] g = new double[nmax];
int[] idim = new int[nidmax];
//int[] inc = new int[ninmax];
//int[] kb = new int[nkbmax];

bool[] ltest = new bool[nsubs];
string[] snames = {
    "ZGEMM ",
    "ZHEMM ",
    "ZSYMM ",
    "ZTRMM ",
    "ZTRSM ",
    "ZHERK ",
    "ZSYRK ",
    "ZHER2K",
    "ZSYR2K" };

// file I/O
//String fin = @"..\dbl3.in";
String fin = @"STDIN";
ArrayList tmp = new ArrayList();
String[] findata = { };
String[] stmp_ld;
String stmp;

StreamWriter fout1, fout2;

Complex[,] tmp2d_ab1, tmp2d_ab2;
Complex[] tmp1d_ab;
err = 0; //added
// Common blocks
//COMMON          /INFOC/INFOT, NOUTC, OK, LERR
//COMMON          /SRNAMC/SRNAMT

// Executable Statements
// read input file
try
{
    tmp.Clear();
    //using (StreamReader infile = new StreamReader(fin))
    using (StreamReader infile = new StreamReader(Console.OpenStandardInput()))
    {
        string line;
        int i = 0;
        while ((line = infile.ReadLine()) != null)
        {
            tmp.Add(line.Trim());
            Console.WriteLine("Read:{0}", line.Trim());
        }
        findata = (String[])tmp.ToArray(typeof(string));
    }
}
catch (Exception e)
{
    Console.WriteLine("Error Reading {0}", fin);
    Environment.Exit(1);
}

// Read name and unit number for summary output file and open file.

summy = findata[0].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
nout = int.Parse(findata[1].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

fout1 = new StreamWriter(summy, false); // true = append
//misc_d.nout = fout1; // for XERBLA
misc_z.nout = fout1; // for XERBLA
fout2 = null;
noutc = nout;

```

```

// Read name and unit number for snapshot output file and open file.

snaps = findata[2].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].Replace("'", "");
ntra = int.Parse(findata[3].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

trace = (ntra >= 0);

//Console.WriteLine("ntra: {0}, trace: {1}", ntra, trace);
if (trace)
{
    fout2 = new StreamWriter(snaps, true); // true = append
}
// Read the flag that directs rewinding of the snapshot file.
//Console.WriteLine("4052: {0}", findata[4]);
//Console.WriteLine("4052: {0}", findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
//rewi = bool.Parse(findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[4].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    rewi = true;
}
else
{
    rewi = false;
}
//Console.WriteLine("4052: {0}", rewi);
rewi = rewi & trace;

// Read the flag that directs stopping on any failure.
//sfatal = bool.Parse(findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[5].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    sfatal = true;
}
else
{
    sfatal = false;
}
// Read the flag that indicates whether error exits are to be tested.
//tsterr = bool.Parse(findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if (findata[6].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0].ToUpper() == "T")
{
    tsterr = true;
}
else
{
    tsterr = false;
}
// Read the threshold value of the test ratio
thresh = double.Parse(findata[7].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

// Read and check the parameter values for the tests.
// Values of N
nidim = int.Parse(findata[8].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);

if ((nidim < 1) || nidim > nidmax)
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "N", nidmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4095:");
    Environment.Exit(1);
}

for (int i = 1; i <= nidim; i = i + 1)
{
    idim[i - 1] = int.Parse(findata[9].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    if ((idim[i - 1] < 0) || (idim[i - 1] > nmax))
    {
        fout1.WriteLine("VALUE OF N IS LESS THAN 0 OR GREATER THAN {0,2:D}", nmax);
        fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        Console.WriteLine("4111:");
        Environment.Exit(1);
    }
}

// Values of ALPHA
nalf = int.Parse(findata[10].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nalf < 1) || (nalf > nalfmax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "ALPHA", nalfmax);
    fout1.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
    if (trace)
    {
        fout1.Flush();
        fout1.Close();
    }
    Console.WriteLine("4191:");
    Environment.Exit(1);
}

for (int i = 1; i <= nalf; i = i + 1)
{
    //alf[i - 1] = Complex.Parse(findata[15].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    stmp = findata[11].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1];
    //Console.WriteLine("{0}", stmp);
    stmp_ld = stmp.Substring(1, stmp.Length - 2).Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
    //Console.WriteLine("{0} {1}", stmp_ld[0], stmp_ld[1]);
    alf[i - 1] = new Complex(double.Parse(stmp_ld[0]), double.Parse(stmp_ld[1]));
    //Console.WriteLine("{0}", alf[i - 1]);
}

// Values of BETA
nbet = int.Parse(findata[12].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0]);
if ((nbet < 1) || (nbet > nbemax))
{
    fout1.WriteLine("NUMBER OF VALUES OF {0} IS LESS THAN 1 OR GREATER THAN {1,2:D}", "BETA", nbemax);
}

```

```

foutl.WriteLine("AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM\n***** TESTS ABANDONED *****"); // 230
if (trace)
{
    foutl.Flush();
    foutl.Close();
}
Console.WriteLine("4210:");
Environment.Exit(1);
}

for (int i = 1; i <= nbet; i = i + 1)
{
    //bet[i - 1] = double.Parse(findata[17].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1]);
    stmp = findata[13].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[i - 1];
    stmp_ld = stmp.Substring(1, stmp.Length - 2).Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
    bet[i - 1] = new Complex(double.Parse(stmp_ld[0]), double.Parse(stmp_ld[1]));
    //Console.WriteLine("{0}", bet[i - 1]);
}
// Report values of parameters
foutl.WriteLine("TESTS OF THE COMPLEX LEVEL 3 BLAS \r\n\r\n THE FOLLOWING PARAMETER VALUES WILL BE USED:");
foutl.Write(" FOR N");
for (int i = 1; i <= nidim; i = i + 1)
{
    foutl.Write("{0,6:D}", idim[i - 1]);
}
foutl.WriteLine();

foutl.Write(" FOR ALPHA");
for (int i = 1; i <= nalf; i = i + 1)
{
    foutl.Write("{0}", alf[i - 1].ToString("F1"));
    //7( '(', F4.1, ',', F4.1, ') ', : )
}
foutl.WriteLine();
foutl.Write(" FOR BETA");
for (int i = 1; i <= nbet; i = i + 1)
{
    foutl.Write("{0}", bet[i - 1].ToString("F1"));
    //7( '(', F4.1, ',', F4.1, ') ', : )
}

if (!tsterr)
{
    foutl.WriteLine();
    foutl.WriteLine("ERROR-EXITS WILL NOT BE TESTED");
}
foutl.WriteLine();
foutl.WriteLine("ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LESS THAN {0,8:F2}", thresh);
foutl.WriteLine();

// Read names of subroutines and flags which indicate whether they are to be tested.
for (int i = 1; i <= nsubs; i = i + 1)
{
    ltest[i - 1] = false;
}
for (int i = 1; i <= nsubs; i = i + 1)
{
    snamet = findata[14 + i - 1].Substring(0, 6);
    //ltestt = bool.Parse(findata[18 + i - 1].Substring(6, 2).Trim());
    if (findata[14 + i - 1].Substring(6, 2).Trim().ToUpper() == "T")
    {
        ltestt = true;
    }
    else
    {
        ltestt = false;
    }
    if (snamet == snames[i - 1])
    {
        ltest[i - 1] = ltestt;
    }
    else
    {
        foutl.WriteLine("SUBPROGRAM NAME {0} NOT RECOGNIZED\n***** TESTS ABANDONED *****", snamet);
        foutl.Flush();
        foutl.Close();
        Console.WriteLine("4276:");
        Environment.Exit(1);
    }
}

// Compute EPS (the machine precision).
// uses misc_d.eps, call DLAMCH first
//eps = misc_d.eps;
eps = misc_d.prec; // EPS = EPSILON(RZERO)
foutl.WriteLine("RELATIVE MACHINE PRECISION IS TAKEN TO BE {0,9:E1}", eps);
//Console.WriteLine("{0} {1} {0,9:E1}", eps, eps * 10, eps * 10);
//Console.WriteLine("4295:");

// Check the reliability of ZMMCH using exact data.
n = Math.Min(32, nmax);
for (int j = 1; j <= n; j = j + 1)
{
    for (int i = 1; i <= n; i = i + 1)
    {
        ab[i - 1, j - 1] = Math.Max(i - j + 1, 0);
    }
    ab[j - 1, (nmax + 1) - 1] = j;
    ab[0, (nmax + j) - 1] = j;
    c[j - 1, 0] = zero;
}

for (int j = 1; j <= n; j = j + 1)
{
    cc[j - 1] = j * ((j + 1) * j) / 2 - ((j + 1) * j * (j - 1)) / 3;
}
// CC holds the exact result. On exit from ZMMCH CT holds the result computed by ZMMCH
transa = "N";
transb = "N";
//ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, AB( 1, NMAX + 1 ), NMAX,
// ZERO, C, NMAX, CT, G,

```

```

// CC, NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH(transa, transb, n, 1, n, one, ab, nmax, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1,
ab)), nmax,
    zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0, nmax, cc), nmax, eps, ref err, ref fatal, fout1, true);

same = LZE(cc, ct, n);
if ((!same) || (err != rzero))
{
    fout1.WriteLine("ERROR IN ZMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("ZMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}", transa, transb);
    fout1.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}

transb = "C";
// ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, AB( 1, NMAX + 1 ), NMAX,
// ZERO, C, NMAX, CT, G,
// CC, NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH(transa, transb, n, 1, n, one, ab, nmax, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1,
ab)), nmax,
    zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0, nmax, cc), nmax, eps, ref err, ref fatal, fout1, true);

same = LZE(cc, ct, n);
if ((!same) || (err != rzero))
{
    fout1.WriteLine("ERROR IN ZMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("ZMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}", transa, transb);
    fout1.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}

for (int j = 1; j <= n; j = j + 1)
{
    ab[j - 1, (nmax + 1) - 1] = n - j + 1;
    ab[0, (nmax + j) - 1] = n - j + 1;
}
for (int j = 1; j <= n; j = j + 1)
{
    cc[(n - j + 1) - 1] = j * ((j + 1) * j) / 2 - ((j + 1) * j * (j - 1)) / 3;
}
transa = "C";
transb = "N";
// ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, AB( 1, NMAX + 1 ), NMAX,
// ZERO, C, NMAX, CT, G,
// CC, NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH(transa, transb, n, 1, n, one, ab, nmax, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1,
ab)), nmax,
    zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0, nmax, cc), nmax, eps, ref err, ref fatal, fout1, true);

same = LZE(cc, ct, n);
if ((!same) || (err != rzero))
{
    fout1.WriteLine("ERROR IN ZMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("ZMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}", transa, transb);
    fout1.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}
transb = "C";
// ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, AB( 1, NMAX + 1 ), NMAX,
// ZERO, C, NMAX, CT, G,
// CC, NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH(transa, transb, n, 1, n, one, ab, nmax, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1,
ab)), nmax,
    zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0, nmax, cc), nmax, eps, ref err, ref fatal, fout1, true);

same = LZE(cc, ct, n);
if ((!same) || (err != rzero))
{
    fout1.WriteLine("ERROR IN ZMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALUATED WRONGLY.");
    fout1.WriteLine("ZMMCH WAS CALLED WITH TRANSA = {0} AND TRANSB = {1}", transa, transb);
    fout1.WriteLine("AND RETURNED SAME = {0} AND ERR = {1,12:F3}", same, err);
    fout1.WriteLine("THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.");
    fout1.WriteLine("***** TESTS ABANDONED *****");
    Environment.Exit(1);
}

// Test each subroutine in turn
for (int isnum = 1; isnum <= nsubs; isnum = isnum + 1)
{
    //fout1.WriteLine("****SNAME={0} {1}", snames[isnum - 1], isnum);
    fout1.WriteLine();
    if (!test(isnum - 1))
    {
        // Subprogram is not to be tested
        fout1.WriteLine(" {0,6} WAS NOT TESTED", snames[isnum - 1]);
    }
    else
    {
        srnamt = snames[isnum - 1];
        // Test error exits
        if (tsterr)
        {
            ZBLAT3_ZCHK( isnum, snames[isnum - 1], fout1 );
            //fout1.WriteLine("end DCHKE in ZBLAT3");
            fout1.WriteLine();
        }
        //Console.WriteLine("H777");
        // Test computations
        infot = 0;
        ok = true;
        fatal = false;
        //fout1.Flush();
        //fout1.WriteLine("^^^isnum={0}", isnum);

        switch (isnum)
        {
            //switch (isnum+50)
            {

```

```

/////CHECK ARRAYS
// case 1-5 ab in as [nmax,nmax]
case 1:
    // Test ZGEMM, 01
    // ZCHK1( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM,
    // NALF, ALF, NBET, BET, NMAX,
    // AB,
    // AA, AS,
    // AB( 1, NMAX + 1 ), BB, BS, C, CC, CS, CT, G )
    ZBLAT3_ZCHK1(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim,
    nalf, alf, nbet, bet, nmax,
    zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d(nmax * 2 * nmax, 0, nmax, ab)),
    aa, _as,
    zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1, ab)), bb, bs, c, cc, cs, ct,
g);

    break;
case 2:
case 3:
    // Test ZHEMM, 02, ZSYMM, 03.
    // ZCHK2( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM,
    // NALF, ALF, NBET, BET, NMAX,
    // AB,
    // AA, AS,
    // AB( 1, NMAX + 1 ), BB, BS, C, CC, CS, CT, G )
    tmp2d_ab1 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d(nmax * 2 * nmax, 0, nmax, ab));
    tmp2d_ab2 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1, ab));
    ZBLAT3_ZCHK2(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim,
    nalf, alf, nbet, bet, nmax,
    ref tmp2d_ab1,
    ref aa, _as,
    ref tmp2d_ab2, ref bb, bs, ref c, ref cc, cs, ct, g); // in as b
    //may not be needed
    for (int i = 1; i <= nmax; i = i + 1)
    {
        for (int j = 1; j <= nmax; j = j + 1)
        {
            ab[i - 1, j - 1] = tmp2d_ab1[i - 1, j - 1];
            ab[i - 1, nmax + j - 1] = tmp2d_ab2[i - 1, j - 1];
        }
    }
    /* static void ZBLAT3_ZCHK2(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra,
    bool trace, bool rewi, ref bool fatal, int nidim, int[] idim,
    int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax,
    ref Complex[,] a, ref Complex[] aa, Complex[] _as,
    ref Complex[,] b, ref Complex[] bb, Complex[] bs, ref Complex[] c, ref Complex[] cc, Complex[] cs, Complex[] ct, double[] g)
    */

    break;
case 4:
case 5:
    // Test ZTRMM, 04, ZTRSM, 05.
    // ZCHK3( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM,
    // NALF, ALF, NMAX,
    // AB,
    // AA, AS,
    // AB( 1, NMAX + 1 ), BB, BS, CT, G, C )

    tmp2d_ab1 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d(nmax * 2 * nmax, 0, nmax, ab));
    tmp2d_ab2 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1, ab));
    ZBLAT3_ZCHK3(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim,
    nalf, alf, nmax,
    ref tmp2d_ab1,
    ref aa, ref _as,
    ref tmp2d_ab2, ref bb, ref bs, ct, g, ref c);
    /*static void ZBLAT3_ZCHK3(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool
    rewi, ref bool fatal, int nidim, int[] idim,
    int nalf, Complex[] alf, int nmax,
    * ref Complex[,] a,
    * ref Complex[] aa, ref Complex[] _as,
    * ref Complex[,] b, ref Complex[] bb, ref Complex[] bs, Complex[] ct, double[] g, ref Complex[,] c)
    */

    //may not be needed
    for (int i = 1; i <= nmax; i = i + 1)
    {
        for (int j = 1; j <= nmax; j = j + 1)
        {
            ab[i - 1, j - 1] = tmp2d_ab1[i - 1, j - 1];
            ab[i - 1, nmax + j - 1] = tmp2d_ab2[i - 1, j - 1];
        }
    }
    break;
case 6:
case 7:
    // Test ZHERK, 06, ZSYRK, 07.
    // ZCHK4( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM,
    // NALF, ALF, NBET, BET, NMAX,
    // AB,
    // AA, AS,
    // AB( 1, NMAX + 1 ), BB, BS, C, CC, CS, CT, G )
    tmp2d_ab1 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d(nmax * 2 * nmax, 0, nmax, ab));

    tmp2d_ab2 = zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, 2 * nmax, nmax + 1 - 1, ab));

    ZBLAT3_ZCHK4(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim,
    nalf, alf, nbet, bet, nmax,
    ref tmp2d_ab1,
    ref aa, ref _as,
    ref tmp2d_ab2, ref bb, ref bs, ref c, ref cc, ref cs, ct, g);
    /*static void ZBLAT3_ZCHK4(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool
    rewi, ref bool fatal, int nidim, int[] idim,
    int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax,
    * Complex[,] a, Complex[] aa, Complex[] _as, Complex[,] b, Complex[] bb, Complex[] bs, Complex[,] c, Complex[] cc, Complex[]
    cs, Complex[] ct, double[] g)
    */

    //may not be needed
    for (int i = 1; i <= nmax; i = i + 1)
    {
        for (int j = 1; j <= nmax; j = j + 1)
        {
            ab[i - 1, j - 1] = tmp2d_ab1[i - 1, j - 1];
            ab[i - 1, nmax + j - 1] = tmp2d_ab2[i - 1, j - 1];
        }
    }

```

```

        }
        break;
    case 8:
    case 9:
        // Test ZHER2K, 08, ZSYR2K, 09.
        // ZCHK5( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, REWI, FATAL, NIDIM, IDIM,
        // NALF, ALF, NBET, BET, NMAX,
        // AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )
        tmpld ab = zget_1dfrom2d_starti_full(nmax, 0, 2 * nmax, 0, ab);
        ZBLAT3_ZCHK5(snames[isnum - 1], eps, thresh, fout1, fout2, trace, rewi, ref fatal, nidim, idim,
            nalf, alf, nbet, bet, nmax,
            ref tmpld ab, ref aa, _as, ref bb, bs, ref c, ref cc, cs, ct, g, w);
        ab = zget_2dfrom1d(2 * nmax * nmax, 0, nmax, tmpld ab);
        //zget_2dfrom1d(int 1, int start, int lda, Complex[] arr

        /* static void ZBLAT3_ZCHK5(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool
        rewi, ref bool fatal, int nidim, int[] idim,
        int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax,
        * ref Complex[] ab, ref Complex[] aa, Complex[] _as, ref Complex[] bb, Complex[] bs, ref Complex[,] c, ref Complex[] cc,
        Complex[] cs, Complex[] ct, double[] g, double[] w
        */
        break;
    default:
        break;
    }
    fout1.Flush();
    if (fatal || sfatal)
    {
        fout1.WriteLine("\n***** FATAL ERROR - TESTS ABANDONED *****");
        if (trace)
        {
            fout1.Flush();
            fout1.Close();
        }
        fout1.Flush();
        fout1.Close();

        Environment.Exit(1);
    }
}
}
fout1.WriteLine("\nEND OF TESTS");
if (trace)
{
    fout2.Flush();
    fout2.Close();
}

Console.WriteLine("end.");
fout1.Flush();
fout1.Close();
}

static void ZBLAT3_ZCHK1(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi,
    ref bool fatal, int nidim, int[] idim, int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax,
    Complex[,] a, Complex[] aa, Complex[] _as, Complex[,] b, Complex[] bb, Complex[] bs, Complex[,] c, Complex[] cc, Complex[] cs, Complex[] ct,
double[] g)
{
    // Tests ZGEMM

    //SUBROUTINE ZCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
    // FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
    // A, AA, AS, B, BB, BS, C, CC, CS, CT, G )

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    const double rzero = 0.0;

    /*
    * .. Scalar Arguments ..
    DOUBLE PRECISION EPS, THRESH
    INTEGER NALF, NBET, NIDIM, NMAX, NOUT, NTRA
    LOGICAL FATAL, REWI, TRACE
    CHARACTER*6 SNAME
    * .. Array Arguments ..
    COMPLEX*16 A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
    $ AS( NMAX*NMAX ), B( NMAX, NMAX ),
    $ BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
    $ C( NMAX, NMAX ), CC( NMAX*NMAX ),
    $ CS( NMAX*NMAX ), CT( NMAX )
    DOUBLE PRECISION G( NMAX )
    INTEGER IDIM( NIDIM ) */

    // Local Scalars
    Complex alpha, als, beta, bls;
    double err, errmax;
    //int i, ia, ib, ica, icb, ik, im, _in
    int k, ks, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, m, ma, mb, ms, n, na, nargs, nb, nc, ns;
    bool _null, reset, same, trana, tranb;
    string tranas, tranbs, transa, transb, ich;

    // Local Arrays
    bool[] isame = new bool[13];
    //COMMON /INFOC/INFOT, NOUTC, OK, LERR
    Complex[,] tmp_bet, tmp2d_aa, tmp2d_bb, tmp2d_cc;
    err = 0.0; // added
    ich = "NTC";

    // Executable Statements
    nargs = 13;
    nc = 0;
    reset = true;
    errmax = rzero;
    //added
    //ma = 0;

    //110
    for (int im = 1; im <= nidim; im = im + 1)
    {
        m = idim[im - 1];
        //100
        for (int _in = 1; _in <= nidim; _in = _in + 1)
        {
            n = idim[_in - 1];
            // Set LDC to 1 more than minimum value if room

```

```

ldc = m;
if (ldc < nmax)
{
    ldc = ldc + 1;
}

// Skip tests if not enough room
if (ldc > nmax)
{
    goto Loop100;
}
lcc = ldc * n;
_null = (n <= 0) || (m <= 0);

//90
for (int ik = 1; ik <= nidim; ik = ik + 1)
{
    k = idim[ik - 1];

    //80
    for (int ica = 1; ica <= 3; ica = ica + 1)
    {
        transa = ich.Substring(ica - 1, 1).ToUpper();
        trana = (transa == "T") || (transa == "C");

        if (trana)
        {
            ma = k;
            na = m;
        }
        else
        {
            ma = m;
            na = k;
        }

        // Set LDA to 1 more than minimum value if room
        lda = ma;
        if (lda < nmax)
        {
            lda = lda + 1;
        }
        // Skip tests if not enough room
        if (lda > nmax)
        {
            //GO TO 80
            goto Loop80;
        }
        laa = lda * na;

        // Generate the matrix A
        // ZMAKE( 'GE', ' ', ' ', ' ', MA, NA, A, NMAX, AA, LDA, RESET, ZERO )
        ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref a, nmax, ref aa, lda, ref reset, zero);

        //70
        for (int icb = 1; icb <= 3; icb = icb + 1)
        {
            transb = ich.Substring(icb - 1, 1).ToUpper();
            tranb = (transb == "T") || (transb == "C");
            if (tranb)
            {
                mb = n;
                nb = k;
            }
            else
            {
                mb = k;
                nb = n;
            }

            // Set LDB to 1 more than minimum value if room
            ldb = mb;
            if (ldb < nmax)
            {
                ldb = ldb + 1;
            }

            // Skip tests if not enough room
            if (ldb > nmax)
            {
                //GO TO 70
                goto Loop70;
            }
            lbb = ldb * nb;

            // Generate the matrix B
            //ZMAKE( 'GE', ' ', ' ', ' ', MB, NB, B, NMAX, BB, LDB, RESET, ZERO )
            //tmp_bet = get_2dfrom1d(nbet, 0, nmax, bet);
            ZBLAT3_ZMAKE("GE", " ", " ", mb, nb, ref b, nmax, ref bb, ldb, ref reset, zero);
            //bet = get_ldfrom2d(nbet, 0, nmax, tmp_bet);
            //static void DBLAT3_DMAKE(string type, string uplo, string diag, int m, int n, ref double[,] a, int nmax, ref double[] aa,
            int lda, ref bool reset, double trans1)
            //DBLAT3_DMAKE("GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);

            //60
            for (int ia = 1; ia <= nalf; ia = ia + 1)
            {
                alpha = alf[ia - 1];
                //50
                for (int ib = 1; ib <= nbet; ib = ib + 1)
                {
                    beta = bet[ib - 1];
                    // Generate the matrix C
                    //ZMAKE( 'GE', ' ', ' ', ' ', M, N, C, NMAX, CC, LDC, RESET, ZERO )
                    ZBLAT3_ZMAKE("GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);

                    nc = nc + 1;

                    // Save every datum before calling the subroutine
                    tranas = transa;
                    tranbs = transb;
                    ms = m;
                    ns = n;

```

```

ks = k;
als = alpha;
for (int i = 1; i <= laa; i = i + 1)
{
    _as[i - 1] = aa[i - 1];
}
ldas = lda;
for (int i = 1; i <= lbb; i = i + 1)
{
    bs[i - 1] = bb[i - 1];
}
ldbs = ldb;
bls = beta;
for (int i = 1; i <= lcc; i = i + 1)
{
    cs[i - 1] = cc[i - 1];
}
ldcs = ldc;

// Call the subroutine
if (trace)
{
    //NC, SNAME, TRANSA, TRANSB, M, N, K, ALPHA, LDA, LDB, BETA, LDC
    ntra.WriteLine(" {0,6:D}: {1,-6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7}, A {8,3:D}, B {9,3:D}, {10}, C,
(11,3:D)).", nc, sname, transa, transb, m, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    //9995 FORMAT( ' ', A6, '(', ' ', A1, ' ', ' ', ' ', A1, ' ', ' ', ' ', 3( ' I3, ' ', ' ), '(', F4.1, ' ', ' ', F4.1, ' '), A, ' ', I3, ' ',
B, ' ', I3, &

    //',(' ', F4.1, ' ', ' ', F4.1, ' '), C, ' ', I3, ' ').' )
}
if (rewi)
{
    //REWIND NTRA
}

//ZGEMM( TRANSA, TRANSB, M, N, K, ALPHA,AA, LDA, BB, LDB, BETA, CC, LDC )

//nout.WriteLine("{0} {1} {2} {3} {4} {5} {6}", im, n, ik, ica, icb, ia, ib);
// nout.WriteLine("{0} {1} {2} {3} {4}", transa, transb, m, n, k);
tmp2d_cc = zget_2dfromld(nmax * nmax, 0, ldc, cc);
ZGEMM(transa, transb, m, n, k, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, zget_2dfromld(nmax * nmax, 0,
ldb, bb), ldb, beta, ref tmp2d_cc, ldc);
cc = zget_ldfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7}, A {8,3:D}, B {9,3:D}, {10}, C,
(11,3:D)).",
        nc, sname, transa, transb, m, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    // 9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
    //9995 FORMAT( 1X, I6, ': ', A6, '(', ' ', A1, ' ', ' ', ' ', A1, ' ', ' ', ' ', 3( ' I3, ' ', ' ), '(', F4.1, ' ', ' ', F4.1, ' '), A, ' ',
I3, ' ', B, ' ', I3, &

    //',(' ', F4.1, ' ', ' ', F4.1, ' '), C, ' ', I3, ' ').' )
    nout.Flush();
    return;
}

// See what data changed inside subroutines
isame[0] = (transa == tranas);
isame[1] = (transb == tranbs);
isame[2] = (ms == m);
isame[3] = (ns == n);
isame[4] = (ks == k);
isame[5] = (als == alpha);
isame[6] = LZE(_as, aa, laa);
isame[7] = (ldas == lda);
isame[8] = LZE(bs, bb, lbb);
isame[9] = (ldbs == ldb);
isame[10] = (bls == beta);
if (_null)
{
    isame[11] = LZE(cs, cc, lcc);
}
else
{
    isame[11] = LZERES("GE", " ", m, n, zget_2dfromld(nmax * nmax, 0, ldc, cs), zget_2dfromld(nmax * nmax, 0, ldc,
cc), ldc);
}
isame[12] = (ldcs == ldc);
// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 120
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7}, A {8,3:D}, B {9,3:D}, {10}, C,
(11,3:D)).",
        nc, sname, transa, transb, m, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    nout.Flush();
    return;
}
if (!_null)
{
    // Check the result
    //ZMMCH( TRANSA, TRANSB, M, N, K,ALPHA, A, NMAX, B, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR, FATAL, NOUT,
.TRUE. )
    ZMMCH(transa, transb, m, n, k, alpha, a, nmax, b, nmax, beta, c, nmax, ref ct, ref g, zget_2dfromld(nmax *
nmax, 0, ldc, cc), ldc, eps, ref err, ref fatal, nout, true);
}

```

```

        errmax = Math.Max(errmax, err);

        // If got really bad answer, report and return
        if (fatal)
        {
            //GO TO 120
            nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,3:D}, {7}, A {8,3:D}, B {9,3:D}, {10},
C, {11,3:D}).", nc, sname, transa, transb, m, n, k,
            alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
            nout.Flush();
            return;
        }
    }
} //50
} //60
Loop70: ;
} //70
Loop80: ;
} //80
} //90
Loop100: ;
} //100
} //110

// Report result
if (errmax < thresh)
{
    nout.WriteLine("{0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine("{0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}
static void ZBLAT3_ZCHK2(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax,
    ref Complex[,] a, ref Complex[] aa, Complex[] _as,
    ref Complex[,] b, ref Complex[] bb, Complex[] bs, ref Complex[,] c, ref Complex[] cc, Complex[] cs, Complex[] ct, double[] g)
{
    // Tests ZHEMM and ZSYMM

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    //Complex half = new Complex(0.5,0.0);
    const double rzero = 0.0;

    // Scalar Arguments
    /*double eps, thresh;
    int nalf, nbet, nidim, nmax; // nout, ntra;
    StringWriter nout, ntra;
    bool fatal, rewi, trace;
    string sname;

    // Array Arguments
    Complex[,] a = new Complex[nmax,nmax];
    Complex[] aa = new Complex[nmax*nmax];
    Complex[] alf = new Complex[nalf];
    Complex[] _as = new Complex[nmax*nmax];
    Complex[,] b = new Complex[nmax,nmax];
    complex[] bb = new Complex[nmax*nmax];
    Complex[] bet = new Complex[nbet];
    Complex[] bs = new Complex[nmax*nmax];

    Complex[,] c = new Complex[nmax,nmax];
    Complex[] cc = new Complex[nmax*nmax];
    Complex[] cs = new Complex[nmax*nmax];
    Complex[] ct = new Complex[nmax];

    double[] g = new double[nmax];
    int[] idim = new int[nidim];
    //int[] inc = new int[ninc];
    //int[] kb = new int[nkb];*/

    // Local Scalars
    Complex alpha, als, beta, bls;
    double err, errmax;
    //int i, ia, ib, ics, icu, im, _in, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, m, ms, n, na, nargs, nc, ns;
    int laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, nargs, nc, ns, m, ms, n, na;
    bool conj, left, _null, reset, same;
    string side, sides, uplo, uplos, ichs, ichu;

    // Local Arrays
    bool[] isame = new bool[13];

    /* External Functions
    LOGICAL      LZE, LZERES
    EXTERNAL     LZE, LZERES */

    Complex[,] tmp2d_x;
    Complex[,] tmp2d_y;
    Complex[,] tmp2d_cc;
    */
    /*
    .. Scalars in Common ..
    INTEGER      INFOT, NOUTC
    LOGICAL      LERR, OK
    .. Common blocks ..
    COMMON       /INFOC/INFOT, NOUTC, OK, LERR */

    err = 0.0; // added
    ichs = "LR";
    ichu = "UL";

    // Executable Statements
    conj = (sname.Substring(1, 2).ToUpper() == "HE");

    nargs = 12;

```

```

nc = 0;
reset = true;
errmax = rzero;

//100
for (int im = 1; im <= nidim; im = im + 1)
{
    m = idim[im - 1];

    //90
    for (int _in = 1; _in <= nidim; _in = _in + 1)
    {
        n = idim[_in - 1];

        // Set LDA to 1 more than minimum value if room
        ldc = m;

        if (ldc < nmax)
        {
            ldc = ldc + 1;
        }
        // Skip tests if not enough room
        if (ldc > nmax)
        {
            //GO TO 90
            continue;
        }
        lcc = ldc * n;
        _null = ((n <= 0) || (m <= 0));

        // Set LDB to 1 more than minimum value if room.
        ldb = m;
        if (ldb < nmax)
        {
            ldb = ldb + 1;
        }
        // Skip tests if not enough room.
        if (ldb > nmax)
        {
            //GO TO 90
            continue;
        }
        lbb = ldb * n;

        // Generate the matrix B.
        // ZMAKE( 'GE', ' ', ' ', ' ', M, N, B, NMAX, BB, LDB, RESET, ZERO )
        ZBLAT3_ZMAKE("GE", " ", " ", m, n, ref b, nmax, ref bb, ldb, ref reset, zero);

    //80
    for (int ics = 1; ics <= 2; ics = ics + 1)
    {
        side = ichs.Substring(ics - 1, 1).ToUpper();
        left = (side == "L");

        if (left)
        {
            na = m;
        }
        else
        {
            na = n;
        }

        // Set LDA to 1 more than minimum value if room.
        lda = na;
        if (lda < nmax)
        {
            lda = lda + 1;
        }
        // Skip tests if not enough room.
        if (lda > nmax)
        {
            //GO TO 80
            continue;
        }
        laa = lda * nalf;

    //70
    for (int icu = 1; icu <= 2; icu = icu + 1)
    {
        uplo = ichu.Substring(icu - 1, 1).ToUpper();
        // Generate the hermitian or symmetric matrix A.
        // ZMAKE( SNAME( 2: 3 ), UPLO, ' ', NA, NA, A, NMAX, AA, LDA, RESET, ZERO )
        ZBLAT3_ZMAKE(sname.Substring(1, 2), uplo, " ", na, na, ref a, nmax, ref aa, lda, ref reset, zero);

    //60
    for (int ia = 1; ia <= nalf; ia = ia + 1)
    {
        alpha = alf[ia - 1];
        //50
        for (int ib = 1; ib <= nbet; ib = ib + 1)
        {
            beta = bet[ib - 1];
            // Generate the matrix C.
            // ZMAKE( 'GE', ' ', ' ', ' ', M, N, C, NMAX, CC, LDC, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", m, n, ref c, nmax, ref cc, ldc, ref reset, zero);
            nc = nc + 1;
            // Save every datum before calling the subroutine.
            sides = side;
            uplos = uplo;
            ms = m;
            ns = n;
            als = alpha;
            for (int i = 1; i <= laa; i = i + 1)
            {
                _as[i - 1] = aa[i - 1];
            }
            ldas = lda;
            for (int i = 1; i <= lbb; i = i + 1)
            {
                bs[i - 1] = bb[i - 1];
            }
            ldfs = ldb;
            bls = beta;

```

```

for (int i = 1; i <= ldc; i = i + 1)
{
    cs[i - 1] = cc[i - 1];
}
ldcs = ldc;

// Call the subroutine
if (trace)
{
    //ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, {8,4:F1}, C {9,3:D}).",
    ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A {7,3:D}, B {8,3:D}, {9}, C {10,3:D}).",
        nc, sname, side, uplo, m, n,
        alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
}

if (rewi)
{
    //REWIND NTRA
}

if (conj)
{
    tmp2d_cc = zget_2dfromld(nmax * nmax, 0, ldc, cc);
    //DSYMM(side, uplo, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb,
    beta, ref tmp2d_cc, ldc);

    //CALL ZHEMM( SIDE, UPLO, M, N, ALPHA, AA, LDA, BB, LDB, BETA, CC, LDC )
    ZHEMM(side, uplo, m, n, alpha, zget_2dfromld(nmax*nmax,0,lda,aa), lda,
        zget_2dfromld(nmax*nmax,0,ldb,bb), ldb, beta, ref tmp2d_cc, ldc);
    cc = zget_1dfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);
    //ZHEMM(sTring side, string uplo, int m, int n, Complex alpha, Complex[,] a, int lda,
    //Complex[,] b, int ldb, Complex beta, ref Complex[,] c, int ldc
    //

    //
}
else
{
    tmp2d_cc = zget_2dfromld(nmax * nmax, 0, ldc, cc);
    //DSYMM(side, uplo, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb,
    beta, ref tmp2d_cc, ldc);

    //CALL ZSYMM( SIDE, UPLO, M, N, ALPHA, AA, LDA, BB, LDB, BETA, CC, LDC )
    ZSYMM(side, uplo, m, n, alpha, zget_2dfromld(nmax*nmax,0,lda,aa), lda,
        zget_2dfromld(nmax*nmax,0,ldb,bb), ldb, beta, ref tmp2d_cc, ldc);
    cc = zget_1dfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);
}

// Check if error-exit was taken incorrectly
if (!ok)
{
    nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
    fatal = true;
    //GO TO 110
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A {7,3:D}, B {8,3:D}, {9}, C {10,3:D}).",
        nc, sname, side, uplo, m, n, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
}

// See what data changed inside subroutines
isame[0] = (sides == side);
isame[1] = (uplos == uplo);
isame[2] = (ms == m);
isame[3] = (ns == n);
isame[4] = (als == alpha);
isame[5] = LZE(as, aa, laa);
isame[6] = (ldas == lda);
isame[7] = LZE(bs, bb, lbb);
isame[8] = (ldbs == ldb);
isame[9] = (bls == beta);

if (_null)
{
    isame[10] = LZE(cs, cc, ldc);
}
else
{
    isame[10] = LZERES("GE", " ", m, n, zget_2dfromld(nmax * nmax, 0, ldc, cs), zget_2dfromld(nmax * nmax, 0, ldc, cc),
    ldc);
}
isame[11] = (ldcs == ldc);

// If data was incorrectly changed, report and return.
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 110
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
    {10,3:D}).",
        //nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A {7,3:D}, {8}, C {9,3:D}).",
        nc, sname, side, uplo, m, n, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
}
if (!_null)
{
    // Check the result
    if (left)
    {
        //CALL DMMCH( 'N', 'N', M, N, M, ALPHA, A,NMAX, B, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR, FATAL, NOUT,
        .TRUE. )
        //DMMCH("N", "N", m, n, m, alpha, a, nmax, b, nmax, beta, c, nmax, ref ct, ref g,
        get_2dfromld(nmax*nmax,0,ldc,cc), ldc, eps, ref fatal, nout, true);
        //ZMMCH( 'N', 'N', M, N, M, ALPHA, A,NMAX, B, NMAX, BETA, C, NMAX,CT, G, CC, LDC, EPS, ERR, FATAL, NOUT, .TRUE.
        )
    }
}

```

```

        ZMMCH("N", "N", m, n, m, alpha, a, nmax, b, nmax, beta, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0,
ldc, cc), ldc, eps, ref err, ref fatal, nout, true);
    }
    else
    {
        // DMMCH( 'N', 'N', M, N, N, ALPHA, B, NMAX, A, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR, FATAL, NOUT,
.TRUE. )
        //DMMCH("N", "N", m, n, n, alpha, b, nmax, a, nmax, beta, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldc,cc), ldc, eps, ref err, ref fatal, nout, true);
        // ZMMCH( 'N', 'N', M, N, N, ALPHA, B, NMAX, A, NMAX, BETA, C, NMAX, CT, G, CC, LDC, EPS, ERR,FATAL, NOUT,
.TRUE. )
        ZMMCH("N", "N", m, n, n, alpha, b, nmax, a, nmax, beta, c, nmax, ref ct, ref g, zget_2dfromld(nmax * nmax, 0,
ldc, cc), ldc, eps, ref err, ref fatal, nout, true);
    }
    errmax = Math.Max(errmax, err);
    // If got really bad answer, report and return
    if (fatal)
    {
        //GO TO 110
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A {7,3:D}, B {8,3:D}, {9,4:F1}, C
(10,3:D)).",
        //nout.WriteLine(" {0,6:D}: {1,6} {(2,1), {3,1}, {4,3:D}, {5,3:D}, {6}, A {7,3:D}, {8,4:F1}, C {9,3:D}).",
        nc, sname, side, uplo, m, n, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    }
}
} //50
} //60
} //70
Loop80: ;
} //80
Loop90: ;
} //90
} //100

// Report result
if (errmax < thresh)
{
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}

static void ZBLAT3_ZCHK3(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
int nalf, Complex[] alf, int nmax, ref Complex[,] a, ref Complex[] aa, ref Complex[] _as, ref Complex[,] b, ref Complex[] bb, ref Complex[] bs,
Complex[] ct, double[] g, ref Complex[,] c)
{
    // Tests ZTRMM and ZTRSM
    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex one = new Complex(1.0, 0.0);
    const double rzero = 0;
    /*
    COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
    $              AS( NMAX*NMAX ), B( NMAX, NMAX ),
    $              BB( NMAX*NMAX ), BS( NMAX*NMAX ),
    $              C( NMAX, NMAX ), CT( NMAX )
    DOUBLE PRECISION G( NMAX )*/

    // Local Scalars
    Complex alpha, als;
    double err, errmax;
    //int i,ia,icd,ics,ict,icu,im, in,j,laa,lb,lda,ldas,ldb,ldbs,m,ms,n,na,nargs,nc,ns;
    int laa, lb, lda, ldas, ldb, ldbs, m, ms, n, na, nargs, nc, ns;
    bool left, _null, reset, same;
    string diag, diags, side, sides, tranas, transa, uplo, uplos, ichd, ichs, ichu, icht;

    // Local Arrays
    bool[] isame = new bool[13];

    // COMMON          /INFOC/INFOT, NOUTC, OK, LERR

    ichu = "UL";
    icht = "NTC";
    ichd = "UN";
    ichs = "LR";

    Complex[,] tmp2d_bb;
    //=====
    /*double[,] t2 = {{1.5,3.5,5.5,7.5},{2.5,4.5,6.5,8.5}};
    double[,] t3 = get_2dfromld(2*2,0,2,get_ldfrom2d_starti(2,0,4,3-1,t2));
    for (int i = 1; i <= 2; i = i + 1)
    {
        for (int j = 1; j <= 2; j = j + 1)
        {
            Console.WriteLine("{0} {1} = {2}",i,j,t3[i-1,j-1]);
        }
    }
    */
    //=====

    // Executable Statements
    nargs = 11;
    nc = 0;
    reset = true;
    errmax = rzero;
    err = 0; //added

    // Set up zero matrix for ZMMCH.
    for (int j = 1; j <= nmax; j = j + 1)
    {
        for (int i = 1; i <= nmax; i = i + 1)
        {
            c[i - 1, j - 1] = zero;
        }
    }
}

```

```

//140
for (int im = 1; im <= nidim; im = im + 1)
{
    m = idim[im - 1];
    //130
    for (int _in = 1; _in <= nidim; _in = _in + 1)
    {
        n = idim[_in - 1];
        // Set LDB to 1 more than minimum value if room
        ldb = m;
        if (ldb < nmax)
        {
            ldb = ldb + 1;
        }
        // Skip tests if not enough room
        if (ldb > nmax)
        {
            //GO TO 130
            goto Loop130;
        }
        lbb = ldb * n;
        _null = (m <= 0) || (n <= 0);
        //120
        //ichs="LR"
        for (int ics = 1; ics <= 2; ics = ics + 1)
        {
            side = ichs.Substring(ics - 1, 1).ToUpper();
            left = (side == "L");
            if (left)
            {
                na = m;
            }
            else
            {
                na = n;
            }
            // Set LDA to 1 more than minimum value if room
            lda = na;
            if (lda < nmax)
            {
                lda = lda + 1;
            }
            // Skip tests if not enough room
            if (lda > nmax)
            {
                //GO TO 130
                goto Loop130;
            }
            laa = lda * na;
            //110
            //ichu="UL"
            for (int icu = 1; icu <= 2; icu = icu + 1)
            {
                uplo = ichu.Substring(icu - 1, 1).ToUpper();
                //100
                //icht = "NTC";
                for (int ict = 1; ict <= 3; ict = ict + 1)
                {
                    transa = icht.Substring(ict - 1, 1).ToUpper();
                    //90
                    //ichd = "UN";
                    for (int icd = 1; icd <= 2; icd = icd + 1)
                    {
                        diag = ichd.Substring(icd - 1, 1).ToUpper();
                        //80
                        for (int ia = 1; ia <= nalf; ia = ia + 1)
                        {
                            alpha = alf[ia - 1];
                            // Generate the matrix A.
                            // DBLAT3 DMAKE("TR", uplo, diag, na, na, ref a, nmax, ref aa, lda, ref reset, zero);
                            // DBLAT3 DMAKE("GE", " ", " ", m, n, ref b, nmax, ref bb, ldb, ref reset, zero);
                            //ZMAKE( 'TR', UPLO, DIAG, NA, NA, A,NMAX, AA, LDA, RESET, ZERO )
                            ZBLAT3_ZMAKE("TR", uplo, diag, na, na, ref a, nmax, ref aa, lda, ref reset, zero);

                            // Generate the matrix B.
                            //ZMAKE( 'GE', ' ', ' ', M, N, B, NMAX,BB, LDB, RESET, ZERO )
                            ZBLAT3_ZMAKE("GE", " ", " ", m, n, ref b, nmax, ref bb, ldb, ref reset, zero);

                            nc = nc + 1;

                            // Save every datum before calling the subroutine
                            sides = side;
                            uplos = uplo;
                            transa = transa;
                            diags = diag;
                            ms = m;
                            ns = n;
                            als = alpha;
                            for (int i = 1; i <= laa; i = i + 1)
                            {
                                _as[i - 1] = aa[i - 1];
                            }
                            ldas = lda;
                            for (int i = 1; i <= lbb; i = i + 1)
                            {
                                bs[i - 1] = bb[i - 1];
                            }
                            ldbs = ldb;

                            // Call the subroutine
                            if (sname.Substring(3, 2).ToUpper() == "MM")
                            {
                                if (trace)
                                {
                                    ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8}, A {9,3:D}, B {10,3:D})",
                                        nc, sname, side, uplo, transa, diag, m, n, alpha.ToString("F1"), lda, ldb);
                                }
                                if (rewi)
                                {
                                    //REWIND NTRA
                                }
                                tmp2d_bb = zget_2dfromld(nmax * nmax, 0, ldb, bb);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        //DTRMM(side,uplo, transa, diag, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
        //ZTRMM( SIDE, UPLO, TRANSA, DIAG, M,N, ALPHA, AA, LDA, BB, LDB )
        ZTRMM(side, uplo, transa, diag, m, n, alpha, zget_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
        //ZTRMM(string side, string uplo, string transa, string diag, int m, int n, Complex alpha, Complex[,] a, int
lda, ref Complex[,] b, int ldb

        bb = zget_ldfrom2d(nmax * nmax, 0, ldb, tmp2d_bb);
    }
    else if (sname.Substring(3, 2).ToUpper() == "SM")
    {
        if (trace)
        {
            ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8}, A {9,3:D}, B {10,3:D})
            nc, sname, side, uplo, transa, diag, m, n, alpha.ToString("F1"), lda, ldb);
        }
        if (rewi)
        {
            //REWIND NTRA
        }

        tmp2d_bb = zget_2dfromld(nmax * nmax, 0, ldb, bb);
        //DTRSM(side,uplo, transa, diag, m, n, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
        //ZTRSM( SIDE, UPLO, TRANSA, DIAG, M,N, ALPHA, AA, LDA, BB, LDB )
        ZTRSM(side, uplo, transa, diag, m, n, alpha, zget_2dfromld(nmax*nmax,0,lda,aa), lda, ref tmp2d_bb, ldb);
        bb = zget_ldfrom2d(nmax * nmax, 0, ldb, tmp2d_bb);
    }

    // Check if error-exit was taken incorrectly.
    if (!ok)
    {
        nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");
        fatal = true;
        //GO TO 150
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8}, A {9,3:D}, B {10,3:D})
        nc, sname, side, uplo, transa, diag, m, n, alpha.ToString("F1"), lda, ldb);
    }

    // See what data changed inside subroutines
    isame[0] = (sides == side);
    isame[1] = (uplos == uplo);
    isame[2] = (transa == transa);
    isame[3] = (diags == diag);
    isame[4] = (ms == m);
    isame[5] = (ns == n);
    isame[6] = (als == alpha);
    isame[7] = LZE(as, aa, laa);
    isame[8] = (ldas == lda);

    if (_null)
    {
        isame[9] = LZE(bs, bb, lbb);
    }
    else
    {
        isame[9] = LZERES("GE", " ", m, n, zget_2dfromld(nmax * nmax, 0, ldb, bs), zget_2dfromld(nmax * nmax, 0, ldb,
bb), ldb);
    }
    isame[10] = (ldbs == ldb);

    // If data was incorrectly changed, report and return
    same = true;
    for (int i = 1; i <= nargs; i = i + 1)
    {
        same = (same && isame[i - 1]);
        if (!isame[i - 1])
        {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }

    if (!same)
    {
        fatal = true;
        //GO TO 150
        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,1}, {5,1}, {6,3:D}, {7,3:D}, {8}, A {9,3:D}, B {10,3:D})
        nc, sname, side, uplo, transa, diag, m, n, alpha.ToString("F1"), lda, ldb);
    }

    if (!_null)
    {
        if (sname.Substring(3, 2).ToUpper() == "MM")
        {
            // Check the result
            if (left)
            {
                //DMMCH(transa, "N", m, n, m, alpha, a, nmax, b, nmax, zero, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout,true);
                //ZMMCH( TRANSA, 'N', M, N, M,ALPHA, A, NMAX, B, NMAX,ZERO, C, NMAX, CT, G,BB, LDB, EPS, ERR, FATAL,
NOUT, .TRUE. )
                ZMMCH(transa, "N", m, n, m, alpha, a, nmax, b, nmax, zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax *
nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
            }
            else
            {
                //DMMCH("N", transa, m, n, n, alpha, b, nmax, a, nmax, zero, c, nmax, ref ct, ref g,
get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout, true);
                //ZMMCH( 'N', TRANSA, M, N, N,ALPHA, B, NMAX, A, NMAX,ZERO, C, NMAX, CT, G,BB, LDB, EPS, ERR,FATAL,
NOUT, .TRUE. )
                ZMMCH("N", transa, m, n, n, alpha, b, nmax, a, nmax, zero, c, nmax, ref ct, ref g, zget_2dfromld(nmax *
nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
            }
        }
        else if (sname.Substring(3, 2).ToUpper() == "SM")
        {
            // Compute approximation to original matrix.
            for (int j = 1; j <= n; j = j + 1)

```

```

        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = bb[(i + (j - 1) * ldb) - 1];
                bb[(i + (j - 1) * ldb) - 1] = alpha * b[i - 1, j - 1];
            }
        }
        if (left)
        {
            #region comment

            #endregion

            get_2dfromld(nmax * nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
            //DMMCH(transa, "N", m, n, m, alpha, a, nmax, b, nmax, zero, c, nmax, ref ct, ref g,
            //DMMCH(transa, "N", m, n, m, one, a, nmax, c, nmax, zero, b, nmax, ref ct, ref g,
            get_2dfromld(nmax*nmax,0,ldb,bb),ldb, eps, ref err, ref fatal, nout, false);
            //ZMMCH( TRANSA, 'N', M, N, M,ONE, A, NMAX, C, NMAX,ZERO, B, NMAX, CT, G,BB, LDB, EPS, ERR,FATAL, NOUT,
            .FALSE. )
            ZMMCH(transa, "N", m, n, m, one, a, nmax, c, nmax, zero, b, nmax, ref ct, ref g, zget_2dfromld(nmax *
            nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, false);

        }
        else
        {
            //nout.WriteLine("Right SM ");
            //DMMCH( 'N', TRANSA, M, N, N, ONE, C, NMAX, A, NMAX, ZERO, B, NMAX, CT, G, BB, LDB, EPS, ERR, FATAL,
            NOUT, .FALSE. )
            //DMMCH("N", transa, m, n, n, alpha, b, nmax, a, nmax, zero, c, nmax, ref ct, ref g,
            get_2dfromld(nmax * nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, true);
            //DMMCH("N", transa, m, n, n, one, c, nmax, a, nmax, zero, b, nmax, ref ct, ref g,
            get_2dfromld(nmax*nmax,0,ldb,bb), ldb, eps, ref err, ref fatal, nout, false);
            //ZMMCH( 'N', TRANSA, M, N, N,ONE, C, NMAX, A, NMAX,ZERO, B, NMAX, CT, G,BB, LDB, EPS, ERR,FATAL, NOUT,
            .FALSE. )
            ZMMCH("N", transa, m, n, n, one, c, nmax, a, nmax, zero, b, nmax, ref ct, ref g, zget_2dfromld(nmax *
            nmax, 0, ldb, bb), ldb, eps, ref err, ref fatal, nout, false);
        }
        }
        errmax = Math.Max(errmax, err);
        // If got really bad answer, report and return
        if (fatal)
        {
            // GO TO 150
            // Report result
            if (errmax < thresh)
            {
                nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
            }
            else
            {
                nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST
            RATIO (2,8:F2) - SUSPECT *****", sname, nc, errmax);
            }
            return;
        }
    }
}
} //80 ia
} //90
} //100
} //110
} //120
Loop130: ;
} //130
} //140
// Report result
if (errmax < thresh)
{
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST
    RATIO (2,8:F2) - SUSPECT *****", sname, nc, errmax);
}
}
static void ZBLAT3_ZCHK4(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
    int half, Complex[] alf, int nbet, Complex[] bet, int nmax, ref Complex[,] a, ref Complex[,] aa, ref Complex[,] _as, ref Complex[,] b, ref
Complex[,] bb, ref Complex[,] bs, ref Complex[,] c, ref Complex[,] cc, ref Complex[,] cs, Complex[,] ct, double[,] g)
{
    // Tests ZHERK and ZSYRK.

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    const double rone = 1.0;
    const double rzero = 0.0;

    /*
    .. Array Arguments ..
    COMPLEX*16  A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),
    $           AS( NMAX*NMAX ), B( NMAX, NMAX ),
    $           BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),
    $           C( NMAX, NMAX ), CC( NMAX*NMAX ),
    $           CS( NMAX*NMAX ), CT( NMAX ),
    $           DOUBLE PRECISION G( NMAX )
    */

    // Local Scalars
    Complex alpha, als, beta, bets;
    double err, errmax, ralpha, rals, rbeta, rbets;
    //int i, ia, ib, ict, icu, ik, _in, j, jc, jj, k, ks, laa, lcc, lda, ldas, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
    int k, _in, jc, jj, ks, laa, lcC, lda, ldas, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
    bool conJ, _null, reset, same, tran, upper;
    string trans, transs, transt, uplo, uplos, ichu, icht;

    bool[] isame = new bool[13];
    Complex[,] tmp2d_cc;
    //COMMON /INFOC/INFOT, NOUTC, OK, LERR

    icht = "NC";
    ichu = "UL";

    // Executable Statements

```

```

conj = (sname.Substring(1, 2) == "HE");

nargs = 10;
nc = 0;
reset = true;
errmax = rzero;

err = 0; //added
ralpha = 0; //added
rbeta = 0; //added
rals = 0; //added
als = new Complex(0, 0); //added
rbets = 0; //added
bets = new Complex(0, 0); //added

//100
for (_in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    // Set LDC to 1 more than minimum value if room
    ldc = n;
    if (ldc < nmax)
    {
        ldc = ldc + 1;
    }
    // Skip tests if not enough room
    if (ldc > nmax)
    {
        goto Loop100;
    }
    lcc = ldc * n;
    // _null = (n <= 0);
    //90
    for (int ik = 1; ik <= nidim; ik = ik + 1)
    {
        k = idim[ik - 1];
        //80
        for (int ict = 1; ict <= 2; ict = ict + 1)
        {
            trans = icht.Substring(ict - 1, 1).ToUpper();
            tran = (trans == "C");
            if (tran && !conj)
            {
                trans = "T";
            }
            if (tran)
            {
                ma = k;
                na = n;
            }
            else
            {
                ma = n;
                na = k;
            }

            // Set LDA to 1 more than minimum value if room
            lda = ma;
            if (lda < nmax)
            {
                lda = lda + 1;
            }
            // Skip tests if not enough room
            if (lda > nmax)
            {
                //GO TO 80
                goto Loop80;
            }
            laa = lda * na;

            // Generate the matrix A
            //DMAKE( 'GE', ' ', ' ', ' ', MA, NA, A, NMAX, AA, LDA, RESET, ZERO )
            //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref a, nmax, ref aa, lda, ref reset, zero);
            // ZMAKE( 'GE', ' ', ' ', ' ', MA, NA, A, NMAX, AA, LDA, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref a, nmax, ref aa, lda, ref reset, zero);
            //70
            for (int icu = 1; icu <= 2; icu = icu + 1)
            {
                uplo = ichu.Substring(icu - 1, 1).ToUpper();
                upper = (uplo == "U");
                //60
                for (int ia = 1; ia <= nalf; ia = ia + 1)
                {
                    alpha = alf[ia - 1];
                    if (conj)
                    {
                        ralpha = alpha.Real;
                        alpha = new Complex(ralpha, rzero);
                    }
                    //50
                    for (int ib = 1; ib <= nbet; ib = ib + 1)
                    {
                        beta = bet[ib - 1];
                        if (conj)
                        {
                            rbeta = beta.Real;
                            beta = new Complex(rbeta, rzero);
                        }
                        _null = (n <= 0);
                        if (conj)
                        {
                            //NULL = NULL.OR.( ( K.LE.0.OR.RALPHA.EQ.RZERO ).AND.RBETA.EQ.RONE )
                            _null = (_null || ((k <= 0) || (ralpha == rzero) && (rbeta == rone)));
                        }

                        // Generate the matrix C
                        //DMAKE( 'SY', UPLO, ' ', N, N, C, NMAX, CC, LDC, RESET, ZERO )
                        //DBLAT3_DMAKE("SY", uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);
                        //ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, C, NMAX, CC, LDC, RESET, ZERO )
                        ZBLAT3_ZMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);

                        nc = nc + 1;
                    }
                }
            }
        }
    }
}

```



```

    }
    isame[5] = LZE(_as, aa, laa);
    isame[6] = (ldas == lda);
    if (conj)
    {
        isame[7] = (rbets == rbeta);
    }
    else
    {
        isame[7] = (bets == beta);
    }
    if (_null)
    {
        isame[8] = LZE(cs, cc, lcc);
    }
    else
    {
        isame[8] = LZERES(sname.Substring(1, 2), uplo, n, n, zget_2dfromld(nmax * nmax, 0, ldc, cs), zget_2dfromld(nmax *
nmax, 0, ldc, cc), ldc);

    }
    isame[9] = (ldcs == ldc);

    // If data was incorrectly changed, report and return
    same = true;
    for (int i = 1; i <= nargs; i = i + 1)
    {
        same = (same && isame[i - 1]);
        if (!isame[i - 1])
        {
            nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
        }
    }
    if (!same)
    {
        fatal = true;
        //GO TO 120

        nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
        if (conj)
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C, {9,3:D})
.",
            nc, sname, uplo, trans, n, k, ralpha, lda, rbeta, ldc);
        }
        else
        {
            nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, {8}, C, {9,3:D})
.",
            nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, beta.ToString("F1"), ldc);
        }
    }

    }
    if (!_null)
    {
        // Check the result column by column
        if (conj)
        {
            transt = "C";
        }
        else
        {
            transt = "T";
        }
        jc = 1;
        //40
        for (int j = 1; j <= n; j = j + 1)
        {
            if (upper)
            {
                jj = 1;
                lj = j;
            }
            else
            {
                jj = j;
                lj = n - j + 1;
            }
            if (tran)
            {
                /*DMMCH("T", "N", lj, 1, k, alpha, get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, 0,
nmax,jj-1, a)),nmax,
                get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, 0,nmax, j-1, a)),nmax, beta,
                get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1, nmax,j-1, c)),nmax,
                ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1,nmax*nmax-1, cc)), ldc, eps,
ref err, ref fatal, nout, true);

                */
                //ZMMCH( TRANST, 'N', LJ, 1, K, ALPHA, A( 1, JJ ), NMAX,
                //A( 1, J ), NMAX, BETA,
                //C( JJ, J ), NMAX,
                //CT, G, CC( JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
                ZMMCH(transt, "N", lj, 1, k, alpha, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0,
nmax, jj - 1, a)), nmax,
                zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, 0, nmax, j - 1, a)), nmax, beta,
                zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, jj - 1, nmax, j - 1, c)), nmax,
                ref ct, ref g, zget_2dfromld(nmax * nmax, 0, ldc, get_zarr_full(nmax * nmax, jc - 1, nmax * nmax - 1,
cc)), ldc, eps, ref err, ref fatal, nout, true);

            }
            else
            {
                /*DMMCH("N", "T", lj, 1, k, alpha, get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1,
nmax,0, a)), nmax,
                get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, j-1, nmax,0,a)),nmax, beta,
                get_2dfromld(nmax*nmax,0,nmax, get_ldfrom2d_starti_full(nmax, jj-1, nmax,j-1, c)),nmax,
                ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1,nmax*nmax-1, cc)), ldc, eps,
ref err, ref fatal, nout, true);

                */
            }
        }
    }

```

```

//ZMMCH( 'N', TRANST, LJ, 1, K, ALPHA, A( JJ, 1 ), NMAX,
// A( J, 1 ), NMAX, BETA,
// C( JJ, J ), NMAX,
// CT, G, CC( JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH("N", transt, lj, 1, k, alpha, zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, jj
- 1, nmax, 0, a)), nmax,
zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, j - 1, nmax, 0, a)), nmax, beta,
zget_2dfromld(nmax * nmax, 0, nmax, zget_ldfrom2d_starti_full(nmax, jj - 1, nmax, j - 1, c)), nmax,
ref ct, ref g, zget_2dfromld(nmax * nmax, 0, ldc, get_zarr_full(nmax * nmax, jc - 1, nmax * nmax - 1,
cc)), ldc, eps, ref err, ref fatal, nout, true);
}
if (upper)
{
jc = jc + ldc;
}
else
{
jc = jc + ldc + 1;
}
errmax = Math.Max(errmax, err);

// If got really bad answer, report and return
if (fatal)
{
//GO TO 110
if (n > 1)
{
nout.WriteLine(" THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
}
nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
if (conj)
{
nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6,4:F1}, A, {7,3:D}, {8,4:F1}, C,
(9,3:D)) .",
nc, sname, uplo, trans, n, k, ralpha, lda, rbeta, ldc);
}
else
{
nout.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, {8}, C, {9,3:D})
.",
nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, beta.ToString("F1"), ldc);
}
}
}
} //40
} //50
} //60
} //70
Loop80: ;
} //80
} //90
Loop100: ;
} //100

// Report result
if (errmax < thresh)
{
nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}

static void ZBLAT3_ZCHK5(string sname, double eps, double thresh, StreamWriter nout, StreamWriter ntra, bool trace, bool rewi, ref bool fatal, int
nidim, int[] idim,
int nalf, Complex[] alf, int nbet, Complex[] bet, int nmax, ref Complex[] ab, ref Complex[] aa, Complex[] _as, ref Complex[] bb, Complex[] bs,
ref Complex[,] c, ref Complex[] cc, Complex[] ct, double[] g, Complex[] w)
{
// Tests ZHER2K and ZSYR2K.

// Parameters
Complex zero = new Complex(0.0, 0.0);
Complex one = new Complex(1.0, 0.0);

const double rone = 1.0;
const double rzero = 0.0;
/*
Complex AA( NMAX*NMAX ), AB( 2*NMAX*NMAX ),
$ ALF( NALF ), AS( NMAX*NMAX ), BB( NMAX*NMAX ),
$ BET( NBET ), BS( NMAX*NMAX ), C( NMAX, NMAX ),
$ CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ),
$ W( 2*NMAX )
double G( NMAX )*/

// Local Scalars
Complex alpha, als, beta, bets;
double err, errmax, rbeta, rbets;
//int i, ia, ib, ict, icu, ik, _in, j, jc, jj, jjab, k, ks, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
int jc, jj, jjab, k, ks, laa, lbb, lcc, lda, ldas, ldb, ldbs, ldc, ldcs, lj, ma, n, na, nargs, nc, ns;
bool conj, _null, reset, same, tran, upper;
string trans, transs, transt, uplo, uplos, icht, ichu;

// Local Arrays
bool[] isame = new bool[13];
//COMMON /INFOC/INFOT, NOUTC, OK, LERR
Complex[,] tmp_ab;
Complex[] tmp_ab_ld;

icht = "NC";
ichu = "UL";
Complex[,] tmp2d_cc;

// Executable Statements
conj = (sname.Substring(1, 2) == "HE");

```

```

nargs = 12;
nc = 0;
reset = true;
errmax = rzero;
err = 0; //added
bets = new Complex(0, 0); //added
rbeta = 0; //added
rbets = 0; //added

//l30
for (int _in = 1; _in <= nidim; _in = _in + 1)
{
    n = idim[_in - 1];
    // Set LDC to 1 more than minimum value if room
    ldc = n;
    if (ldc < nmax)
    {
        ldc = ldc + 1;
    }
    // Skip tests if not enough room
    if (ldc > nmax)
    {
        //GO TO l30
        goto Loopl30;
    }
    lcc = ldc * n;
    // _null = (n <= 0);

//l20
for (int ik = 1; ik <= nidim; ik = ik + 1)
{
    k = idim[ik - 1];
    //l10
    for (int ict = 1; ict <= 2; ict = ict + 1)
    {
        trans = icht.Substring(ict - 1, 1).ToUpper();
        tran = (trans == "C");
        if (tran && !conj)
        {
            trans = "T";
        }
        if (tran)
        {
            ma = k;
            na = n;
        }
        else
        {
            ma = n;
            na = k;
        }
        // Set LDA to 1 more than minimum value if room
        lda = ma;
        if (lda < nmax)
        {
            lda = lda + 1;
        }
        // Skip tests if not enough room
        if (lda > nmax)
        {
            //GO TO l10
            goto Loopl10;
        }
        laa = lda * na;

        // Generate the matrix A
        if (tran)
        {
            tmp_ab = zget_2dfromld(2 * nmax * nmax, 0, 2 * nmax, ab);
            //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2*nmax, ref aa, lda, ref reset, zero);
            // ZMAKE( 'GE', ' ', ' ', MA, NA, AB, 2*NMAX, AA, LDA, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2 * nmax, ref aa, lda, ref reset, zero);
            ab = zget_ldfrom2d(2 * nmax * nmax, 0, 2 * nmax, tmp_ab);
        }
        else
        {
            tmp_ab = zget_2dfromld(2 * nmax * nmax, 0, nmax, ab);
            //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref aa, lda, ref reset, zero);
            // ZMAKE( 'GE', ' ', ' ', MA, NA, AB, NMAX, AA, LDA, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref aa, lda, ref reset, zero);
            ab = zget_ldfrom2d(2 * nmax * nmax, 0, nmax, tmp_ab);
            //tatic void ZBLAT3_ZMAKE(string type, string uplo, string diag, int m, int n, ref Complex[,] a, int nmax, ref Complex[] aa,
            int lda, ref bool reset, Complex transl

        }

        //

        // Generate the matrix B
        ldb = lda;
        lbb = laa;
        if (tran)
        {
            tmp_ab = zget_2dfromld(2 * nmax * nmax, 0, 2 * nmax, get_zarr_full(2 * nmax * nmax, k + 1 - 1, 2 * nmax * nmax - 1, ab));
            //DBLAT3_DMAKE( "GE", " ", " ", ma, na, ref tmp_ab , 2*nmax, ref bb, ldb, ref reset, zero);
            //ZMAKE( 'GE', ' ', ' ', MA, NA, AB( K + 1 ),2*NMAX, BB, LDB, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref tmp_ab, 2 * nmax, ref bb, ldb, ref reset, zero);
            tmp_ab_ld = zget_ldfrom2d(2 * nmax * nmax, 0, 2 * nmax, tmp_ab);
            zcopy_ld(tmp_ab_ld, 0, 2 * nmax * nmax - (k + 1) + 1, ab, k + 1 - 1);
        }
        else
        {
            tmp_ab = zget_2dfromld(2 * nmax * nmax, 0, nmax, get_zarr_full(2 * nmax * nmax, k * nmax + 1 - 1, 2 * nmax * nmax - 1, ab));
            //DBLAT3_DMAKE("GE", " ", " ", ma, na, ref tmp_ab , nmax, ref bb,ldb, ref reset, zero);
            //ZMAKE( 'GE', ' ', ' ', MA, NA, AB( K*NMAX + 1 ),NMAX, BB, LDB, RESET, ZERO )
            ZBLAT3_ZMAKE("GE", " ", " ", ma, na, ref tmp_ab, nmax, ref bb, ldb, ref reset, zero);
            tmp_ab_ld = zget_ldfrom2d(2 * nmax * nmax, 0, nmax, tmp_ab);
            zcopy_ld(tmp_ab_ld, 0, 2 * nmax * nmax - (k * nmax + 1) + 1, ab, k * nmax + 1 - 1);
        }

        //l00
        for (int icu = 1; icu <= 2; icu = icu + 1)
        {

```

```

uplo = ichu.Substring(ichu - 1, 1).ToUpper();
upper = (uplo == "U");

//90
for (int ia = 1; ia <= nalf; ia = ia + 1)
{
    alpha = alf[ia - 1];
    //80
    for (int ib = 1; ib <= nbet; ib = ib + 1)
    {
        beta = bet[ib - 1];

        if (conj)
        {
            rbeta = beta.Real;
            beta = new Complex(rbeta, rzero);
        }
        _null = (n <= 0);
        if (conj)
        {
            //NULL = NULL.OR.( ( K.LE.0.OR.ALPHA.EQ.ZERO ).AND.RBETA.EQ.RONE )
            _null = (_null || ((k <= 0) || (alpha == zero)) && (rbeta == rone));
        }

        // Generate the matrix C
        //DELAT3_DMAKE("SY", uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);
        //ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, C,NMAX, CC, LDC, RESET, ZERO )
        ZBLAT3_ZMAKE(sname.Substring(1, 2), uplo, " ", n, n, ref c, nmax, ref cc, ldc, ref reset, zero);

        nc = nc + 1;

        // Save every datum before calling the subroutine
        uplos = uplo;
        transs = trans;
        ns = n;
        ks = k;
        als = alpha;
        for (int i = 1; i <= laa; i = i + 1)
        {
            _as[i - 1] = aa[i - 1];
        }
        ldas = lda;
        for (int i = 1; i <= lbb; i = i + 1)
        {
            bs[i - 1] = bb[i - 1];
        }
        ldbs = ldb;
        if (conj)
        {
            rbets = rbeta;
        }
        else
        {
            bets = beta;
        }
        for (int i = 1; i <= lcc; i = i + 1)
        {
            cs[i - 1] = cc[i - 1];
        }
        ldcs = ldc;

        // Call the subroutine
        if (conj)
        {
            if (trace)
            {
                ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C,
(10,3:D))      .",
                                nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, rbeta, ldc);
            }
            if (rewi)
            {
                //REWIND NTRA
            }

            tmp2d_cc = zget_2dfromld(nmax * nmax, 0, ldc, cc);
            //DSYR2K(uplo, trans, n, k, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb,
beta, ref tmp2d_cc, ldc);
            //ZHER2K( UPLO, TRANS, N, K, ALPHA, AA,LDA, BB, LDB, RBETA, CC, LDC )
            ZHER2K(uplo, trans, n, k, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, zget_2dfromld(nmax * nmax, 0, ldb,
bb), ldb, rbeta, ref tmp2d_cc, ldc);

            cc = zget_ldfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);
        }
        else
        {
            if (trace)
            {
                ntra.WriteLine(" {0,6:D}: {1,6} ({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9}, C, (10,3:D))
.",
                                nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
            }
            if (rewi)
            {
                //REWIND NTRA
            }

            tmp2d_cc = zget_2dfromld(nmax * nmax, 0, ldc, cc);
            //DSYR2K(uplo, trans, n, k, alpha, get_2dfromld(nmax*nmax,0,lda,aa), lda, get_2dfromld(nmax*nmax,0,ldb,bb), ldb,
beta, ref tmp2d_cc, ldc);
            //ZSYR2K( UPLO, TRANS, N, K, ALPHA, AA,LDA, BB, LDB, BETA, CC, LDC )
            ZSYR2K(uplo, trans, n, k, alpha, zget_2dfromld(nmax * nmax, 0, lda, aa), lda, zget_2dfromld(nmax * nmax, 0, ldb,
bb), ldb, beta, ref tmp2d_cc, ldc);

            cc = zget_ldfrom2d(nmax * nmax, 0, ldc, tmp2d_cc);
        }

        // Check if error-exit was taken incorrectly
        if (!ok)
        {
            nout.WriteLine("***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *****");

            fatal = true;
        }
    }
}

```

```

//GO TO 150
nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
if (conj)
{
    nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C
{10,3:D}) .",
        nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, rbeta, ldc);
}
else
{
    nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9}, C {10,3:D})
.",
        nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
}
return;
}

// See what data changed inside subroutines
isame[0] = (uplos == uplo);
isame[1] = (trans == trans);
isame[2] = (ns == n);
isame[3] = (ks == k);
isame[4] = (als == alpha);
isame[5] = LZE(_as, aa, laa);
isame[6] = (ldas == lda);
isame[7] = LZE(bs, bb, lbb);
isame[8] = (ldbs == ldb);
if (conj)
{
    isame[9] = (rbets == rbeta);
}
else
{
    isame[9] = (bets == beta);
}

if (_null)
{
    isame[10] = LZE(cs, cc, lcc);
}
else
{
    isame[10] = LZERES("HE", uplo, n, n, zget_2dfromld(nmax * nmax, 0, ldc, cs), zget_2dfromld(nmax * nmax, 0, ldc,
cc), ldc);
}

isame[11] = (ldcs == ldc);

// If data was incorrectly changed, report and return
same = true;
for (int i = 1; i <= nargs; i = i + 1)
{
    same = (same && isame[i - 1]);
    if (!isame[i - 1])
    {
        nout.WriteLine("***** FATAL ERROR - PARAMETER NUMBER {0,2:D} WAS CHANGED INCORRECTLY *****", i);
    }
}
if (!same)
{
    fatal = true;
    //GO TO 150
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (conj)
    {
        nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9,4:F1}, C
{10,3:D}) .",
            nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, rbeta, ldc);
    }
    else
    {
        nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9}, C {10,3:D})
.",
            nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    }
    return;
}

if (!_null)
{
    // Check the result column by column
    if (conj)
    {
        transt = "C";
    }
    else
    {
        transt = "T";
    }

    jjab = 1;
    jc = 1;
    //70
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            jj = 1;
            lj = j;
        }
        else
        {
            jj = j;
            lj = n - j + 1;
        }
        if (tran)
        {
            for (int i = 1; i <= k; i = i + 1)
            {
                w[i - 1] = alpha * ab[(jj - 1) * 2 * nmax + k + i] - 1];
                if (conj)
                {
                    w[k + i - 1] = Complex.Conjugate(alpha) * ab[(jj - 1) * 2 * nmax + i] - 1];
                }
            }
        }
    }
}

```

```

    }
    else
    {
        w[k + i - 1] = alpha * ab[((j - 1) * 2 * nmax + i) - 1];
    }
}
/*DMMCH("T", "N", lj, 1, 2*k, alpha, get_2dfromld( 2*nmax*nmax,jjab-1, 2*nmax, ab), 2*nmax,
get_2dfromld(2*nmax,0,2*nmax,w), 2*nmax, beta,
get_2dfromld(nmax, 0, ldc, get_ldfrom2d_starti_full(nmax,jj-1,nmax,j-1, c)), nmax,
ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1, nmax * nmax - 1, cc)), ldc,
eps, ref err, ref fatal, nout, true);*/

//ZMMCH( TRANST, 'N', LJ, 1, 2*K, ONE, AB( JJAB ), 2*NMAX,
//W, 2*NMAX, BETA,
// C( JJ, J ), NMAX,
// CT, G, CC( JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH(transt, "N", lj, 1, 2 * k, one, zget_2dfromld(2 * nmax * nmax, jjab - 1, 2 * nmax, ab), 2 * nmax,
zget_2dfromld(2 * nmax, 0, 2 * nmax, w), 2 * nmax, beta,
zget_2dfromld(nmax, 0, ldc, zget_ldfrom2d_starti_full(nmax, jj - 1, nmax, j - 1, c)), nmax,
ref ct, ref g, zget_2dfromld(nmax * nmax, 0, ldc, get_zarr_full(nmax * nmax, jc - 1, nmax * nmax - 1,
cc)), ldc, eps, ref err, ref fatal, nout, true);

}
else
{
    for (int i = 1; i <= k; i = i + 1)
    {
        if (conj)
        {
            w[i - 1] = alpha * Complex.Conjugate(ab[((k + i - 1) * nmax + j) - 1]);
            w[k + i - 1] = Complex.Conjugate(alpha * ab[((i - 1) * nmax + j) - 1]);
        }
        else
        {
            w[i - 1] = alpha * ab[((k + i - 1) * nmax + j) - 1];
            w[(k + i) - 1] = alpha * ab[((i - 1) * nmax + j) - 1];
        }
    }
    /*DMMCH("N", "N", lj, 1, 2*k, alpha, get_2dfromld(2*nmax*nmax,jj-1, nmax, ab), nmax,
get_2dfromld(2*nmax,0,2*nmax,w), 2*nmax, beta,
get_2dfromld(nmax, 0, ldc, get_ldfrom2d_starti_full(nmax, jj-1, nmax,j-1, c)), nmax,
ref ct, ref g, get_2dfromld(nmax*nmax,0,ldc,get_darr_full(nmax*nmax,jc-1,nmax*nmax-1,cc)),ldc, eps, ref
err, ref fatal, nout, true);*/

//ZMMCH( 'N', 'N', LJ, 1, 2*K, ONE, AB( JJ ), NMAX,
// W, 2*NMAX, BETA,
// C( JJ, J ), NMAX,
// CT, G, CC( JC ), LDC, EPS, ERR, FATAL, NOUT, .TRUE. )
ZMMCH("N", "N", lj, 1, 2 * k, one, zget_2dfromld(2 * nmax * nmax, jj - 1, nmax, ab), nmax,
zget_2dfromld(2 * nmax, 0, 2 * nmax, w), 2 * nmax, beta,
zget_2dfromld(nmax, 0, ldc, zget_ldfrom2d_starti_full(nmax, jj - 1, nmax, j - 1, c)), nmax,
ref ct, ref g, zget_2dfromld(nmax * nmax, 0, ldc, get_zarr_full(nmax * nmax, jc - 1, nmax * nmax - 1,
cc)), ldc, eps, ref err, ref fatal, nout, true);

}
if (upper)
{
    jc = jc + ldc;
}
else
{
    jc = jc + ldc + 1;
    if (tran)
    {
        jjab = jjab + 2 * nmax;
    }
}
errmax = Math.Max(errmax, err);

// If got really bad answer, report and return
if (fatal)
{
    //GO TO 140
    if (n > 1)
    {
        nout.WriteLine("      THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
    }
    nout.WriteLine("***** {0,6} FAILED ON CALL NUMBER:", sname);
    if (conj)
    {
        nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9,4:F1},
C {10,3:D})      .",
nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, rbeta, ldc);
    }
    else
    {
        nout.WriteLine(" {0,6:D}: {1,6}({2,1}, {3,1}, {4,3:D}, {5,3:D}, {6}, A, {7,3:D}, B, {8,3:D}, {9}, C
{10,3:D})      .",
nc, sname, uplo, trans, n, k, alpha.ToString("F1"), lda, ldb, beta.ToString("F1"), ldc);
    }
    return;
}
} //70
} //80
} //90
} //100
Loop110: ;
} //110
} //120
Loop130: ;
} //130

// Report result
if (errmax < thresh)
{
    nout.WriteLine(" {0,6} PASSED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)", sname, nc);
}
else
{
    nout.WriteLine(" {0,6} COMPLETED THE COMPUTATIONAL TESTS ({1,6:D} CALLS)\n***** BUT WITH MAXIMUM TEST RATIO {2,8:F2} - SUSPECT *****",
sname, nc, errmax);
}
}
static void

```

```

ZBLAT3_ZCHKKE(int isnum, string srnamt, StreamWriter nout)
{
    // Tests the error exits from the Level 3 Blas. Requires a special version of the error-handling routine XERBLA.
    // A, B and C should not need to be defined.

    // Parameters
    const double one = 1.0;
    const double two = 2.0;

    // Local Scalars
    Complex alpha, beta;
    double ralpha, rbeta;

    // Local Arrays
    Complex[,] a = new Complex[2, 1];
    Complex[,] b = new Complex[2, 1];
    Complex[,] c = new Complex[2, 1];

    // COMMON          /INFOC/INFOT, NOUTC, OK, LERR

    double[] tmp1d_a;

    alpha = 0.0; //added
    beta = 0.0; // added

    // Executable Statements
    // OK is set to .FALSE. by the special version of XERBLA or by CHKXER if anything is wrong.
    ok = true;

    // LERR is set to .TRUE. by the special version of XERBLA each time it is called, and is then tested and re-set by CHKXER
    lerr = false;

    // Initialize ALPHA, BETA, RALPHA, and RBETA.
    alpha = new Complex(one, -one);
    beta = new Complex(two, -two);
    ralpha = one;
    rbeta = 2;

    switch (isnum)
    {
        case 1:
            infot = 1;
            /* static void ZGEMM(string transa, string transb, int m, int n, int k, Complex alpha, Complex[,] a, int lda,
               Complex[,] b, int ldb, Complex beta, ref Complex[,] c, int ldc)
            */
            ZGEMM("/", "N", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 1;
            ZGEMM("/", "C", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 1;
            ZGEMM("/", "T", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 2;
            ZGEMM("N", "/", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 2;
            ZGEMM("C", "/", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 2;
            ZGEMM("T", "/", 0, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("N", "N", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("N", "C", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("N", "T", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("C", "N", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("C", "C", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("C", "T", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("T", "N", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("T", "C", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 3;
            ZGEMM("T", "T", -1, 0, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("N", "N", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("N", "C", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("N", "T", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("C", "N", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("C", "C", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("C", "T", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("T", "N", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;
            ZGEMM("T", "C", 0, -1, 0, alpha, a, 1, b, 1, beta, ref c, 1);
            misc.d.CHKXER(srnamt, infot, nout, lerr, ok);
            infot = 4;

```



```

        ZSYR2K("U", "T", 0, 2, alpha, a, 2, b, 1, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        ZSYR2K("L", "N", 2, 0, alpha, a, 2, b, 1, beta, ref c, 2);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 9;
        ZSYR2K("L", "T", 0, 2, alpha, a, 2, b, 1, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        ZSYR2K("U", "N", 2, 0, alpha, a, 2, b, 2, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        ZSYR2K("U", "T", 2, 0, alpha, a, 1, b, 1, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        ZSYR2K("L", "N", 2, 0, alpha, a, 2, b, 2, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);
        infot = 12;
        ZSYR2K("L", "T", 2, 0, alpha, a, 1, b, 1, beta, ref c, 1);
        misc_d.CHKXER(srnamt, infot, nout, lerr, ok);

        break;
    default:
        break;
}
//100
if (ok)
{
    nout.WriteLine(" {0,6} PASSED THE TESTS OF ERROR-EXITS", srnamt);
}
else
{
    nout.WriteLine("***** {0,6} FAILED THE TESTS OF ERROR-EXITS *****", srnamt);
}
return;
}
static void ZBLAT3_ZMAKE(string type, string uplo, string diag, int m, int n, ref Complex[,] a, int nmax, ref Complex[] aa, int lda, ref bool
reset, Complex transl)
{
    // Generates values for an M by N matrix A. Stores the values in the array AA in the data structure required
    // by the routine, with unwanted elements set to rogue value.
    // TYPE is 'GE', 'HE', 'SY' or 'TR'.

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex one = new Complex(1.0, 0.0);
    Complex rogue = new Complex(-1.0e10, 1.0e10);
    const double rzero = 0.0;
    const double rrogue = -1.0e10;

    // Complex A( NMAX, * ), AA( * )

    // Local Scalars
    //int i, ibeg, iend, j, jj
    int ibeg, iend, jj;
    bool gen, her, lower, sym, tri, unit, upper;

    // Executable Statements
    gen = (type.Substring(0, 2).ToUpper() == "GE");
    her = (type.Substring(0, 2).ToUpper() == "HE");
    sym = (type.Substring(0, 2).ToUpper() == "SY");
    tri = (type.Substring(0, 2).ToUpper() == "TR");
    upper = ((her || sym || tri) && (uplo.Substring(0, 1).ToUpper() == "U"));
    lower = ((her || sym || tri) && (uplo.Substring(0, 1).ToUpper() == "L"));
    unit = tri && (diag.Substring(0, 1).ToUpper() == "U");

    // Generate data in array A

    //20
    for (int j = 1; j <= n; j = j + 1)
    {
        //10
        for (int i = 1; i <= m; i = i + 1)
        {
            if (gen || (upper && (i <= j)) || (lower && (i >= j)))
            {
                a[i - 1, j - 1] = ZBEG(ref reset) + transl;
                if (i != j)
                {
                    // Set some elements to zero
                    if ((n > 3) && (j == n / 2))
                    {
                        a[i - 1, j - 1] = zero;
                    }
                    if (her)
                    {
                        a[j - 1, i - 1] = Complex.Conjugate(a[i - 1, j - 1]);
                    }
                    else if (sym)
                    {
                        a[j - 1, i - 1] = a[i - 1, j - 1];
                    }
                    else if (tri)
                    {
                        a[j - 1, i - 1] = zero;
                    }
                }
            }
        }
        //10
        if (her)
        {
            a[j - 1, j - 1] = new Complex(a[j - 1, j - 1].Real, rzero);
        }
        if (tri)
        {
            a[j - 1, j - 1] = a[j - 1, j - 1] + one;
        }
        if (unit)
        {
            a[j - 1, j - 1] = one;
        }
    }
    //20
    // Store elements in array AS in data structure required by routine
    if (type.Substring(0, 2).ToUpper() == "GE")

```

```

{
    //50
    //Console.WriteLine("3_DMAKE GE");
    for (int j = 1; j <= n; j = j + 1)
    {
        //30
        for (int i = 1; i <= m; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
        }
        for (int i = m + 1; i <= lda; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
    } //50
}
else if ((type.Substring(0, 2).ToUpper() == "HE") || (type.Substring(0, 2).ToUpper() == "SY") || (type.Substring(0, 2).ToUpper() == "TR"))
{
    //90
    for (int j = 1; j <= n; j = j + 1)
    {
        if (upper)
        {
            {
                ibeg = 1;
                if (unit)
                {
                    iend = j - 1;
                }
                else
                {
                    iend = j;
                }
            }
        }
        else
        {
            {
                if (unit)
                {
                    ibeg = j + 1;
                }
                else
                {
                    ibeg = j;
                }
                iend = n;
            }
        }
        for (int i = 1; i <= ibeg - 1; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
        for (int i = ibeg; i <= iend; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = a[i - 1, j - 1];
        }
        for (int i = iend + 1; i <= lda; i = i + 1)
        {
            aa[(i + (j - 1) * lda) - 1] = rogue;
        }
        if (her)
        {
            {
                jj = j + (j - 1) * lda;
                aa[jj - 1] = new Complex(aa[jj - 1].Real, rrogue);
            }
        }
    } //90
}
}
static void ZMMCH(string transa, string transb, int m, int n, int kk, Complex alpha, Complex[,] a, int lda, Complex[,] b, int ldb, Complex beta,
Complex[,] c, int ldc,
    ref Complex[] ct, ref double[] g, Complex[,] cc, int ldcc, double eps, ref double err, ref bool fatal, StreamWriter nout, bool mv)
{
    // Checks the results of the computational tests.

    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    //Complex one = new Complex(1.0,0.0);
    const double rzero = 0.0;
    const double rone = 1.0;

    // Local Scalars
    Complex cl;
    double erri;
    bool ctrana, ctranb, trana, tranb;

    // COMPLEX*16 A( LDA, * ), B( LDB, * ), C( LDC, * ), CC( LDCC, * ), CT( * ),
    // DOUBLE PRECISION G( * )

    // Executable Statements
    trana = (transa.Substring(0, 1).ToUpper() == "T") || (transa.Substring(0, 1).ToUpper() == "C");
    tranb = (transb.Substring(0, 1).ToUpper() == "T") || (transb.Substring(0, 1).ToUpper() == "C");
    ctrana = (transa.Substring(0, 1).ToUpper() == "C");
    ctranb = (transb.Substring(0, 1).ToUpper() == "C");

    // Compute expected result, one column at a time, in CT using data in A, B and C. Compute gauges in G.
    //220
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            {
                ct[i - 1] = zero;
                g[i - 1] = rzero;
            }
            if (!trana && !tranb)
            {
                for (int k = 1; k <= kk; k = k + 1)
                {
                    for (int i = 1; i <= m; i = i + 1)
                    {
                        {
                            ct[i - 1] = ct[i - 1] + a[i - 1, k - 1] * b[k - 1, j - 1];
                            g[i - 1] = g[i - 1] + ABS1(a[i - 1, k - 1]) * ABS1(b[k - 1, j - 1]);
                        }
                    }
                }
            }
            else if (trana && !tranb)
            {
                {
                    if (ctrana)

```

```

    {
        for (int k = 1; k <= kk; k = k + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                ct[i - 1] = ct[i - 1] + Complex.Conjugate(a[k - 1, i - 1]) * b[k - 1, j - 1];
                g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[k - 1, j - 1]);
            }
        }
    }
    else
    {
        for (int k = 1; k <= kk; k = k + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                ct[i - 1] = ct[i - 1] + a[k - 1, i - 1] * b[k - 1, j - 1];
                g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[k - 1, j - 1]);
            }
        }
    }
}
else if (!trana && tranb)
{
    if (ctranb)
    {
        for (int k = 1; k <= kk; k = k + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                ct[i - 1] = ct[i - 1] + a[i - 1, k - 1] * Complex.Conjugate(b[j - 1, k - 1]);
                g[i - 1] = g[i - 1] + ABS1(a[i - 1, k - 1]) * ABS1(b[j - 1, k - 1]);
            }
        }
    }
    else
    {
        for (int k = 1; k <= kk; k = k + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                ct[i - 1] = ct[i - 1] + a[i - 1, k - 1] * b[j - 1, k - 1];
                g[i - 1] = g[i - 1] + ABS1(a[i - 1, k - 1]) * ABS1(b[j - 1, k - 1]);
            }
        }
    }
}
else if (trana && tranb)
{
    if (ctrana)
    {
        if (ctranb)
        {
            for (int k = 1; k <= kk; k = k + 1)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    ct[i - 1] = ct[i - 1] + Complex.Conjugate(a[k - 1, i - 1]) * Complex.Conjugate(b[j - 1, k - 1]);
                    g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[j - 1, k - 1]);
                }
            }
        }
        else
        {
            for (int k = 1; k <= kk; k = k + 1)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    ct[i - 1] = ct[i - 1] + Complex.Conjugate(a[k - 1, i - 1]) * b[j - 1, k - 1];
                    g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[j - 1, k - 1]);
                }
            }
        }
    }
    else
    {
        if (ctranb)
        {
            for (int k = 1; k <= kk; k = k + 1)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    ct[i - 1] = ct[i - 1] + a[k - 1, i - 1] * Complex.Conjugate(b[j - 1, k - 1]);
                    g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[j - 1, k - 1]);
                }
            }
        }
        else
        {
            for (int k = 1; k <= kk; k = k + 1)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    ct[i - 1] = ct[i - 1] + a[k - 1, i - 1] * b[j - 1, k - 1];
                    g[i - 1] = g[i - 1] + ABS1(a[k - 1, i - 1]) * ABS1(b[j - 1, k - 1]);
                }
            }
        }
    }
}
}
for (int i = 1; i <= m; i = i + 1)
{
    ct[i - 1] = alpha * ct[i - 1] + beta * c[i - 1, j - 1];
    g[i - 1] = ABS1(alpha) * g[i - 1] + ABS1(beta) * ABS1(c[i - 1, j - 1]);
}

// Compute the error ratio for this result
err = rzero; //corrected from zero
//210

```

```

        for (int i = 1; i <= m; i = i + 1)
        {
            erri = ABS1(ct[i - 1] - cc[i - 1, j - 1]) / eps;
            if (g[i - 1] != rzero)
            {
                erri = erri / g[i - 1];
            }
            err = Math.Max(err, erri);
            if (err * Math.Sqrt(eps) >= rone)
            {
                //GO TO 230
                // Report fatal error
                fatal = true;
                nout.WriteLine("***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HALF ACCURATE *****\n          EXPECTED RESULT    COMPUTED
RESULT");
                for (int ii = 1; ii <= m; ii = ii + 1)
                {
                    if (mv)
                    {
                        nout.WriteLine(" {0,7:D} {1,18} {2,18}", ii, ct[ii - 1].ToString("F6"), cc[ii - 1, j - 1].ToString("F6"));
                    }
                    else
                    {
                        nout.WriteLine(" {0,7:D} {1,18} {2,18}", ii, cc[ii - 1, j - 1].ToString("F6"), ct[ii - 1].ToString("F6"));
                    }
                }
                if (n > 1)
                {
                    nout.WriteLine("          THESE ARE THE RESULTS FOR COLUMN {0,3:D}", j);
                }
                return;
            }
        }
    } //210
} //220

// If the loop completes, all results are at least half accurate
//GO TO 250
}
public static void ZGEMM(string transa, string transb, int m, int n, int k, Complex alpha, Complex[,] a, int lda, Complex[,] b, int ldb, Complex
beta, ref Complex[,] c, int ldc)
{
    // ZGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
    // COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)

    // ZGEMM performs one of the matrix-matrix operations
    // C := alpha*op( A )*op( B ) + beta*C,
    // where op( X ) is one of
    // op( X ) = X or op( X ) = X**T or op( X ) = X**H,
    // alpha and beta are scalars, and A, B and C are matrices, with op( A ) an m by k matrix, op( B ) a k by n matrix and C an m by n matrix.

    // TRANSA is CHARACTER*1
    // On entry, TRANSA specifies the form of op( A ) to be used in the matrix multiplication as follows:
    // TRANSA = 'N' or 'n', op( A ) = A.
    // TRANSA = 'T' or 't', op( A ) = A**T.
    // TRANSA = 'C' or 'c', op( A ) = A**H.

    // TRANSB is CHARACTER*1
    // On entry, TRANSB specifies the form of op( B ) to be used in the matrix multiplication as follows:
    // TRANSB = 'N' or 'n', op( B ) = B.
    // TRANSB = 'T' or 't', op( B ) = B**T.
    // TRANSB = 'C' or 'c', op( B ) = B**H.

    // M is INTEGER
    // On entry, M specifies the number of rows of the matrix op( A ) and of the matrix C. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of the matrix op( B ) and the number of columns of the matrix C. N must be at least zero.

    // K is INTEGER
    // On entry, K specifies the number of columns of the matrix op( A ) and the number of rows of the matrix op( B ). K must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA =
    // 'N' or 'n', the leading m by k
    // part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at
    // least max( 1, m ), otherwise LDA must be at
    // least max( 1, k ).

    // B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is n when TRANSB = 'N' or 'n', and is k otherwise.
    // Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by
    // k part of the array B must contain the
    // matrix B.

    // LDB is INTEGER
    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at
    // least max( 1, k ), otherwise LDB must be at
    // least max( 1, n ).

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

    // C is COMPLEX*16 array of DIMENSION ( LDC, n ).
    // Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be
    // set on entry.
    // On exit, the array C is overwritten by the m by n matrix ( alpha*op( A )*op( B ) + beta*C ).

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, m ).

    // Local Scalars
    Complex temp;
    int info, ncola, nrowa, nrowb;
    bool conja, conjb, nota, notb;

    // Parameters
    Complex one = new Complex(1.0, 0.0);
    Complex zero = new Complex(0.0, 0.0);

```

```

// Set NOTA and NOTB as true if A and B respectively are not conjugated or transposed, set CONJA and CONJB as true if A and B
respectively are to be transposed but not conjugated and set
// NROWA, NCOLA and NROWB as the number of rows and columns of A and the number of rows of B respectively.
nota = (transa.Substring(0, 1).ToUpper() == "N");
notb = (transb.Substring(0, 1).ToUpper() == "N");
conja = (transa.Substring(0, 1).ToUpper() == "C");
conjb = (transb.Substring(0, 1).ToUpper() == "C");
if (nota)
{
    nrowa = m;
    ncola = k;
}
else
{
    nrowa = k;
    ncola = m;
}
if (notb)
{
    nrowb = k;
}
else
{
    nrowb = n;
}

// Test the input parameters
info = 0;
if (!nota && !conja && !(transa.Substring(0, 1).ToUpper() == "T"))
{
    info = 1;
}
else if (!notb && !conjb && !(transb.Substring(0, 1).ToUpper() == "T"))
{
    info = 2;
}
else if (m < 0)
{
    info = 3;
}
else if (n < 0)
{
    info = 4;
}
else if (k < 0)
{
    info = 5;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 8;
}
else if (ldb < Math.Max(1, nrowb))
{
    info = 10;
}
else if (ldc < Math.Max(1, m))
{
    info = 13;
}
if (info != 0)
{
    XERBLA("ZGEMM ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) || (k == 0)) && (beta == one))
{
    return;
}
// And when alpha.eq.zero.
if (alpha == zero)
{
    if (beta == zero)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
    }
    return;
}
// Start the operations.
if (notb)
{
    if (nota)
    {
        // Form C := alpha*A*B + beta*C
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {
                for (int i = 1; i <= m; i = i + 1)
            }
        }
    }
}

```

```

        {
            c[i - 1, j - 1] = beta * c[i - 1, j - 1];
        }
    }
    for (int l = 1; l <= k; l = l + 1)
    {
        if (b[l - 1, j - 1] != zero)
        {
            temp = alpha * b[l - 1, j - 1];
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
            }
        }
    }
}
else if (conja)
{
    // Form C := alpha*A**H*B + beta*C.
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            temp = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp = temp + Complex.Conjugate(a[l - 1, i - 1]) * b[l - 1, j - 1];
            }
            if (beta == zero)
            {
                c[i - 1, j - 1] = alpha * temp;
            }
            else
            {
                c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
            }
        }
    }
}
else
{
    // Form C := alpha*A**T*B + beta*C
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            temp = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp = temp + a[l - 1, i - 1] * b[l - 1, j - 1];
            }
            if (beta == zero)
            {
                c[i - 1, j - 1] = alpha * temp;
            }
            else
            {
                c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
            }
        }
    }
}
}
else if (nota)
{
    if (conjB)
    {
        // Form C := alpha*A*B**H + beta*C.
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }

            for (int l = 1; l <= k; l = l + 1)
            {
                if (b[j - 1, l - 1] != zero)
                {
                    temp = alpha * Complex.Conjugate(b[j - 1, l - 1]);
                    for (int i = 1; i <= m; i = i + 1)
                    {
                        c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
                    }
                }
            }
        }
    }
}
else
{
    // Form C := alpha*A*B**T + beta*C
    for (int j = 1; j <= n; j = j + 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
        else if (beta != one)
        {

```

```

        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
        for (int l = 1; l <= k; l = l + 1)
        {
            if (b[j - 1, l - 1] != zero)
            {
                temp = alpha * b[j - 1, l - 1];
                for (int i = 1; i <= m; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
                }
            }
        }
    }
}
else if (conja)
{
    if (conjb)
    {
        // Form C := alpha*A**H*B**H + beta*C.
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + Complex.Conjugate(a[l - 1, i - 1]) * Complex.Conjugate(b[j - 1, l - 1]);
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
    else
    {
        // Form C := alpha*A**H*B**T + beta*C
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + Complex.Conjugate(a[l - 1, i - 1]) * b[j - 1, l - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
}
else
{
    if (conjb)
    {
        // Form C := alpha*A**T*B**H + beta*C
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + a[l - 1, i - 1] * Complex.Conjugate(b[j - 1, l - 1]);
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
    else
    {
        // Form C := alpha*A**T*B**T + beta*C
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + a[l - 1, i - 1] * b[j - 1, l - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
}
}

```



```

{
    info = 7;
}
else if (ldb < Math.Max(1, m))
{
    info = 9;
}
else if (ldc < Math.Max(1, m))
{
    info = 12;
}
if (info != 0)
{
    XERBLA("ZHEMM ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}
// And when alpha.eq.zero
if (alpha == zero)
{
    if (beta == zero)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
    }
    return;
}
// Start the operations
if (side.Substring(0, 1).ToUpper() == "L")
{
    // Form C := alpha*A*B + beta*C
    if (upper)
    {
        //70
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp1 = alpha * b[i - 1, j - 1];
                temp2 = zero;
                for (int k = 1; k <= i - 1; k = k + 1)
                {
                    c[k - 1, j - 1] = c[k - 1, j - 1] + temp1 * a[k - 1, i - 1];
                    temp2 = temp2 + b[k - 1, j - 1] * Complex.Conjugate(a[k - 1, i - 1]);
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = temp1 * a[i - 1, i - 1].Real + alpha * temp2;
                }
                else
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * a[i - 1, i - 1].Real + alpha * temp2;
                }
            }
        } //70
    }
    else
    {
        //100
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = m; i >= 1; i = i - 1)
            {
                { //DO 90 I = M,1,-1
                    temp1 = alpha * b[i - 1, j - 1];
                    temp2 = zero;
                    for (int k = i + 1; k <= m; k = k + 1)
                    { // DO 80 K = I + 1,M
                        c[k - 1, j - 1] = c[k - 1, j - 1] + temp1 * a[k - 1, i - 1];
                        temp2 = temp2 + b[k - 1, j - 1] * Complex.Conjugate(a[k - 1, i - 1]);
                    }
                    if (beta == zero)
                    {
                        c[i - 1, j - 1] = temp1 * a[i - 1, i - 1].Real + alpha * temp2;
                    }
                    else
                    {
                        c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * a[i - 1, i - 1].Real + alpha * temp2;
                    }
                }
            }
        }
    }
}
else
{
    // Form C := alpha*B*A + beta*C
    //170
    for (int j = 1; j <= n; j = j + 1)
    {
        temp1 = alpha * a[j - 1, j - 1].Real;
        if (beta == zero)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = temp1 * b[i - 1, j - 1];
            }
        }
    }
}
}

```

```

    }
}
else
{
    for (int i = 1; i <= m; i = i + 1)
    {
        c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * b[i - 1, j - 1];
    }
}
for (int k = 1; k <= j - 1; k = k + 1)
{
    if (upper)
    {
        temp1 = alpha * a[k - 1, j - 1];
    }
    else
    {
        temp1 = alpha * Complex.Conjugate(a[j - 1, k - 1]);
    }
    for (int i = 1; i <= m; i = i + 1)
    {
        c[i - 1, j - 1] = c[i - 1, j - 1] + temp1 * b[i - 1, k - 1];
    }
}
for (int k = j + 1; k <= n; k = k + 1)
{
    if (upper)
    {
        temp1 = alpha * Complex.Conjugate(a[j - 1, k - 1]);
    }
    else
    {
        temp1 = alpha * a[k - 1, j - 1];
    }
    for (int i = 1; i <= m; i = i + 1)
    {
        c[i - 1, j - 1] = c[i - 1, j - 1] + temp1 * b[i - 1, k - 1];
    }
}
}
}
}

public static void ZSYMM(string side, string uplo, int m, int n, Complex alpha, Complex[,] a, int lda, Complex[,] b, int ldb, Complex beta, ref
Complex[,] c, int ldc)
{
    // COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)

    // ZSYMM performs one of the matrix-matrix operations
    // C := alpha*A*B + beta*C,
    // or
    // C := alpha*B*A + beta*C,
    // where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

    // SIDE is CHARACTER*1
    // On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:
    // SIDE = 'L' or 'l' C := alpha*A*B + beta*C,
    // SIDE = 'R' or 'r' C := alpha*B*A + beta*C,

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as
    // follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

    // M is INTEGER
    // On entry, M specifies the number of rows of the matrix C. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of the matrix C. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.

    // A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is m when SIDE = 'L' or 'l' and is n otherwise.
    // Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U'
    // or 'u', the leading m by m upper triangular
    // part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not
    // referenced, and when UPLO = 'L' or 'l',
    // the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the
    // strictly upper triangular part of A is not
    // referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that
    // when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric
    // matrix and the strictly lower triangular
    // part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the
    // lower triangular part of the symmetric
    // matrix and the strictly upper triangular part of A is not referenced.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at
    // least max( 1, m ), otherwise LDA must be at
    // least max( 1, n ).

    // B is COMPLEX*16 array of DIMENSION ( LDB, n ).
    // Before entry, the leading m by n part of the array B must contain the matrix B.

    // LDB is INTEGER
    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max( 1, m ).

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

    // C is COMPLEX*16 array of DIMENSION ( LDC, n ).
    // Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be
    // set on entry.
    // On exit, the array C is overwritten by the m by n updated matrix.

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, m ).

    // Local Scalars
    Complex temp1, temp2;
    int info, nrowa;
    bool upper;

```

```

// Parameters
Complex one = new Complex(1.0, 0.0);
Complex zero = new Complex(0.0, 0.0);

// Set NROWA as the number of rows of A
if (side.Substring(0, 1).ToUpper() == "L")
{
    nrowa = m;
}
else
{
    nrowa = n;
}
upper = (uplo.Substring(0, 1).ToUpper() == "U");

// Test the input parameters
info = 0;
if (!(side.Substring(0, 1).ToUpper() == "L") && !(side.Substring(0, 1).ToUpper() == "R"))
{
    info = 1;
}
else if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 2;
}
else if (m < 0)
{
    info = 3;
}
else if (n < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 7;
}
else if (ldb < Math.Max(1, m))
{
    info = 9;
}
else if (ldc < Math.Max(1, m))
{
    info = 12;
}
if (info != 0)
{
    XERBLA("ZSYMM ", info);
    return;
}

// Quick return if possible
if ((m == 0) || (n == 0) || ((alpha == zero) && (beta == one)))
{
    return;
}

// And when alpha.eq.zero
if (alpha == zero)
{
    if (beta == zero)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
    }
    return;
}

// Start the operations
if (side.Substring(0, 1).ToUpper() == "L")
{
    // Form C := alpha*A*B + beta*C
    if (upper)
    {
        //70
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp1 = alpha * b[i - 1, j - 1];
                temp2 = zero;
                for (int k = 1; k <= i - 1; k = k + 1)
                {
                    c[k - 1, j - 1] = c[k - 1, j - 1] + temp1 * a[k - 1, i - 1];
                    temp2 = temp2 + b[k - 1, j - 1] * a[k - 1, i - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = temp1 * a[i - 1, i - 1] + alpha * temp2;
                }
                else
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * a[i - 1, i - 1] + alpha * temp2;
                }
            }
        } //70
    }
    else
    {

```

```

//100
for (int j = 1; j <= n; j = j + 1)
{
    for (int i = m; i >= 1; i = i - 1)
    { //DO 90 I = M,1,-1
        temp1 = alpha * b[i - 1, j - 1];
        temp2 = zero;
        for (int k = i + 1; k <= m; k = k + 1)
        { //DO 80 K = I + 1, M
            c[k - 1, j - 1] = c[k - 1, j - 1] + temp1 * a[k - 1, i - 1];
            temp2 = temp2 + b[k - 1, j - 1] * a[k - 1, i - 1];
        }
        if (beta == zero)
        {
            c[i - 1, j - 1] = temp1 * a[i - 1, i - 1] + alpha * temp2;
        }
        else
        {
            c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * a[i - 1, i - 1] + alpha * temp2;
        }
    }
}
}
else
{
    // Form C := alpha*B*A + beta*C
    //170
    for (int j = 1; j <= n; j = j + 1)
    {
        temp1 = alpha * a[j - 1, j - 1];
        if (beta == zero)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = temp1 * b[i - 1, j - 1];
            }
        }
        else
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1] + temp1 * b[i - 1, j - 1];
            }
        }
        for (int k = 1; k <= j - 1; k = k + 1)
        {
            if (upper)
            {
                temp1 = alpha * a[k - 1, j - 1];
            }
            else
            {
                temp1 = alpha * a[j - 1, k - 1];
            }
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = c[i - 1, j - 1] + temp1 * b[i - 1, k - 1];
            }
        }
        for (int k = j + 1; k <= n; k = k + 1)
        {
            if (upper)
            {
                temp1 = alpha * a[j - 1, k - 1];
            }
            else
            {
                temp1 = alpha * a[k - 1, j - 1];
            }
            for (int i = 1; i <= m; i = i + 1)
            {
                c[i - 1, j - 1] = c[i - 1, j - 1] + temp1 * b[i - 1, k - 1];
            }
        }
    }
}
}
public static void ZTRMM(string side, string uplo, string transa, string diag, int m, int n, Complex alpha, Complex[,] a, int lda, ref Complex[,]
b, int ldb)
{
    // COMPLEX*16 A(LDA,*),B(LDB,*)

    // ZTRMM performs one of the matrix-matrix operations
    // B := alpha*op( A )*B, or B := alpha*B*op( A )
    // where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op( A ) is one of
    // op( A ) = A or op( A ) = A**T or op( A ) = A**H.

    // SIDE is CHARACTER*1
    // On entry, SIDE specifies whether op( A ) multiplies B from the left or right as follows:
    // SIDE = 'L' or 'l' B := alpha*op( A )*B.
    // SIDE = 'R' or 'r' B := alpha*B*op( A ).

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the form of op( A ) to be used in the matrix multiplication as follows:
    // TRANS = 'N' or 'n' op( A ) = A.
    // TRANS = 'T' or 't' op( A ) = A**T.
    // TRANS = 'C' or 'c' op( A ) = A**H.

    // DIAG is CHARACTER*1
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    // DIAG = 'U' or 'u' A is assumed to be unit triangular.
    // DIAG = 'N' or 'n' A is not assumed to be unit triangular.

    // M is INTEGER
    // On entry, M specifies the number of rows of B. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of B. N must be at least zero.

```

```

// ALPHA is COMPLEX*16
// On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

// A is COMPLEX*16 array of DIMENSION ( LDA, k ), where k is m when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'.
// Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix
and the strictly lower triangular part of
// A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the
lower
// triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements
of
// A are not referenced either, but are assumed to be unity.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at
least max( 1, m ), when SIDE = 'R' or 'r'
// then LDA must be at least max( 1, n ).

// B is (input/output) COMPLEX*16 array of DIMENSION ( LDB, n ).
// Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed
matrix.

// LDB is INTEGER
// On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max( 1, m ).

// Parameters
Complex zero = new Complex(0.0, 0.0);
Complex one = new Complex(1.0, 0.0);

// Local Scalars
Complex temp;
int info, nrowa; //i,j,k
bool lside, noconj, nounit, upper;

//kx = 0; // added

// Test the input parameters
lside = (side.Substring(0, 1).ToUpper() == "L");
if (lside)
{
    nrowa = m;
}
else
{
    nrowa = n;
}
noconj = (transa.Substring(0, 1).ToUpper() == "T");
nounit = (diag.Substring(0, 1).ToUpper() == "N");
upper = (uplo.Substring(0, 1).ToUpper() == "U");

//
info = 0;
if (!lside && !(side.Substring(0, 1).ToUpper() == "R"))
{
    info = 1;
}
else if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 2;
}
else if (!(transa.Substring(0, 1).ToUpper() == "N") && !(transa.Substring(0, 1).ToUpper() == "T") && !(transa.Substring(0, 1).ToUpper() ==
"C"))
{
    info = 3;
}
else if (!(diag.Substring(0, 1).ToUpper() == "U") && !(diag.Substring(0, 1).ToUpper() == "N"))
{
    info = 4;
}
else if (m < 0)
{
    info = 5;
}
else if (n < 0)
{
    info = 6;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 9;
}
else if (ldb < Math.Max(1, m))
{
    info = 11;
}

if (info != 0)
{
    XERBLA("ZTRMM ", info);
    return;
}

// Quick return if possible
if (m == 0 || n == 0)
{
    return;
}

// And when alpha.eq.zero.
if (alpha == zero)
{
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            b[i - 1, j - 1] = zero;
        }
    }
    return;
}

// Start the operations.
if (lside)
{
    if (transa.Substring(0, 1).ToUpper() == "N")

```

```

{
    // Form B := alpha*A*B.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int k = 1; k <= m; k = k + 1)
            {
                if (b[k - 1, j - 1] != zero)
                {
                    temp = alpha * b[k - 1, j - 1];
                    for (int i = 1; i <= k - 1; i = i + 1)
                    {
                        b[i - 1, j - 1] = b[i - 1, j - 1] + temp * a[i - 1, k - 1];
                    }
                    if (nounit)
                    {
                        temp = temp * a[k - 1, k - 1];
                    }
                    b[k - 1, j - 1] = temp;
                }
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int k = m; k >= 1; k = k - 1)
            {
                if (b[k - 1, j - 1] != zero)
                {
                    temp = alpha * b[k - 1, j - 1];
                    b[k - 1, j - 1] = temp;
                    if (nounit)
                    {
                        b[k - 1, j - 1] = b[k - 1, j - 1] * a[k - 1, k - 1];
                    }
                    for (int i = k + 1; i <= m; i = i + 1)
                    {
                        b[i - 1, j - 1] = b[i - 1, j - 1] + temp * a[i - 1, k - 1];
                    }
                }
            }
        }
    }
}
else
{
    // Form B := alpha*A**T*B or B := alpha*A**H*B.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = m; i >= 1; i = i - 1)
            {
                temp = b[i - 1, j - 1];
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * a[i - 1, i - 1];
                    }
                    for (int k = 1; k <= i - 1; k = k + 1)
                    {
                        temp = temp + a[k - 1, i - 1] * b[k - 1, j - 1];
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(a[i - 1, i - 1]);
                    }
                    for (int k = 1; k <= i - 1; k = k + 1)
                    {
                        temp = temp + Complex.Conjugate(a[k - 1, i - 1]) * b[k - 1, j - 1];
                    }
                }
                b[i - 1, j - 1] = alpha * temp;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = b[i - 1, j - 1];
                if (noconj)
                {
                    if (nounit)
                    {
                        temp = temp * a[i - 1, i - 1];
                    }
                    for (int k = i + 1; k <= m; k = k + 1)
                    {
                        temp = temp + a[k - 1, i - 1] * b[k - 1, j - 1];
                    }
                }
                else
                {
                    if (nounit)
                    {
                        temp = temp * Complex.Conjugate(a[i - 1, i - 1]);
                    }
                    for (int k = i + 1; k <= m; k = k + 1)
                    {
                        temp = temp + Complex.Conjugate(a[k - 1, i - 1]) * b[k - 1, j - 1];
                    }
                }
                b[i - 1, j - 1] = alpha * temp;
            }
        }
    }
}

```

```

    }
}
}
else
{
    if (transa.Substring(0, 1).ToUpper() == "N")
    {
        // Form B := alpha*B*A.
        if (upper)
        {
            for (int j = n; j >= 1; j = j - 1)
            {
                temp = alpha;
                if (nounit)
                {
                    temp = temp * a[j - 1, j - 1];
                }
                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = temp * b[i - 1, j - 1];
                }
                for (int k = 1; k <= j - 1; k = k + 1)
                {
                    if (a[k - 1, j - 1] != zero)
                    {
                        temp = alpha * a[k - 1, j - 1];
                        for (int i = 1; i <= m; i = i + 1)
                        {
                            b[i - 1, j - 1] = b[i - 1, j - 1] + temp * b[i - 1, k - 1];
                        }
                    }
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                temp = alpha;
                if (nounit)
                {
                    temp = temp * a[j - 1, j - 1];
                }
                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = temp * b[i - 1, j - 1];
                }
                for (int k = j + 1; k <= n; k = k + 1)
                {
                    if (a[k - 1, j - 1] != zero)
                    {
                        temp = alpha * a[k - 1, j - 1];
                        for (int i = 1; i <= m; i = i + 1)
                        {
                            b[i - 1, j - 1] = b[i - 1, j - 1] + temp * b[i - 1, k - 1];
                        }
                    }
                }
            }
        }
    }
}
else
{
    //! Form B := alpha*B*A**T or B := alpha*B*A**H.
    if (upper)
    {
        for (int k = 1; k <= n; k = k + 1)
        {
            for (int j = 1; j <= k - 1; j = j + 1)
            {
                if (a[j - 1, k - 1] != zero)
                {
                    if (noconj)
                    {
                        temp = alpha * a[j - 1, k - 1];
                    }
                    else
                    {
                        temp = alpha * Complex.Conjugate(a[j - 1, k - 1]);
                    }
                    for (int i = 1; i <= m; i = i + 1)
                    {
                        b[i - 1, j - 1] = b[i - 1, j - 1] + temp * b[i - 1, k - 1];
                    }
                }
            }
            temp = alpha;
            if (nounit)
            {
                if (noconj)
                {
                    temp = temp * a[k - 1, k - 1];
                }
                else
                {
                    temp = temp * Complex.Conjugate(a[k - 1, k - 1]);
                }
            }
            if (temp != one)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, k - 1] = temp * b[i - 1, k - 1];
                }
            }
        }
    }
}
else
{
    for (int k = n; k >= 1; k = k - 1)
    {

```

```

        for (int j = k + 1; j <= n; j = j + 1)
        {
            if (a[j - 1, k - 1] != zero)
            {
                if (noconj)
                {
                    temp = alpha * a[j - 1, k - 1];
                }
                else
                {
                    temp = alpha * Complex.Conjugate(a[j - 1, k - 1]);
                }

                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = b[i - 1, j - 1] + temp * b[i - 1, k - 1];
                }
            }
        }
        temp = alpha;
        if (nunit)
        {
            if (noconj)
            {
                temp = temp * a[k - 1, k - 1];
            }
            else
            {
                temp = temp * Complex.Conjugate(a[k - 1, k - 1]);
            }
        }
        if (temp != one)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                b[i - 1, k - 1] = temp * b[i - 1, k - 1];
            }
        }
    }
}

}

public static void ZTRSM(string side, string uplo, string transa, string diag, int m, int n, Complex alpha, Complex[,] a, int lda, ref Complex[,]
b, int ldb)
{
    // COMPLEX*16 A(LDA,*),B(LDB,*)

    // ZTRSM solves one of the matrix equations
    // op( A )*X = alpha*B, or X*op( A ) = alpha*B,
    // where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op( A ) is one of
    // op( A ) = A or op( A ) = A**T or op( A ) = A**H.
    // The matrix X is overwritten on B.

    // SIDE is CHARACTER*1
    // On entry, SIDE specifies whether op( A ) appears on the left or right of X as follows:
    // SIDE = 'L' or 'l' op( A )*X = alpha*B.
    // SIDE = 'R' or 'r' X*op( A ) = alpha*B.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:
    // UPLO = 'U' or 'u' A is an upper triangular matrix.
    // UPLO = 'L' or 'l' A is a lower triangular matrix.

    // TRANSA is CHARACTER*1
    // On entry, TRANSA specifies the form of op( A ) to be used in the matrix multiplication as follows:
    // TRANSA = 'N' or 'n' op( A ) = A.
    // TRANSA = 'T' or 't' op( A ) = A**T.
    // TRANSA = 'C' or 'c' op( A ) = A**H.

    // DIAG is CHARACTER*1
    // On entry, DIAG specifies whether or not A is unit triangular as follows:
    // DIAG = 'U' or 'u' A is assumed to be unit triangular.
    // DIAG = 'N' or 'n' A is not assumed to be unit triangular.

    // M is INTEGER
    // On entry, M specifies the number of rows of B. M must be at least zero.

    // N is INTEGER
    // On entry, N specifies the number of columns of B. N must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

    // A is COMPLEX*16 array of DIMENSION ( LDA, k ), where k is m when SIDE = 'L' or 'l' and k is n when SIDE = 'R' or 'r'.
    // Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix
    and the strictly lower triangular part of
    // A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the
    lower
    // triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements
    of
    // A are not referenced either, but are assumed to be unity.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at
    least max( 1, m ), when SIDE = 'R' or 'r'
    // then LDA must be at least max( 1, n ).

    // B is COMPLEX*16 array of DIMENSION ( LDB, n ).
    // Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the
    solution matrix X.

    // LDB is INTEGER
    // On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max( 1, m ).
    // Parameters
    Complex zero = new Complex(0.0, 0.0);
    Complex one = new Complex(1.0, 0.0);

    // Local Scalars
    Complex temp;
    int info, nrowa; //i,j,k
    bool lside, noconj, nunit, upper;

```

```

//kx = 0; // added

// Test the input parameters
lside = (side.Substring(0, 1).ToUpper() == "L");
if (lside)
{
    nrowa = m;
}
else
{
    nrowa = n;
}
noconj = (transa.Substring(0, 1).ToUpper() == "T");
nount = (diag.Substring(0, 1).ToUpper() == "N");
upper = (uplo.Substring(0, 1).ToUpper() == "U");

//
info = 0;
if (!lside && !(side.Substring(0, 1).ToUpper() == "R"))
{
    info = 1;
}
else if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 2;
}
else if (!(transa.Substring(0, 1).ToUpper() == "N") && !(transa.Substring(0, 1).ToUpper() == "T") && !(transa.Substring(0, 1).ToUpper() ==
"C"))
{
    info = 3;
}
else if !(diag.Substring(0, 1).ToUpper() == "U") && !(diag.Substring(0, 1).ToUpper() == "N"))
{
    info = 4;
}
else if (m < 0)
{
    info = 5;
}
else if (n < 0)
{
    info = 6;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 9;
}
else if (ldb < Math.Max(1, m))
{
    info = 11;
}

if (info != 0)
{
    XERBLA("ZTRSM ", info);
    return;
}

// Quick return if possible
if (m == 0 || n == 0)
{
    return;
}
// And when alpha.eq.zero.
if (alpha == zero)
{
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            b[i - 1, j - 1] = zero;
        }
    }
    return;
}

// Start the operations.
if (lside)
{
    if (transa.Substring(0, 1).ToUpper() == "N")
    {
        // Form B := alpha*inv( A )*B.
        if (upper)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (alpha != one)
                {
                    for (int i = 1; i <= m; i = i + 1)
                    {
                        b[i - 1, j - 1] = alpha * b[i - 1, j - 1];
                    }
                }
                for (int k = m; k >= 1; k = k - 1)
                {
                    if (b[k - 1, j - 1] != zero)
                    {
                        if (nount)
                        {
                            b[k - 1, j - 1] = b[k - 1, j - 1] / a[k - 1, k - 1];
                        }

                        for (int i = 1; i <= k - 1; i = i + 1)
                        {
                            b[i - 1, j - 1] = b[i - 1, j - 1] - b[k - 1, j - 1] * a[i - 1, k - 1];
                        }
                    }
                }
            }
        }
        // Console.WriteLine("out m=1, n=1, side=L, uplo=U, transa=N, diag=N, alpha=1");
    }
    else
    {

```

```

for (int j = 1; j <= n; j = j + 1)
{
    if (alpha != one)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            b[i - 1, j - 1] = alpha * b[i - 1, j - 1];
        }
        for (int k = 1; k <= m; k = k + 1)
        {
            if (b[k - 1, j - 1] != zero)
            {
                if (nounit)
                {
                    b[k - 1, j - 1] = b[k - 1, j - 1] / a[k - 1, k - 1];
                }
                for (int i = k + 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = b[i - 1, j - 1] - b[k - 1, j - 1] * a[i - 1, k - 1];
                }
            }
        }
    }
}
}
else
{
    // Form B := alpha*inv( A**T )*B
    // or B := alpha*inv( A**H )*B.

    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                temp = alpha * b[i - 1, j - 1];
                if (noconj)
                {
                    for (int k = 1; k <= i - 1; k = k + 1)
                    {
                        temp = temp - a[k - 1, i - 1] * b[k - 1, j - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / a[i - 1, i - 1];
                    }
                }
                else
                {
                    for (int k = 1; k <= i - 1; k = k + 1)
                    {
                        temp = temp - Complex.Conjugate(a[k - 1, i - 1]) * b[k - 1, j - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / Complex.Conjugate(a[i - 1, i - 1]);
                    }
                }
                b[i - 1, j - 1] = temp;
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = m; i >= 1; i = i - 1)
            {
                temp = alpha * b[i - 1, j - 1];
                if (noconj)
                {
                    for (int k = i + 1; k <= m; k = k + 1)
                    {
                        temp = temp - a[k - 1, i - 1] * b[k - 1, j - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / a[i - 1, i - 1];
                    }
                }
                else
                {
                    for (int k = i + 1; k <= m; k = k + 1)
                    {
                        temp = temp - Complex.Conjugate(a[k - 1, i - 1]) * b[k - 1, j - 1];
                    }
                    if (nounit)
                    {
                        temp = temp / Complex.Conjugate(a[i - 1, i - 1]);
                    }
                }
                b[i - 1, j - 1] = temp;
            }
        }
    }
}
}
else
{
    if (transa.Substring(0, 1).ToUpper() == "N")
    {
        // Form B := alpha*B*inv( A ).
        if (upper)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                if (alpha != one)
                {
                    for (int i = 1; i <= m; i = i + 1)
                    {

```

```

        b[i - 1, j - 1] = alpha * b[i - 1, j - 1];
    }
}
for (int k = 1; k <= j - 1; k = k + 1)
{
    if (a[k - 1, j - 1] != zero)
    {
        for (int i = 1; i <= m; i = i + 1)
        {
            b[i - 1, j - 1] = b[i - 1, j - 1] - a[k - 1, j - 1] * b[i - 1, k - 1];
        }
    }
}

if (nounit)
{
    temp = one / a[j - 1, j - 1];
    for (int i = 1; i <= m; i = i + 1)
    {
        b[i - 1, j - 1] = temp * b[i - 1, j - 1];
    }
}
}
}
else
{
    for (int j = n; j >= 1; j = j - 1)
    {
        if (alpha != one)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                b[i - 1, j - 1] = alpha * b[i - 1, j - 1];
            }
        }
        for (int k = j + 1; k <= n; k = k + 1)
        {
            if (a[k - 1, j - 1] != zero)
            {
                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = b[i - 1, j - 1] - a[k - 1, j - 1] * b[i - 1, k - 1];
                }
            }
        }
        if (nounit)
        {
            temp = one / a[j - 1, j - 1];
            for (int i = 1; i <= m; i = i + 1)
            {
                b[i - 1, j - 1] = temp * b[i - 1, j - 1];
            }
        }
    }
}
}
}
else
{
    // Form B := alpha*B*inv( A**T )
    // or B := alpha*B*inv( A**H ).
    if (upper)
    {
        for (int k = n; k >= 1; k = k - 1)
        {
            if (nounit)
            {
                if (noconj)
                {
                    temp = one / a[k - 1, k - 1];
                }
                else
                {
                    temp = one / Complex.Conjugate(a[k - 1, k - 1]);
                }

                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, k - 1] = temp * b[i - 1, k - 1];
                }
            }
        }
        for (int j = 1; j <= k - 1; j = j + 1)
        {
            if (a[j - 1, k - 1] != zero)
            {
                if (noconj)
                {
                    temp = a[j - 1, k - 1];
                }
                else
                {
                    temp = Complex.Conjugate(a[j - 1, k - 1]);
                }

                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, j - 1] = b[i - 1, j - 1] - temp * b[i - 1, k - 1];
                }
            }
        }
        if (alpha != one)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                b[i - 1, k - 1] = alpha * b[i - 1, k - 1];
            }
        }
    }
}
}
}
else

```

```

    {
        for (int k = 1; k <= n; k = k + 1)
        {
            if (nounit)
            {
                if (noconj)
                {
                    temp = one / a[k - 1, k - 1];
                }
                else
                {
                    temp = one / Complex.Conjugate(a[k - 1, k - 1]);
                }

                for (int i = 1; i <= m; i = i + 1)
                {
                    b[i - 1, k - 1] = temp * b[i - 1, k - 1];
                }
            }
            for (int j = k + 1; j <= n; j = j + 1)
            {
                if (a[j - 1, k - 1] != zero)
                {
                    if (noconj)
                    {
                        temp = a[j - 1, k - 1];
                    }
                    else
                    {
                        temp = Complex.Conjugate(a[j - 1, k - 1]);
                    }

                    for (int i = 1; i <= m; i = i + 1)
                    {
                        b[i - 1, j - 1] = b[i - 1, j - 1] - temp * b[i - 1, k - 1];
                    }
                }
            }
        }

        if (alpha != one)
        {
            for (int i = 1; i <= m; i = i + 1)
            {
                b[i - 1, k - 1] = alpha * b[i - 1, k - 1];
            }
        }
    }
}

public static void ZHERK(string uplo, string trans, int n, int k, double alpha, Complex[,] a, int lda, double beta, ref Complex[,] c, int ldc)
{
    //COMPLEX*16 A(LDA,*),C(LDC,*)

    // ZHERK performs one of the hermitian rank k operations
    // C := alpha*A*A**H + beta*C,
    // or
    // C := alpha*A**H*A + beta*C,
    // where alpha and beta are real scalars, C is an n by n hermitian matrix and A is an n by k matrix in the first case and a k by n
    // matrix in the second case.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' C := alpha*A*A**H + beta*C.
    // TRANS = 'C' or 'c' C := alpha*A**H*A + beta*C.

    // N is INTEGER
    // On entry, N specifies the order of the matrix C. N must be at least zero.

    // K is INTEGER
    // On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or
    // 'c', K specifies the number of rows of the
    // matrix A. K must be at least zero.

    // ALPHA is DOUBLE PRECISION .
    // On entry, ALPHA specifies the scalar alpha.
    // A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS =
    // 'N' or 'n', the leading n by k
    // part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
    // at least max( 1, n ), otherwise LDA must
    // be at least max( 1, k ).

    // BETA is DOUBLE PRECISION.
    // On entry, BETA specifies the scalar beta.

    // C is COMPLEX*16 array of DIMENSION ( LDC, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
    // the hermitian matrix and the strictly
    // lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
    // part of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
    // the hermitian matrix and the strictly
    // upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
    // part of the updated matrix.
    // Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

    // Local Scalars
    Complex temp;
    double rtemp;

```

```

int info, nrowa;
bool upper;

// Parameters
const double one = 1.0;
const double zero = 0.0;

// Test the input parameters
if (trans.Substring(0, 1).ToUpper() == "N")
{
    nrowa = n;
}
else
{
    nrowa = k;
}
upper = (uplo.Substring(0, 1).ToUpper() == "U");

info = 0;
if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "C"))
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (k < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 7;
}
else if (ldc < Math.Max(1, n))
{
    info = 10;
}
if (info != 0)
{
    XERBLA("ZHERK ", info);
    return;
}

// Quick return if possible.
if ((n == 0) || (((alpha == zero) || (k == 0)) && (beta == one)))
{
    return;
}
// And when alpha.eq.zero.
if (alpha == zero)
{
    //Console.WriteLine("a=0");
    if (upper)
    {
        //Console.WriteLine("a=0 1");
        if (beta == zero)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
                c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            }
        }
    }
    else
    {
        //Console.WriteLine("a=0 2");
        if (beta == zero)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }
        }
    }
}
return;
}

// Start the operations.
if (trans.Substring(0, 1).ToUpper() == "N")
{

```

```

// Form C := alpha*A*A**H + beta*C.
if (upper)
{
    //Console.WriteLine("bef dsyrk 1");
    for (int j = 1; j <= n; j = j + 1)
    {
        if (beta == zero)
        {
            for (int i = 1; i <= j; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
        else if (beta != one)
        {
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
            c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
        }
        else
        {
            c[j - 1, j - 1] = c[j - 1, j - 1].Real;
        }
        for (int l = 1; l <= k; l = l + 1)
        {
            if (a[j - 1, l - 1] != new Complex(zero, zero))
            {
                //DCMLX(ZERO)
                temp = alpha * Complex.Conjugate(a[j - 1, l - 1]);
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
                }
                c[j - 1, j - 1] = c[j - 1, j - 1].Real + (temp * a[j - 1, l - 1]).Real; // changed a[i-1 to a[j-1...tested and I=J here
            }
        }
    }
}
else
{
    //Console.WriteLine("bef dsyrk 2");
    for (int j = 1; j <= n; j = j + 1)
    {
        if (beta == zero)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
        else if (beta != one)
        {
            c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            for (int i = j + 1; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
        else
        {
            c[j - 1, j - 1] = c[j - 1, j - 1].Real;
        }
        for (int l = 1; l <= k; l = l + 1)
        {
            if (a[j - 1, l - 1] != new Complex(zero, zero))
            {
                temp = alpha * Complex.Conjugate(a[j - 1, l - 1]);
                c[j - 1, j - 1] = c[j - 1, j - 1].Real + (temp * a[j - 1, l - 1]).Real;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
                }
            }
        }
    }
}
}
else
{
    // Form C := alpha*A*A**H*A + beta*C.
    if (upper)
    {
        //Console.WriteLine("bef dsyrk 3");
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= j - 1; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + Complex.Conjugate(a[l - 1, i - 1]) * a[l - 1, j - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
            rtemp = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                rtemp = rtemp + (Complex.Conjugate(a[l - 1, j - 1]) * a[l - 1, j - 1]).Real; //added .Real
            }
            if (beta == zero)
            {
                c[j - 1, j - 1] = alpha * rtemp;
            }
            else
            {
                c[j - 1, j - 1] = alpha * rtemp + beta * c[j - 1, j - 1].Real;
            }
        }
    }
}

```

```

    }
    //Console.WriteLine("dsyrk 3");
}
else
{
    //Console.WriteLine("bef dsyrk 4");
    for (int j = 1; j <= n; j = j + 1)
    {
        rtemp = zero;
        for (int l = 1; l <= k; l = l + 1)
        {
            rtemp = rtemp + (Complex.Conjugate(a[l - 1, j - 1]) * a[l - 1, j - 1]).Real; //added .Real
        }
        if (beta == zero)
        {
            c[j - 1, j - 1] = alpha * rtemp;
        }
        else
        {
            c[j - 1, j - 1] = alpha * rtemp + beta * c[j - 1, j - 1].Real;
        }
        for (int i = j + 1; i <= n; i = i + 1)
        {
            temp = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp = temp + Complex.Conjugate(a[l - 1, i - 1]) * a[l - 1, j - 1]; //added .Real
            }
            if (beta == zero)
            {
                c[i - 1, j - 1] = alpha * temp;
            }
            else
            {
                c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
            }
        }
    }
}
}

public static void ZSYRK(string uplo, string trans, int n, int k, Complex alpha, Complex[,] a, int lda, Complex beta, ref Complex[,] c, int ldc)
{
    // COMPLEX*16 A(LDA,*),C(LDC,*)
    // ZSYRK performs one of the symmetric rank k operations
    // C := alpha*A*A**T + beta*C,
    // or
    // C := alpha*A**T*A + beta*C,
    // where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix
    // in the second case.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

    // TRANS is CHARACTER*1
    // On entry, TRANS specifies the operation to be performed as follows:
    // TRANS = 'N' or 'n' C := alpha*A*A**T + beta*C.
    // TRANS = 'T' or 't' C := alpha*A**T*A + beta*C.

    // N is INTEGER
    // On entry, N specifies the order of the matrix C. N must be at least zero.

    // K is INTEGER
    // On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or
    // 't', K specifies the number of rows of the
    // matrix A. K must be at least zero.

    // ALPHA is COMPLEX*16
    // On entry, ALPHA specifies the scalar alpha.
    // A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS =
    // 'N' or 'n', the leading n by k
    // part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

    // LDA is INTEGER
    // On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
    // at least max( 1, n ), otherwise LDA must
    // be at least max( 1, k ).

    // BETA is COMPLEX*16
    // On entry, BETA specifies the scalar beta.

    // C is COMPLEX*16 array of DIMENSION ( LDC, n ).
    // Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
    // the symmetric matrix and the strictly
    // lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
    // part of the updated matrix.
    // Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
    // the symmetric matrix and the strictly
    // upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
    // part of the updated matrix.

    // LDC is INTEGER
    // On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

    // Local Scalars
    Complex temp;
    int info, nrowa;
    bool upper;

    // Parameters
    Complex one = new Complex(1.0, 0.0);
    Complex zero = new Complex(0.0, 0.0);

    // Test the input parameters
    if (trans.Substring(0, 1).ToUpper() == "N")
    {
        nrowa = n;
    }
}

```

```

else
{
    nrowa = k;
}
upper = (uplo.Substring(0, 1).ToUpper() == "U");
info = 0;
if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T"))
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (k < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 7;
}
else if (ldc < Math.Max(1, n))
{
    info = 10;
}
if (info != 0)
{
    XERBLA("ZSYRK ", info);
    return;
}

// Quick return if possible.
if ((n == 0) || ((alpha == zero) || (k == 0)) && (beta == one))
{
    return;
}

// And when alpha.eq.zero.
if (alpha == zero)
{
    if (upper)
    {
        if (beta == zero)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }
        }
    }
    else
    {
        if (beta == zero)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }
        }
    }
    return;
}

// Start the operations.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    // Form C := alpha*A*A**T + beta*C.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {
                for (int i = 1; i <= j; i = i + 1)
                {

```

```

        c[i - 1, j - 1] = beta * c[i - 1, j - 1];
    }
}
for (int l = 1; l <= k; l = l + 1)
{
    if (a[j - 1, l - 1] != zero)
    {
        temp = alpha * a[j - 1, l - 1];
        for (int i = 1; i <= j; i = i + 1)
        {
            c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
        }
    }
}
}
}
else
{
    for (int j = 1; j <= n; j = j + 1)
    {
        if (beta == zero)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
        else if (beta != one)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }
        for (int l = 1; l <= k; l = l + 1)
        {
            if (a[j - 1, l - 1] != zero)
            {
                temp = alpha * a[j - 1, l - 1];
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + temp * a[i - 1, l - 1];
                }
            }
        }
    }
}
}
else
{
    // Form C := alpha*A**T*A + beta*C.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= j; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + a[l - 1, i - 1] * a[l - 1, j - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                temp = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp = temp + a[l - 1, i - 1] * a[l - 1, j - 1];
                }
                if (beta == zero)
                {
                    c[i - 1, j - 1] = alpha * temp;
                }
                else
                {
                    c[i - 1, j - 1] = alpha * temp + beta * c[i - 1, j - 1];
                }
            }
        }
    }
}
}
}

public static void ZHER2K(string uplo, string trans, int n, int k, Complex alpha, Complex[,] a, int lda, Complex[,] b, int ldb, double beta, ref
Complex[,] c, int ldc)
{
    // COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)

    // ZHER2K performs one of the hermitian rank 2k operations
    // C := alpha*A*B**H + conjg( alpha )*B*A**H + beta*C,
    // or
    // C := alpha*A**H*B + conjg( alpha )*B**H*A + beta*C,
    // where alpha and beta are scalars with beta real, C is an n by n hermitian matrix and A and B are n by k matrices in the first case
    // and k by n matrices in the second case.

    // UPLO is CHARACTER*1
    // On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
    // UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
    // UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

```

```

// TRANS is CHARACTER*1
// On entry, TRANS specifies the operation to be performed as follows:
// TRANS = 'N' or 'n' C := alpha*A*B**H +
// conjg( alpha )*B*A**H +
// beta*C.
// TRANS = 'C' or 'c' C := alpha*A**H*B +
// conjg( alpha )*B**H*A +
// beta*C.

// N is INTEGER
// On entry, N specifies the order of the matrix C. N must be at least zero.

// K is INTEGER
// On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'C' or
'c', K specifies the number of rows of the
// matrices A and B. K must be at least zero.

// ALPHA is COMPLEX*16 .
// On entry, ALPHA specifies the scalar alpha.

// A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise.
// Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by
n part of the array A must contain the
// matrix A.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
at least max( 1, n ), otherwise LDA must
// be at least max( 1, k ).

// B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is k when TRANS = 'N' or 'n', and is n otherwise.
// Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by
n part of the array B must contain the
// matrix B.

// LDB is INTEGER
// On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be
at least max( 1, n ), otherwise LDB must
// be at least max( 1, k ).

// BETA is DOUBLE PRECISION .
// On entry, BETA specifies the scalar beta.

// C is COMPLEX*16 array of DIMENSION ( LDC, n ).
// Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
the hermitian matrix and the strictly
// lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
part of the updated matrix.
// Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
the hermitian matrix and the strictly
// upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
part of the updated matrix.
// Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

// LDC is INTEGER
// On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

// Local Scalars
Complex temp1, temp2;
int info, nrowa;
bool upper;

// Parameters
const double one = 1.0;
Complex zero = new Complex(0.0, 0.0);

// Test the input parameters
if (trans.Substring(0, 1).ToUpper() == "N")
{
    nrowa = n;
}
else
{
    nrowa = k;
}
upper = (uplo.Substring(0, 1).ToUpper() == "U");

info = 0;

if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "C"))
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (k < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 7;
}
else if (ldb < Math.Max(1, nrowa))
{
    info = 9;
}
else if (ldc < Math.Max(1, n))
{
    info = 12;
}
if (info != 0)
{
    XERBLA("ZHER2K", info);
    return;
}

```

```

// Quick return if possible.
if ((n == 0) || ((alpha == zero) || (k == 0)) && (beta == one)))
{
    return;
}

// And when alpha.eq.zero.
if (alpha == zero)
{
    if (upper)
    {
        if (beta == zero.Real)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
                c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            }
        }
    }
    else
    {
        if (beta == zero.Real)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }
        }
    }
    return;
}

// Start the operations.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    //! Form C := alpha*A*B**H + conjg(alpha)*B*A**H + C.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero.Real)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {
                for (int i = 1; i <= j - 1; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
                c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            }
            else
            {
                c[j - 1, j - 1] = c[j - 1, j - 1].Real;
            }
            for (int l = 1; l <= k; l = l + 1)
            {
                if ((a[j - 1, l - 1] != zero) || (b[j - 1, l - 1] != zero))
                {
                    temp1 = alpha * Complex.Conjugate(b[j - 1, l - 1]);
                    temp2 = Complex.Conjugate(alpha * a[j - 1, l - 1]);
                    for (int i = 1; i <= j - 1; i = i + 1)
                    {
                        c[i - 1, j - 1] = c[i - 1, j - 1] + a[i - 1, l - 1] * temp1 + b[i - 1, l - 1] * temp2;
                    }
                    c[j - 1, j - 1] = c[j - 1, j - 1].Real + (a[j - 1, l - 1] * temp1 + b[j - 1, l - 1] * temp2).Real;
                }
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero.Real)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {

```

```

        {
            for (int i = j + 1; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
            c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
        }
        else
        {
            c[j - 1, j - 1] = c[j - 1, j - 1].Real;
        }
        for (int l = 1; l <= k; l = l + 1)
        {
            if ((a[j - 1, l - 1] != zero) || (b[j - 1, l - 1] != zero))
            {
                temp1 = alpha * Complex.Conjugate(b[j - 1, l - 1]);
                temp2 = Complex.Conjugate(alpha * a[j - 1, l - 1]);
                for (int i = j + 1; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + a[i - 1, l - 1] * temp1 + b[i - 1, l - 1] * temp2;
                }
                c[j - 1, j - 1] = c[j - 1, j - 1].Real + (a[j - 1, l - 1] * temp1 + b[j - 1, l - 1] * temp2).Real;
            }
        }
    }
}
}
else
    /*! Form C := alpha*A**H*B + conjg( alpha )*B**H*A + C.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            for (int i = 1; i <= j; i = i + 1)
            {
                temp1 = zero;
                temp2 = zero;
                for (int l = 1; l <= k; l = l + 1)
                {
                    temp1 = temp1 + Complex.Conjugate(a[l - 1, i - 1]) * b[l - 1, j - 1];
                    temp2 = temp2 + Complex.Conjugate(b[l - 1, i - 1]) * a[l - 1, j - 1];
                }
                if (i == j)
                {
                    if (beta == zero.Real)
                    {
                        c[i - 1, j - 1] = (alpha * temp1 + Complex.Conjugate(alpha) * temp2).Real;
                    }
                    else
                    {
                        c[i - 1, j - 1] = beta * c[i - 1, j - 1].Real + (alpha * temp1 + Complex.Conjugate(alpha) * temp2).Real;
                    }
                }
                else
                {
                    if (beta == zero.Real)
                    {
                        c[i - 1, j - 1] = alpha * temp1 + Complex.Conjugate(alpha) * temp2;
                    }
                    else
                    {
                        c[i - 1, j - 1] = beta * c[i - 1, j - 1] + alpha * temp1 + Complex.Conjugate(alpha) * temp2;
                    }
                }
            }
        }
    }
}
else
{
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = j; i <= n; i = i + 1)
        {
            temp1 = zero;
            temp2 = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp1 = temp1 + Complex.Conjugate(a[l - 1, i - 1]) * b[l - 1, j - 1];
                temp2 = temp2 + Complex.Conjugate(b[l - 1, i - 1]) * a[l - 1, j - 1];
            }
            if (i == j)
            {
                if (beta == zero.Real)
                {
                    c[j - 1, j - 1] = (alpha * temp1 + Complex.Conjugate(alpha) * temp2).Real;
                }
                else
                {
                    c[j - 1, j - 1] = beta * c[i - 1, j - 1].Real + (alpha * temp1 + Complex.Conjugate(alpha) * temp2).Real;
                }
            }
            else
            {
                if (beta == zero.Real)
                {
                    c[i - 1, j - 1] = alpha * temp1 + Complex.Conjugate(alpha) * temp2;
                }
                else
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1] + alpha * temp1 + Complex.Conjugate(alpha) * temp2;
                }
            }
        }
    }
}
}

public static void ZSYR2K(string uplo, string trans, int n, int k, Complex alpha, Complex[,] a, int lda, Complex[,] b, int ldb, Complex beta, ref
Complex[,] c, int ldc)
{

```

```

//COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)

// ZSYR2K performs one of the symmetric rank 2k operations
// C := alpha*A*B**T + alpha*B*A**T + beta*C,
// or
// C := alpha*A**T*B + alpha*B**T*A + beta*C,
// where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n
matrices in the second case.

// UPLO is CHARACTER*1
// On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:
// UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.
// UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

// TRANS is CHARACTER*1
// On entry, TRANS specifies the operation to be performed as follows:
// TRANS = 'N' or 'n' C := alpha*A*B**T + alpha*B*A**T + beta*C.
// TRANS = 'T' or 't' C := alpha*A**T*B + alpha*B**T*A + beta*C.

// N is INTEGER
// On entry, N specifies the order of the matrix C. N must be at least zero.

// K is INTEGER
// On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or
't', K specifies the number of rows of the
// matrices A and B. K must be at least zero.

// ALPHA is COMPLEX*16
// On entry, ALPHA specifies the scalar alpha.

// A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS =
'N' or 'n',
// the leading n by k
// part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

// LDA is INTEGER
// On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be
at least max( 1, n ), otherwise LDA must
// be at least max( 1, k ).

// B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS =
'N' or 'n',
// the leading n by k
// part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

// LDB is INTEGER
// On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be
at least max( 1, n ), otherwise LDB must
// be at least max( 1, k ).

// BETA is COMPLEX*16
// On entry, BETA specifies the scalar beta.

// C is COMPLEX*16 array of DIMENSION ( LDC, n ).
// Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of
the symmetric matrix and the strictly
// lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular
part of the updated matrix.
// Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of
the symmetric matrix and the strictly
// upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular
part of the updated matrix.

// LDC is INTEGER
// On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max( 1, n ).

// Local Scalars
Complex temp1, temp2;
int info, nrowa;
bool upper;

// Parameters
Complex one = new Complex(1.0,0.0);
Complex zero = new Complex(0.0, 0.0);

// Test the input parameters
if (trans.Substring(0, 1).ToUpper() == "N")
{
    nrowa = n;
}
else
{
    nrowa = k;
}
upper = (uplo.Substring(0, 1).ToUpper() == "U");

info = 0;

if (!upper && !(uplo.Substring(0, 1).ToUpper() == "L"))
{
    info = 1;
}
else if (!(trans.Substring(0, 1).ToUpper() == "N") && !(trans.Substring(0, 1).ToUpper() == "T"))
{
    info = 2;
}
else if (n < 0)
{
    info = 3;
}
else if (k < 0)
{
    info = 4;
}
else if (lda < Math.Max(1, nrowa))
{
    info = 7;
}
else if (ldb < Math.Max(1, nrowa))
{
    info = 9;
}
else if (ldc < Math.Max(1, n))
{
    info = 12;
}

```

```

}
if (info != 0)
{
    XERBLA("ZSYR2K", info);
    return;
}

// Quick return if possible.
if ((n == 0) || ((alpha == zero) || (k == 0)) && (beta == one))
{
    return;
}

// And when alpha.eq.zero.
if (alpha == zero)
{
    if (upper)
    {
        if (beta == zero.Real)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
                //c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            }
        }
    }
    else
    {
        if (beta == zero)
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
        }
        else
        {
            for (int j = 1; j <= n; j = j + 1)
            {
                //c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
            }
        }
    }
    return;
}

// Start the operations.
if (trans.Substring(0, 1).ToUpper() == "N")
{
    /** Form C := alpha*A*B**T + alpha*B*A**T + C.
    if (upper)
    {
        for (int j = 1; j <= n; j = j + 1)
        {
            if (beta == zero)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = zero;
                }
            }
            else if (beta != one)
            {
                for (int i = 1; i <= j; i = i + 1)
                {
                    c[i - 1, j - 1] = beta * c[i - 1, j - 1];
                }
                // c[j - 1, j - 1] = beta * c[j - 1, j - 1].Real;
            }

            for (int l = 1; l <= k; l = l + 1)
            {
                if ((a[j - 1, l - 1] != zero) || (b[j - 1, l - 1] != zero))
                {
                    temp1 = alpha * b[j - 1, l - 1];
                    temp2 = alpha * a[j - 1, l - 1];
                    for (int i = 1; i <= j; i = i + 1)
                    {
                        c[i - 1, j - 1] = c[i - 1, j - 1] + a[i - 1, l - 1] * temp1 + b[i - 1, l - 1] * temp2;
                    }
                    //c[j - 1, j - 1] = c[j - 1, j - 1].Real + (a[j - 1, l - 1] * temp1 + b[j - 1, l - 1] * temp2).Real;
                }
            }
        }
    }
    else
    {
        for (int j = 1; j <= n; j = j + 1)
        {

```

```

        if (beta == zero)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = zero;
            }
        }
        else if (beta != one)
        {
            for (int i = j; i <= n; i = i + 1)
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1];
            }
        }

        for (int l = 1; l <= k; l = l + 1)
        {
            if ((a[j - 1, l - 1] != zero) || (b[j - 1, l - 1] != zero))
            {
                temp1 = alpha * b[j - 1, l - 1];
                temp2 = alpha * a[j - 1, l - 1];
                for (int i = j; i <= n; i = i + 1)
                {
                    c[i - 1, j - 1] = c[i - 1, j - 1] + a[i - 1, l - 1] * temp1 + b[i - 1, l - 1] * temp2;
                }
            }
        }
    }
}

else
//Form C := alpha*A**T*B + alpha*B**T*A + C.
if (upper)
{
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = 1; i <= j; i = i + 1)
        {
            temp1 = zero;
            temp2 = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp1 = temp1 + a[l - 1, i - 1] * b[l - 1, j - 1];
                temp2 = temp2 + b[l - 1, i - 1] * a[l - 1, j - 1];
            }

            if (beta == zero)
            {
                c[i - 1, j - 1] = alpha * temp1 + alpha * temp2;
            }
            else
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1] + alpha * temp1 + alpha * temp2;
            }
        }
    }
}

else
{
    for (int j = 1; j <= n; j = j + 1)
    {
        for (int i = j; i <= n; i = i + 1)
        {
            temp1 = zero;
            temp2 = zero;
            for (int l = 1; l <= k; l = l + 1)
            {
                temp1 = temp1 + a[l - 1, i - 1] * b[l - 1, j - 1];
                temp2 = temp2 + b[l - 1, i - 1] * a[l - 1, j - 1];
            }

            if (beta == zero)
            {
                c[i - 1, j - 1] = alpha * temp1 + alpha * temp2;
            }
            else
            {
                c[i - 1, j - 1] = beta * c[i - 1, j - 1] + alpha * temp1 + alpha * temp2;
            }
        }
    }
}
}

}

public static class misc_s {
    // SLAMCH
    static bool slamch_first = true;
    public static float _base, emax, emin, eps, prec, rmax, rmin, rnd, sfmin, t;
    // SLAMC1
    static bool slamc1_first = true, lieee1, lrnd;
    static int slamc1_lbeta, slamc1_lt;
    // SLAMC2
    static bool slamc2_first = true, iwarn = false;
    static int slamc2_lbeta, lemax, lemin, slamc2_lt;
    static float leps, lrmax, lrmin;
}

```

```

public static float SLAMCH(string cmach)
{
    // C# LAPACK auxiliary routine (version 1.1) of LAPACK SLAMCH
    //
    // SLAMCH determines float precision machine parameters.
    /*
    * Arguments
    * =====
    * cmach = Specifies the value to be returned by SLAMCH:
    *         = 'E' or 'e', SLAMCH := eps
    *         = 'S' or 's', SLAMCH := sfmin
    *         = 'B' or 'b', SLAMCH := base
    *         = 'P' or 'p', SLAMCH := eps*base
    *         = 'N' or 'n', SLAMCH := t
    *         = 'R' or 'r', SLAMCH := rnd
    *         = 'M' or 'm', SLAMCH := emin
    *         = 'U' or 'u', SLAMCH := rmin
    *         = 'L' or 'l', SLAMCH := emax
    *         = 'O' or 'o', SLAMCH := rmax
    *
    * where
    *
    * eps = relative machine precision
    * sfmin = safe minimum, such that 1/sfmin does not overflow
    * base = base of the machine
    * prec = eps*base
    * t = number of (base) digits in the mantissa
    * rnd = 1.0 when rounding occurs in addition, 0.0 otherwise
    * emin = minimum exponent before (gradual) underflow
    * rmin = underflow threshold - base**(emin-1)
    * emax = largest exponent before overflow
    * rmax = overflow threshold - (base**emax)*(1-eps)
    * =====
    */

    // Parameters
    const float one = 1.0F;
    const float zero = 0.0F;

    // Local Scalars
    //static bool first = true;
    bool lrnd = false;
    int beta = 0, imax = 0, imin = 0, it = 0;
    //static float base, emax, emin, eps, prec, rmax, rmin, rnd, sfmin, t;
    float rmach = 0.0F, small = 0.0F;

    if (slamch_first)
    {
        slamch_first = false;
        SLAMC2(ref beta, ref it, ref lrnd, ref eps, ref imin, ref rmin, ref imax, ref rmax);
        //Console.WriteLine("{0} {1} {2} {3} {4} {5} {6} {7}", beta, it, lrnd, eps, imin, rmin, imax, rmax);
        //Console.ReadKey();
        //Environment.Exit(1);
        _base = beta;
        t = it;

        if (lrnd)
        {
            rnd = one;
            eps = (float)Math.Pow(_base, 1 - it) / 2;
        }
        else
        {
            rnd = zero;
            eps = (float)Math.Pow(_base, 1 - it);
        }

        prec = eps * _base;
        emin = imin;
        emax = imax;
        sfmin = rmin;
        small = one / rmax;

        if (small >= sfmin)
        {
            /*
            * Use SMALL plus a bit, to avoid the possibility of rounding
            * causing overflow when computing 1/sfmin.
            */
            sfmin = small * (one + eps);
        }
    }

    switch (cmach.ToUpper().Substring(0,1))
    {
        case "E":
            rmach = eps;
            break;
        case "S":
            rmach = sfmin;
            break;
        case "B":
            rmach = _base;
            break;
        case "P":
            rmach = prec;
            break;
        case "N":
            rmach = t;
            break;
        case "R":
            rmach = rnd;
            break;
        case "M":
            rmach = emin;
            break;
        case "U":
            rmach = rmin;
            break;
        case "L":
            rmach = emax;
            break;
    }
}

```

```

        break;
    case "O":
        rmach = rmax;
        break;
    //default:
}

return rmach;
}

static void SLAMC1(ref int beta, ref int t, ref bool rnd, ref bool ieee1)
{
    // C# LAPACK auxiliary routine (version 1.1) of the original BLAS SLAMC1
    //
    // SLAMC1 determines the machine parameters given by BETA, T, RND, and IEEE1
    /*
    * Arguments
    * =====
    *
    * BETA      (output) INTEGER
    *            The base of the machine.
    *
    * T         (output) INTEGER
    *            The number of ( BETA ) digits in the mantissa.
    *
    * RND       (output) LOGICAL
    *            Specifies whether proper rounding ( RND = .TRUE. ) or
    *            chopping ( RND = .FALSE. ) occurs in addition. This may not
    *            be a reliable guide to the way in which the machine performs
    *            its arithmetic.
    *
    * IEEE1     (output) LOGICAL
    *            Specifies whether rounding appears to be done in the IEEE
    *            'round to nearest' style.
    *
    * Further Details
    * =====
    *
    * The routine is based on the routine ENVRON by Malcolm and
    * incorporates suggestions by Gentleman and Marovich. See
    *
    * Malcolm M. A. (1972) Algorithms to reveal properties of
    * floating-point arithmetic. Comms. of the ACM, 15, 949-951.
    *
    * Gentleman W. M. and Marovich S. B. (1974) More on algorithms
    * that reveal properties of floating point arithmetic units.
    * Comms. of the ACM, 17, 276-277.
    *
    * =====
    */

    // Local Scalars
    //static bool first=true, lieeel, lrnd;
    //static int lbeta, lt;
    float a, b, c, f, one, qtr, savec, t1, t2;

    if (slamcl_first)
    {
        slamcl_first = false;
        one = 1;

        /*
        * LBETA, LIEEE1, LT and LRND are the local values of BETA,
        * IEEE1, T and RND.
        *
        * Throughout this routine we use the function SLAMC3 to ensure
        * that relevant values are stored and not held in registers, or
        * are not affected by optimizers.
        *
        * Compute a = 2.0**m with the smallest positive integer m such
        * that
        *
        *      fl( a + 1.0 ) = a.
        */

        a = 1;
        c = 1;

        while (c == one)
        {
            a = 2 * a;
            c = SLAMC3(a, one);
            c = SLAMC3(c, -a);
        }

        /*
        * Now compute b = 2.0**m with the smallest positive integer m
        * such that
        *
        *      fl( a + b ) .gt. a.
        */

        b = 1;
        c = SLAMC3(a, b);

        while (c == a)
        {
            b = 2 * b;
            c = SLAMC3(a, b);
        }

        /*
        * Now compute the base. a and c are neighbouring floating point
        * numbers in the interval ( beta**t, beta**( t + 1 ) ) and so
        * their difference is beta. Adding 0.25 to c is to ensure that it
        * is truncated to beta and not ( beta - 1 ).
        */
        //Console.WriteLine("{0} {1} {2}", a, b, c);
        //Console.ReadKey();
        //Environment.Exit(1);
        qtr = one / 4;
        savec = c;
        c = SLAMC3(c, -a);
        //Console.WriteLine("c,qtr={0} {1}",c,qtr);
        slamcl_lbeta = (int)(c + qtr);
    }
}

```

```

//Console.WriteLine("lbeta={0}", slamcl_lbeta);

/*
 *      Now determine whether rounding or chopping occurs, by adding a
 *      bit less than beta/2 and a bit more than beta/2 to a.
 */

b = slamcl_lbeta;
f = SLAMC3(b / 2, -b / 100);
c = SLAMC3(f, a);

if (c == a)
{
    lrnd = true;
}
else
{
    lrnd = false;
}

f = SLAMC3(b / 2, b / 100);
c = SLAMC3(f, a);
//Console.WriteLine("{0} {1:E20} {2:E20} {3} {4:E20} {5} {6}", lrnd, c, a, qtr, savec, b, f);
//Console.ReadKey();
//    Environment.Exit(1);
//Console.WriteLine("lrnd, c, a {0} {1:E20} {2:E20}", lrnd, c, a);

if ((lrnd) & (c == a))
{
    lrnd = false;
}
//Console.WriteLine("lrnd = false");
/*
 *      Try and decide whether rounding is done in the IEEE 'round to
 *      nearest' style. B/2 is half a unit in the last place of the two
 *      numbers A and SAVEC. Furthermore, A is even, i.e. has last bit
 *      zero, and SAVEC is odd. Thus adding B/2 to A should not change
 *      A, but adding B/2 to SAVEC should change SAVEC.
 */

t1 = SLAMC3(b / 2, a);
t2 = SLAMC3(b / 2, savec);
lieceel = (t1 == a) & (t2 > savec) & lrnd;

/*
 *      Now find the mantissa, t. It should be the integer part of
 *      log to the base beta of a, however it is safer to determine t
 *      by powering. So we find t as the smallest positive integer for
 *      which
 *
 *      fl( beta**t + 1.0 ) = 1.0.
 */

slamcl_lt = 0;
a = 1.0F;
c = 1.0F;

while (c == one)
{
    slamcl_lt = slamcl_lt + 1;
    a = a * slamcl_lbeta;
    c = SLAMC3(a, one);
    c = SLAMC3(c, -a);
}

beta = slamcl_lbeta;
t = slamcl_lt;
rnd = lrnd;
ieceel = lieeel;
//Console.WriteLine("{0} {1} {2} {3}", beta, t2, rnd, iieceel);
//Console.ReadKey();
//Environment.Exit(1);
}

}

static void SLAMC2(ref int beta, ref int t, ref bool rnd, ref float eps, ref int emin, ref float rmin, ref int emax, ref float rmax)
{
    // C# LAPACK auxiliary routine (version 1.1) of the original BLAS SLAMC2
    //
    // SLAMC2 determines the machine parameters specified in its argument list.
    /*
     * Arguments
     * =====
     *
     * BETA      (output) INTEGER
     *           The base of the machine.
     *
     * T         (output) INTEGER
     *           The number of ( BETA ) digits in the mantissa.
     *
     * RND       (output) LOGICAL
     *           Specifies whether proper rounding ( RND = .TRUE. ) or
     *           chopping ( RND = .FALSE. ) occurs in addition. This may not
     *           be a reliable guide to the way in which the machine performs
     *           its arithmetic.
     *
     * EPS       (output) float PRECISION
     *           The smallest positive number such that
     *
     *           fl( 1.0 - EPS ) .LT. 1.0,
     *
     *           where fl denotes the computed value.
     *
     * EMIN      (output) INTEGER
     *           The minimum exponent before (gradual) underflow occurs.
     *
     * RMIN      (output) float PRECISION
     *           The smallest normalized number for the machine, given by
     *           BASE**( EMIN - 1 ), where BASE is the floating point value
     *           of BETA.
     *
     * EMAX      (output) INTEGER
     *           The maximum exponent before overflow occurs.
     */
}

```

```

* RMAX      (output) float PRECISION
*            The largest positive number for the machine, given by
*            BASE**EMAX * ( 1 - EPS ), where BASE is the floating point
*            value of BETA.
*
* Further Details
* =====
*
* The computation of EPS is based on a routine PARANOIA by
* W. Kahan of the University of California at Berkeley.
*
* =====
*/

// Local Scalars
//static bool first=true, iwarn=false;
bool ieee = false, lieee1 = false, lrnd = false;
//static int lbeta, lemax, lemin, lt;
int gnmin = 0, gpmin = 0, i, ngnmin = 0, ngpmin = 0;
//static float leps, lrmax, lrmin;
float a, b, c, half, one, rbase, sixth, small, third, two, zero;

string hstr = ""
WARNING. The value EMIN may be incorrect: EMIN = {0}
If, after inspection, the value EMIN looks acceptable please comment out
the IF block as marked within the code of routine SLAMC2
otherwise supply EMIN explicitly.
";

if (slamc2_first)
{
    slamc2_first = false;
    zero = 0.0F;
    one = 1.0F;
    two = 2.0F;

    /*
     * LBETA, LT, LRND, LEPS, LEMIN and LRMIN are the local values of
     * BETA, T, RND, EPS, EMIN and RMIN.
     *
     * Throughout this routine we use the function SLAMC3 to ensure
     * that relevant values are stored and not held in registers, or
     * are not affected by optimizers.
     *
     * SLAMC1 returns the parameters LBETA, LT, LRND and LIEEE1.
     */

    SLAMC1(ref slamc2_lbeta, ref slamc2_lt, ref lrnd, ref lieee1);
    //Console.WriteLine("{0} {1} {2} {3}", slamc2_lbeta, slamc2_lt, lrnd, lieee1);
    //Console.ReadKey();
    //Environment.Exit(1);

    /*
     * Start to find EPS.
     */

    b = slamc2_lbeta;
    //Console.WriteLine("b.1={0}", b);
    //Console.WriteLine("slamc2_lt={0}", slamc2_lt);
    a = (float)Math.Pow(b, -slamc2_lt);
    //Console.WriteLine("a={0}", a);
    leps = a;

    /*
     * Try some tricks to see whether or not this is the correct EPS.
     */

    b = two / 3.0F;
    //Console.WriteLine("b.2={0}", b);
    half = one / 2.0F;
    sixth = SLAMC3(b, -half);
    third = SLAMC3(sixth, sixth);
    b = SLAMC3(third, -half);
    //Console.WriteLine("b.3={0}", b);
    b = SLAMC3(b, sixth);
    //Console.WriteLine("b.4={0}", b);
    b = Math.Abs(b);
    //Console.WriteLine("b.5={0}", b);
    //Console.WriteLine("b,leps={0} {1}", b, leps);
    if (b < leps)
    {
        b = leps;
        // Console.WriteLine("b=leps {0}", leps);
    }
    leps = 1.0F;
    /*
     * Console.WriteLine("b={0}", b);
     * Console.WriteLine("half={0}", half);
     * Console.WriteLine("sixth={0}", sixth);
     * Console.WriteLine("third={0}", third);
     * Console.WriteLine("leps={0}", leps);
     * Console.ReadKey();
     * Environment.Exit(1);
     */

    while ((leps > b) & (b > zero))
    {
        leps = b;
        c = SLAMC3(half * leps, (float)(Math.Pow(two, 5) * Math.Pow(leps, 2)));
        c = SLAMC3(half, -c);
        b = SLAMC3(half, c);
        c = SLAMC3(half, -b);
        b = SLAMC3(half, c);
    }

    if (a < leps)
    {
        leps = a;
    }

    /*
     * Computation of EPS complete.
     *
     * Now find EMIN. Let A = + or - 1, and + or - (1 + BASE**(-3)).
     * Keep dividing A by BETA until (gradual) underflow occurs. This
     * is detected when we cannot recover the previous A.
     */

```

```

*/

rbase = one / slamc2_lbeta;
small = one;

for (i = 1; i <= 3; i = i + 1)
{
    small = SLAMC3(small * rbase, zero);
}

a = SLAMC3(one, small);
SLAMC4(ref ngpmin, one, slamc2_lbeta);
SLAMC4(ref ngnmin, -one, slamc2_lbeta);
SLAMC4(ref gpmin, a, slamc2_lbeta);
SLAMC4(ref gnmin, -a, slamc2_lbeta);
ieee = false;

if ((ngpmin == ngnmin) & (gpmin == gnmin))
{
    if (ngpmin == gpmin)
    {
        lemin = ngpmin;

        /*      ( Non twos-complement machines, no gradual underflow;
         *      e.g., VAX )
        */

    }
    else if ((gpmin - ngpmin) == 3)
    {
        lemin = ngpmin - 1 + slamc2_lt;
        ieee = true;

        /*      ( Non twos-complement machines, with gradual underflow;
         *      e.g., IEEE standard followers )
        */

    }
    else
    {
        lemin = Math.Min(ngpmin, gpmin);
        /*      ( A guess; no known machine )
        */
        iwarn = true;
    }
}
else if ((ngpmin == gpmin) & (ngnmin == gnmin))
{
    if (Math.Abs(ngpmin - ngnmin) == 1)
    {
        lemin = Math.Max(ngpmin, ngnmin);

        /*      ( Twos-complement machines, no gradual underflow;
         *      e.g., CYBER 205 )
        */

    }
    else
    {
        lemin = Math.Min(ngpmin, ngnmin);
        /*      ( A guess; no known machine )
        */
        iwarn = true;
    }
}
else if ((Math.Abs(ngpmin - ngnmin) == 1) & (gpmin == gnmin))
{
    if ((gpmin - Math.Min(ngpmin, ngnmin)) == 3)
    {
        lemin = Math.Max(ngpmin, ngnmin) - 1 + slamc2_lt;

        /*      ( Twos-complement machines with gradual underflow;
         *      no known machine )
        */

    }
    else
    {
        lemin = Math.Min(ngpmin, ngnmin);
        /*      ( A guess; no known machine )
        */
        iwarn = true;
    }
}
else
{
    lemin = Math.Min(Math.Min(Math.Min(ngpmin, ngnmin), gpmin), gnmin);
    /*      ( A guess; no known machine )
    */
    iwarn = true;
}

/* Comment out this if block if EMIN is ok
if (iwarn)
{
    slamc2_first = true;
    Console.WriteLine(hstr, lemin);
}
/* Comment out this if block if EMIN is ok

/*      Assume IEEE arithmetic if we found denormalised numbers above,
 *      or if arithmetic seems to round in the IEEE style, determined
 *      in routine SLAMC1. A true IEEE machine should have both things
 *      true; however, faulty machines may have one or the other.
 */

ieee = ieee | ieee1;

/*
 *      Compute RMIN by successive division by BETA. We could compute
 *      RMIN as BASE**( EMIN - 1 ), but some machines underflow during
 *      this computation.
 */

lrmin = 1.0F;
for (i = 1; i <= 1 - lemin; i = i + 1)
{
    lrmin = SLAMC3(lrmin * rbase, zero);
}

```

```

/*
 *      Finally, call SLAMC5 to compute EMAX and RMAX.
 */

SLAMC5(slamc2_lbeta, slmc2_lt, lemin, ieee, ref lemax, ref lrmx);
}

beta = slmc2_lbeta;
t = slmc2_lt;
rnd = lrnd;
eps = leps;
emin = lemin;
rmin = lrmin;
emax = lemax;
rmax = lrmx;
/*
    Console.WriteLine("beta={0}",beta);
    Console.WriteLine("t={0}",t);
    Console.WriteLine("rnd={0}",rnd);
    Console.WriteLine("eps={0}",eps);
    Console.WriteLine("emin={0}",emin);
    Console.WriteLine("rmin={0}",rmin);
    Console.WriteLine("emax={0}",emax);
    Console.WriteLine("rmax={0}",rmax);

    Console.ReadKey();
    Environment.Exit(1);
*/
}

static float SLAMC3(float a, float b)
{
    // C# LAPACK auxiliary routine (version 1.1) of original BLAS SLAMC3
    //
    /* SLAMC3 is intended to force A and B to be stored prior to doing
     * the addition of A and B, for use in situations where optimizers
     * might hold one of these in a register.
     */
    /*
     * Arguments
     * =====
     *
     * A, B      (input) float PRECISION
     *           The values A and B.
     *
     * =====
     */
    return (a + b);
}

static void SLAMC4(ref int emin, float start, int _base)
{
    // C# LAPACK auxiliary routine (version 1.1) of original BLAS SLAMC4
    //
    // SLAMC4 is a service routine for SLAMC2.
    /*
     * Arguments
     * =====
     *
     * EMIN      (output) EMIN
     *           The minimum exponent before (gradual) underflow, computed by
     *           setting A = START and dividing by BASE until the previous A
     *           can not be recovered.
     *
     * START     (input) float PRECISION
     *           The starting point for determining EMIN.
     *
     * BASE      (input) INTEGER
     *           The base of the machine.
     *
     * =====
     */

    // Local Scalars
    int i;
    float a, b1, b2, c1, c2, d1, d2, one, rbase, zero;

    a = start;
    one = 1.0F;
    rbase = one / _base;
    zero = 0.0F;
    emin = 1;
    b1 = SLAMC3(a * rbase, zero);
    c1 = a;
    d1 = a;
    c2 = a;
    d2 = a;

    while ((c1 == a) & (c2 == a) & (d1 == a) & (d2 == a))
    {
        emin = emin - 1;
        a = b1;
        b1 = SLAMC3(a / _base, zero);
        c1 = SLAMC3(b1 * _base, zero);
        d1 = zero;

        for (i = 1; i <= _base; i = i + 1)
        {
            d1 = d1 + b1;
        }

        b2 = SLAMC3(a * rbase, zero);
        c2 = SLAMC3(b2 / rbase, zero);
        d2 = zero;

        for (i = 1; i <= _base; i = i + 1)
        {
            d2 = d2 + b2;
        }
    }
}

static void SLAMC5(int beta, int p, int emin, bool ieee, ref int emax, ref float rmax)
{
    // C# LAPACK auxiliary routine (version 1.1) of original BLAS SLAMC5
    //
    /* SLAMC5 attempts to compute RMAX, the largest machine floating-point

```

```

* number, without overflow. It assumes that EMAX + abs(EMIN) sum
* approximately to a power of 2. It will fail on machines where this
* assumption does not hold, for example, the Cyber 205 (EMIN = -28625,
* EMAX = 28718). It will also fail if the value supplied for EMIN is
* too large (i.e. too close to zero), probably with overflow.
*/
/* Arguments
* =====
* BETA      (input) INTEGER
*           The base of floating-point arithmetic.
*
* P         (input) INTEGER
*           The number of base BETA digits in the mantissa of a
*           floating-point value.
*
* EMIN      (input) INTEGER
*           The minimum exponent before (gradual) underflow.
*
* IEEE      (input) LOGICAL
*           A logical flag specifying whether or not the arithmetic
*           system is thought to comply with the IEEE standard.
*
* EMAX      (output) INTEGER
*           The largest exponent before overflow
*
* RMAX      (output) float PRECISION
*           The largest machine floating-point number.
*
* =====
*/

// Parameters
const float zero = 0.0F, one = 1.0F;

// Local Scalars
int exbits, expsum, i, lexp, nbits, _try, uexp;
float oldy = 0.0F, recbas, y, z;

/*      First compute LEXP and UEXP, two powers of 2 that bound
*      abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
*      approximately to the bound that is closest to abs(EMIN).
*      (EMAX is the exponent of the required number RMAX).
*/

lexp = 1;
exbits = 1;

/*      10 CONTINUE
*      TRY = LEXP*2
*      IF( TRY.LE.( -EMIN ) ) THEN
*          LEXP = TRY
*          EXBITS = EXBITS + 1
*      GO TO 10
*      END IF
*/

_try = lexp * 2;
while (_try <= (-emin))
{
    lexp = _try;
    exbits = exbits + 1;
    _try = lexp * 2;
}

if (lexp == -emin)
{
    uexp = lexp;
}
else
{
    uexp = _try;
    exbits = exbits + 1;
}

/*      Now -LEXP is less than or equal to EMIN, and -UEXP is greater
*      than or equal to EMIN. EXBITS is the number of bits needed to
*      store the exponent.
*/

if ((uexp + emin) > (-lexp - emin))
{
    expsum = 2 * lexp;
}
else
{
    expsum = 2 * uexp;
}

/*      EXPSUM is the exponent range, approximately equal to
*      EMAX - EMIN + 1 .
*/

emax = expsum + emin - 1;
nbits = 1 + exbits + p;

/*      NBITS is the total number of bits needed to store a
*      floating-point number.
*/

if (((nbits % 2) == 1) & (beta == 2))
{
    /*      Either there are an odd number of bits used to store a
    *      floating-point number, which is unlikely, or some bits are
    *      not used in the representation of numbers, which is possible,
    *      (e.g. Cray machines) or the mantissa has an implicit bit,
    *      (e.g. IEEE machines, Dec Vax machines), which is perhaps the
    *      most likely. We have to assume the last alternative.
    *      If this is true, then we need to reduce EMAX by one because
    *      there must be some way of representing zero in an implicit-bit
    *      system. On machines like Cray, we are reducing EMAX by one
    *      unnecessarily.
    */
}

```

```

    emax = emax - 1;
}

if (ieee)
{
    /*      Assume we are on an IEEE machine which reserves one exponent
     *      for infinity and NaN.
     */
    emax = emax - 1;
}

/*      Now create RMAX, the largest machine number, which should
 *      be equal to (1.0 - BETA**(-P)) * BETA**EMAX .
 *
 *      First compute 1.0 - BETA**(-P), being careful that the
 *      result is less than 1.0 .
 */

recbas = one / beta;
z = beta - one;
y = zero;
for (i = 1; i <= p; i = i + 1)
{
    z = z * recbas;
    if (y < one)
    {
        oldy = y;
    }
    y = SLAMC3(y, z);
}

if (y >= one)
{
    y = oldy;
}

/*      Now multiply by BETA**EMAX to get RMAX.
 */

for (i = 1; i <= emax; i = i + 1)
{
    y = SLAMC3(y * beta, zero);
}

rmax = y;
}
//
public static long SECOND()
{
    // C# of LAPACK auxiliary routine DSECDN returns the user time for a process in seconds.
    //long gg = DateTimeOffset.UtcNow.UtcTicks;

    return DateTimeOffset.UtcNow.UtcTicks;
}
public static void SECONDTSTError! Bookmark not defined.()
{
    // C# of LAPACK test routine
    // Parameters
    const int nmax = 1000;
    const int its = 50000;

    // Local Scalars
    int i, j;
    float alpha, avg, tnosec, total;
    long t1, t2;
    float tnosec2;
    float tnosecm, tnosecm2m, avgm;

    // Local Arrays
    float[] x = new float[nmax];
    float[] y = new float[nmax];
    //Console.WriteLine("{0}", y.Length);
    //return;
    // Figure TOTAL flops ..
    total = (float)nmax * (float)its * 2.0F;

    // Initialize X and Y
    for (i = 0; i < nmax; i = i + 1)
    {
        //Console.WriteLine("i={0}", i);
        x[i] = 1.0F / ((float)i);
        y[i] = (float)(nmax - i) / ((float)nmax);
    }
    alpha = 0.315F;

    // Time TOTAL SAXPY operations
    t1 = SECOND();

    for (j = 0; j < its; j = j + 1)
    {
        for (i = 0; i < nmax; i = i + 1)
        {
            y[i] = y[i] + alpha * x[i];
        }
        alpha = -alpha;
    }

    t2 = SECOND();
    tnosec = (float)(new TimeSpan(t2 - t1)).TotalSeconds;
    //tnosecm = (float)(new TimeSpan(t2 - t1)).TotalMilliseconds;
    //Console.WriteLine("{0} {1} {2}", t1,t2,new TimeSpan(t2-t1).Ticks);
    Console.WriteLine("Time for {0,10:G3} SAXPY ops = {1,10:G3} seconds", total, tnosec);
    //Console.WriteLine("Time for {0,10:G3} SAXPY ops = {1,10:G3} milliseconds", total, tnosecm);
    if (tnosec > 0.0)
    //if (tnosecm > 0.0)
    {
        Console.WriteLine("SAXPY performance rate      = {0,10:G3} mflops", (total / 1.0e6) / tnosec);
        //Console.WriteLine("SAXPY performance rate      = {0,10:G3} mflops", (total / 1.0e3) / tnosecm);
    }
    else
    {

```

```

        Console.WriteLine("*** Warning: Time for operations was less or equal than zero => timing in TESTING might be dubious");
    }

    // Time TOTAL SAXPY operations with DSECND in the outer loop
    t1 = SECOND();

    for (j = 0; j < its; j = j + 1)
    {
        for (i = 0; i < nmax; i = i + 1)
        {
            y[i] = y[i] + alpha * x[i];
        }
        alpha = -alpha;
        t2 = SECOND();
    }

    tnosec2 = (float)(new TimeSpan(t2 - t1)).TotalSeconds;
    ///==tnosec2m = (float)(new TimeSpan(t2 - t1)).TotalMilliseconds;
    // Compute the time used in milliseconds used by an average call to DSECND.
    Console.WriteLine("Including SECOND, time = {0,10:G3} seconds", tnosec2);
    //Console.WriteLine("Including SECOND, time = {0,10:G3} milliseconds", tnosec2m);
    avg = (tnosec2 - tnosec) * 1000.0F / ((float)its);
    //avgm = (tnosec2m - tnosecm) / ((float)its);
    if (avg > 0.0)
    //if (avgm > 0.0)
    {
        Console.WriteLine("Average time for SECOND = {0,10:G3} milliseconds", avg);
        //Console.WriteLine("Average time for SECOND = {0,10:G3} milliseconds", avgm);
    }

    // Compute the equivalent number of floating point operations used by an average call to DSECND.

    if ((avg > 0.0) & (tnosec > 0.0))
    //if ((avgm > 0.0) & (tnosecm > 0.0))
    {
        Console.WriteLine("Equivalent floating point ops = {0,10:G3} ops", (avg / 1000) * total / tnosec);
        //Console.WriteLine("Equivalent floating point ops = {0,10:G3} ops", (avgm) * total / tnosecm);
    }
    MYSUB(nmax, x, y);
}

public static void MYSUB(int nmax, float[] x, float[] y)
{
    return;
}
}
}

```

APPENDIX A.2 – DBLAT1.f90

```

program testdblat1
implicit none

! Variables
CHARACTER :: kin
DOUBLE PRECISION X(9), Y(5)
print *, ' Epsilon                = ',DLAMCH('E')
print *, ' Safe minimum           = ',DLAMCH('S')
print *, ' Base                   = ',DLAMCH('B')
print *, ' Precision              = ',DLAMCH('P')
print *, ' Number of digits in mantissa = ',DLAMCH('N')
print *, ' Rounding mode          = ',DLAMCH('R')
print *, ' Minimum exponent       = ',DLAMCH('M')
print *, ' Underflow threshold    = ',DLAMCH('U')
print *, ' Largest exponent       = ',DLAMCH('L')
print *, ' Overflow threshold     = ',DLAMCH('O')
print *, ' Reciprocal of safe minimum = ',1/DLAMCH('S')
print *, '=====
print *, ' Epsilon                = ',SLAMCH('E')
print *, ' Safe minimum           = ',SLAMCH('S')
print *, ' Base                   = ',SLAMCH('B')
print *, ' Precision              = ',SLAMCH('P')
print *, ' Number of digits in mantissa = ',SLAMCH('N')
print *, ' Rounding mode          = ',SLAMCH('R')
print *, ' Minimum exponent       = ',SLAMCH('M')
print *, ' Underflow threshold    = ',SLAMCH('U')
print *, ' Largest exponent       = ',SLAMCH('L')
print *, ' Overflow threshold     = ',SLAMCH('O')
print *, ' Reciprocal of safe minimum = ',1/SLAMCH('S')
print *, '=====

CALL DSECDTST()
print *, '=====
CALL SECDTST()
print *, '=====
CALL TSTIEE()
print *, 'TSTIEE Done.'

CALL DBLAT1
print *, 'DBLAT1 Done.'

!https://software.intel.com/en-us/node/522288
DO 10 I=1,5
  X((I-1)*ABS(2)+1) = 2.0D0
  Y((I-1)*ABS(1)+1) = 1.0D0
10 CONTINUE
print *, DDOT(6,X,2,Y,1)
read(*,'(A1)',kin)
STOP

!
! Specifies the value to be returned by DLAMCH:
!
! = 'E' or 'e', DLAMCH := eps
! = 'S' or 's', DLAMCH := sfmin
! = 'B' or 'b', DLAMCH := base
! = 'P' or 'p', DLAMCH := eps*base
! = 'N' or 'n', DLAMCH := t
! = 'R' or 'r', DLAMCH := rnd
! = 'M' or 'm', DLAMCH := emin
! = 'U' or 'u', DLAMCH := rmin
! = 'L' or 'l', DLAMCH := emax
! = 'O' or 'o', DLAMCH := rmax
!
! where
!
! eps = relative machine precision
! sfmin = safe minimum, such that 1/sfmin does not overflow
! base = base of the machine
! prec = eps*base
! t = number of (base) digits in the mantissa
! rnd = 1.0 when rounding occurs in addition, 0.0 otherwise
! emin = minimum exponent before (gradual) underflow
! rmin = underflow threshold - base**(emin-1)
! emax = largest exponent before overflow
! rmax = overflow threshold - (base**emax)*(1-eps)
CONTAINS

LOGICAL FUNCTION LSAME(CA,CB)
!
! -- LAPACK auxiliary routine (version 3.1) --
! Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
! November 2006
!
! .. Scalar Arguments ..
CHARACTER CA,CB
!
! ..
!
! Purpose
! =====
!
! LSAME returns .TRUE. if CA is the same letter as CB regardless of
! case.
!
! Arguments
! =====
!
! CA (input) CHARACTER*1
!
! CB (input) CHARACTER*1
! CA and CB specify the single characters to be compared.
!
! =====
!
! .. Intrinsic Functions ..
INTRINSIC ICHAR
!
! ..
! .. Local Scalars ..
INTEGER INTA,INTB,ZCODE
!
! ..
!
! Test if the characters are equal
!
!

```

```

LSAME = CA .EQ. CB
IF (LSAME) RETURN
!
! Now test for equivalence if both characters are alphabetic.
!
ZCODE = ICHAR('Z')
!
! Use 'Z' rather than 'A' so that ASCII can be detected on Prime
! machines, on which ICHAR returns a value with bit 8 set.
! ICHAR('A') on Prime machines returns 193 which is the same as
! ICHAR('A') on an EBCDIC machine.
!
INTA = ICHAR(CA)
INTB = ICHAR(CB)
!
IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
!
! ASCII is assumed - ZCODE is the ASCII code of either lower or
! upper case 'Z'.
!
IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
!
ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN
!
! EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or
! upper case 'Z'.
!
IF (INTA.GE.129 .AND. INTA.LE.137 .OR. &
INTA.GE.145 .AND. INTA.LE.153 .OR. &
INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
IF (INTB.GE.129 .AND. INTB.LE.137 .OR. &
INTB.GE.145 .AND. INTB.LE.153 .OR. &
INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
!
ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
!
! ASCII is assumed, on Prime machines - ZCODE is the ASCII code
! plus 128 of either lower or upper case 'Z'.
!
IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
END IF
LSAME = INTA .EQ. INTB
!
RETURN
!
End of LSAME
!
END FUNCTION

DOUBLE PRECISION FUNCTION DLAMCH( CMACH )
!!!DEC$ ATTRIBUTES DLLEXPORT :: DLAMCH_1

!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
CHARACTER CMACH
..
!
! Purpose
! =====
!
! DLAMCH determines double precision machine parameters.
!
! Arguments
! =====
!
! CMACH (input) CHARACTER*1
! Specifies the value to be returned by DLAMCH:
! = 'E' or 'e', DLAMCH := eps
! = 'S' or 's', DLAMCH := sfmin
! = 'B' or 'b', DLAMCH := base
! = 'P' or 'p', DLAMCH := eps*base
! = 'N' or 'n', DLAMCH := t
! = 'R' or 'r', DLAMCH := rnd
! = 'M' or 'm', DLAMCH := emin
! = 'U' or 'u', DLAMCH := rmin
! = 'L' or 'l', DLAMCH := emax
! = 'O' or 'o', DLAMCH := rmax
!
! where
!
! eps = relative machine precision
! sfmin = safe minimum, such that 1/sfmin does not overflow
! base = base of the machine
! prec = eps*base
! t = number of (base) digits in the mantissa
! rnd = 1.0 when rounding occurs in addition, 0.0 otherwise
! emin = minimum exponent before (gradual) underflow
! rmin = underflow threshold - base**(emin-1)
! emax = largest exponent before overflow
! rmax = overflow threshold - (base**emax)*(1-eps)
!
! =====
!
! .. Parameters ..
DOUBLE PRECISION ONE, ZERO
PARAMETER ( ONE = 1.0D+0, ZERO = 0.0D+0 )
!
! ..
! .. Local Scalars ..
LOGICAL FIRST, LRND
INTEGER BETA, IMAX, IMIN, IT
DOUBLE PRECISION BASE, EMAX, EMIN, EPS, PREC, RMACH, RMAX, RMIN,
!
! ..
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..

```

```

!      EXTERNAL          DLAMC2
!      ..
!      .. Save statement ..
!      SAVE              FIRST, EPS, SFMIN, BASE, T, RND, EMIN, RMIN, EMAX, RMAX, PREC
!      ..
!      .. Data statements ..
!      DATA             FIRST / .TRUE. /
!      ..
!      .. Executable Statements ..
!
!      IF( FIRST ) THEN
!          FIRST = .FALSE.
!          CALL DLAMC2( BETA, IT, LRND, EPS, IMIN, RMIN, IMAX, RMAX )
!          print *,BETA,IT,LRND,EPS,IMIN,RMIN,IMAX,RMAX
!          STOP
!          BASE = BETA
!          T = IT
!          IF( LRND ) THEN
!              RND = ONE
!              EPS = ( BASE** ( 1-IT ) ) / 2
!          ELSE
!              RND = ZERO
!              EPS = BASE** ( 1-IT )
!          END IF
!          PREC = EPS*BASE
!          EMIN = IMIN
!          EMAX = IMAX
!          SFMIN = RMIN
!          SMALL = ONE / RMAX
!          IF( SMALL.GE.SFMIN ) THEN
!
!              Use SMALL plus a bit, to avoid the possibility of rounding
!              causing overflow when computing 1/sfmin.
!
!              SFMIN = SMALL*( ONE+EPS )
!          END IF
!      END IF
!
!      IF( LSAME( CMACH, 'E' ) ) THEN
!          RMACH = EPS
!      ELSE IF( LSAME( CMACH, 'S' ) ) THEN
!          RMACH = SFMIN
!      ELSE IF( LSAME( CMACH, 'B' ) ) THEN
!          RMACH = BASE
!      ELSE IF( LSAME( CMACH, 'P' ) ) THEN
!          RMACH = PREC
!      ELSE IF( LSAME( CMACH, 'N' ) ) THEN
!          RMACH = T
!      ELSE IF( LSAME( CMACH, 'R' ) ) THEN
!          RMACH = RND
!      ELSE IF( LSAME( CMACH, 'M' ) ) THEN
!          RMACH = EMIN
!      ELSE IF( LSAME( CMACH, 'U' ) ) THEN
!          RMACH = RMIN
!      ELSE IF( LSAME( CMACH, 'L' ) ) THEN
!          RMACH = EMAX
!      ELSE IF( LSAME( CMACH, 'O' ) ) THEN
!          RMACH = RMAX
!      END IF
!
!      DLAMCH = RMACH
!      RETURN
!
!      End of DLAMCH
!
!      END FUNCTION
!
!*****
!
!      SUBROUTINE DLAMC1( BETA, T, RND, IEEE1 )
!
!      -- LAPACK auxiliary routine (version 1.1) --
!      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
!      Courant Institute, Argonne National Lab, and Rice University
!      October 31, 1992
!
!      .. Scalar Arguments ..
!      LOGICAL          IEEE1, RND
!      INTEGER           BETA, T
!      ..
!
!      Purpose
!      =====
!
!      DLAMC1 determines the machine parameters given by BETA, T, RND, and
!      IEEE1.
!
!      Arguments
!      =====
!
!      BETA      (output) INTEGER
!                  The base of the machine.
!
!      T         (output) INTEGER
!                  The number of ( BETA ) digits in the mantissa.
!
!      RND       (output) LOGICAL
!                  Specifies whether proper rounding ( RND = .TRUE. ) or
!                  chopping ( RND = .FALSE. ) occurs in addition. This may not
!                  be a reliable guide to the way in which the machine performs
!                  its arithmetic.
!
!      IEEE1     (output) LOGICAL
!                  Specifies whether rounding appears to be done in the IEEE
!                  'round to nearest' style.
!
!      Further Details
!      =====
!
!      The routine is based on the routine ENVIRON by Malcolm and
!      incorporates suggestions by Gentleman and Marovich. See
!
!      Malcolm M. A. (1972) Algorithms to reveal properties of

```

```

! floating-point arithmetic. Comms. of the ACM, 15, 949-951.
!
! Gentleman W. M. and Marovich S. B. (1974) More on algorithms
! that reveal properties of floating point arithmetic units.
! Comms. of the ACM, 17, 276-277.
!
! =====
!
! .. Local Scalars ..
LOGICAL FIRST, LIEEE1, LRND
INTEGER LBETA, LT
DOUBLE PRECISION A, B, C, F, ONE, QTR, SAVEC, T1, T2
!
! .. External Functions ..
DOUBLE PRECISION DLAMC3
EXTERNAL DLAMC3
!
! .. Save statement ..
SAVE FIRST, LIEEE1, LBETA, LRND, LT
!
! .. Data statements ..
DATA FIRST / .TRUE. /
!
! .. Executable Statements ..
!
IF( FIRST ) THEN
    FIRST = .FALSE.
    ONE = 1
!
! LBETA, LIEEE1, LT and LRND are the local values of BETA,
! LIEEE1, T and RND.
!
! Throughout this routine we use the function DLAMC3 to ensure
! that relevant values are stored and not held in registers, or
! are not affected by optimizers.
!
! Compute a = 2.0**m with the smallest positive integer m such
! that
!
!     fl( a + 1.0 ) = a.
!
    A = 1
    C = 1
!
!+ 10 WHILE( C.EQ.ONE )LOOP
    CONTINUE
    IF( C.EQ.ONE ) THEN
        A = 2*A
        C = DLAMC3( A, ONE )
        C = DLAMC3( C, -A )
        GO TO 10
    END IF
END WHILE
!
! Now compute b = 2.0**m with the smallest positive integer m
! such that
!
!     fl( a + b ) .gt. a.
!
    B = 1
    C = DLAMC3( A, B )
!
!+ 20 WHILE( C.EQ.A )LOOP
    CONTINUE
    IF( C.EQ.A ) THEN
        B = 2*B
        C = DLAMC3( A, B )
        GO TO 20
    END IF
END WHILE
!
! Now compute the base. a and c are neighbouring floating point
! numbers in the interval ( beta**t, beta**( t + 1 ) ) and so
! their difference is beta. Adding 0.25 to c is to ensure that it
! is truncated to beta and not ( beta - 1 ).
!
!     print *,A,B,C
!
    QTR = ONE / 4
    SAVEC = C
    C = DLAMC3( C, -A )
!
!     print *, 'C,QTR=',C,QTR
    LBETA = C + QTR
!
!     print *, 'LBETA=',LBETA
!
! Now determine whether rounding or chopping occurs, by adding a
! bit less than beta/2 and a bit more than beta/2 to a.
!
    B = LBETA
    F = DLAMC3( B / 2, -B / 100 )
    C = DLAMC3( F, A )
    IF( C.EQ.A ) THEN
        LRND = .TRUE.
    ELSE
        LRND = .FALSE.
    END IF
    F = DLAMC3( B / 2, B / 100 )
    C = DLAMC3( F, A )
!
!     print *, LRND, C,A,QTR,SAVEC,B,F
!
!     print *, 'C,A',C,A
    IF( ( LRND ) .AND. ( C.EQ.A ) ) LRND = .FALSE.
!
!     print *, 'LRND = FALSE'
!
! Try and decide whether rounding is done in the IEEE 'round to
! nearest' style. B/2 is half a unit in the last place of the two
! numbers A and SAVEC. Furthermore, A is even, i.e. has last bit
! zero, and SAVEC is odd. Thus adding B/2 to A should not change
! A, but adding B/2 to SAVEC should change SAVEC.
!
    T1 = DLAMC3( B / 2, A )
    T2 = DLAMC3( B / 2, SAVEC )
    LIEEE1 = ( T1.EQ.A ) .AND. ( T2.GT.SAVEC ) .AND. LRND
!

```

```

!      Now find the mantissa, t. It should be the integer part of
!      log to the base beta of a, however it is safer to determine t
!      by powering. So we find t as the smallest positive integer for
!      which
!
!      fl( beta**t + 1.0 ) = 1.0.
!
!      LT = 0
!      A = 1
!      C = 1
!
!+  WHILE( C.EQ.ONE )LOOP
30  CONTINUE
!      IF( C.EQ.ONE ) THEN
!          LT = LT + 1
!          A = A*LBETA
!          C = DLAMC3( A, ONE )
!          C = DLAMC3( C, -A )
!          GO TO 30
!      END IF
!+  END WHILE
!
!      END IF
!
!      BETA = LBETA
!      T = LT
!      RND = LRND
!      IEEE1 = LIEEE1
!      print *,BETA,T,RND,IEEE1
!      STOP
!      RETURN
!
!      End of DLAMC1
!
!      END SUBROUTINE
!
!+*****
!
!      SUBROUTINE DLAMC2( BETA, T, RND, EPS, EMIN, RMIN, EMAX, RMAX )
!
!      -- LAPACK auxiliary routine (version 1.1) --
!      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
!      Courant Institute, Argonne National Lab, and Rice University
!      October 31, 1992
!
!      .. Scalar Arguments ..
!      LOGICAL      RND
!      INTEGER      BETA, EMAX, EMIN, T
!      DOUBLE PRECISION  EPS, RMAX, RMIN
!      ..
!
!      Purpose
!      =====
!
!      DLAMC2 determines the machine parameters specified in its argument
!      list.
!
!      Arguments
!      =====
!
!      BETA      (output) INTEGER
!                  The base of the machine.
!
!      T         (output) INTEGER
!                  The number of ( BETA ) digits in the mantissa.
!
!      RND       (output) LOGICAL
!                  Specifies whether proper rounding ( RND = .TRUE. ) or
!                  chopping ( RND = .FALSE. ) occurs in addition. This may not
!                  be a reliable guide to the way in which the machine performs
!                  its arithmetic.
!
!      EPS       (output) DOUBLE PRECISION
!                  The smallest positive number such that
!
!                  fl( 1.0 - EPS ) .LT. 1.0,
!
!                  where fl denotes the computed value.
!
!      EMIN      (output) INTEGER
!                  The minimum exponent before (gradual) underflow occurs.
!
!      RMIN      (output) DOUBLE PRECISION
!                  The smallest normalized number for the machine, given by
!                  BASE** ( EMIN - 1 ), where BASE is the floating point value
!                  of BETA.
!
!      EMAX      (output) INTEGER
!                  The maximum exponent before overflow occurs.
!
!      RMAX      (output) DOUBLE PRECISION
!                  The largest positive number for the machine, given by
!                  BASE**EMAX * ( 1 - EPS ), where BASE is the floating point
!                  value of BETA.
!
!      Further Details
!      =====
!
!      The computation of EPS is based on a routine PARANOIA by
!      W. Kahan of the University of California at Berkeley.
!
!+*****
!
!      .. Local Scalars ..
!      LOGICAL      FIRST, IEEE, IWARN, LIEEE1, LRND
!      INTEGER      GNMIN, GPMIN, I, LBETA, LEMAX, LEMIN, LT,
!      DOUBLE PRECISION  A, B, C, HALF, LEPS, LRMAX, LRMIN, ONE, RBASE,
!
!      ..
!
!      .. External Functions ..
!      DOUBLE PRECISION  DLAMC3
!      EXTERNAL          DLAMC3
!
!      ..
!
!      .. External Subroutines ..
!      EXTERNAL          DLAMC1, DLAMC4, DLAMC5

```

```

! ..
! .. Intrinsic Functions ..
INTRINSIC          ABS, MAX, MIN
! ..
! .. Save statement ..
SAVE              FIRST, IWARN, LBETA, LEMAX, LEMIN, LEPS, LRMAX,          LRMIN, LT
! ..
! .. Data statements ..
DATA              FIRST / .TRUE. / , IWARN / .FALSE. /
! ..
! .. Executable Statements ..
!
IF( FIRST ) THEN
    FIRST = .FALSE.
    ZERO = 0
    ONE = 1
    TWO = 2
!
!   LBETA, LT, LRND, LEPS, LEMIN and LRMIN are the local values of
!   BETA, T, RND, EPS, EMIN and RMIN.
!
!   Throughout this routine we use the function DLAMC3 to ensure
!   that relevant values are stored and not held in registers, or
!   are not affected by optimizers.
!
!   DLAMC1 returns the parameters LBETA, LT, LRND and LIEEE1.
!
CALL DLAMC1( LBETA, LT, LRND, LIEEE1 )
    print *,LBETA,LT,LRND,LIEEE1
    STOP
!
!   Start to find EPS.
!
B = LBETA
    print *, 'B.1=', B
    print *, 'LT=', LT
A = B**(-LT)
    print *, 'A=', A
LEPS = A
!
!   Try some tricks to see whether or not this is the correct EPS.
!
B = TWO / 3
    print *, 'B.2=', B
HALF = ONE / 2
SIXTH = DLAMC3( B, -HALF )
THIRD = DLAMC3( SIXTH, SIXTH )
B = DLAMC3( THIRD, -HALF )
    print *, 'B.3=', B
B = DLAMC3( B, SIXTH )
    print *, 'B.4=', B
B = ABS( B )
    print *, 'B.5=', B
    print *, 'B,LEPS=', B, LEPS
IF( B.LT.LEPS )      B = LEPS
IF( B.LT.LEPS ) print *, 'B = LEPS', LEPS
!
LEPS = 1
    print *, 'B=', B
    print *, 'HALF=', HALF
    print *, 'SIXTH=', SIXTH
    print *, 'THIRD=', THIRD
    print *, 'LEPS=', LEPS
!
STOP
!
!+
10  WHILE( ( LEPS.GT.B ) .AND. ( B.GT.ZERO ) ) LOOP
    CONTINUE
    IF( ( LEPS.GT.B ) .AND. ( B.GT.ZERO ) ) THEN
        LEPS = B
        C = DLAMC3( HALF*LEPS, ( TWO**5 )*( LEPS**2 ) )
        C = DLAMC3( HALF, -C )
        B = DLAMC3( HALF, C )
        C = DLAMC3( HALF, -B )
        B = DLAMC3( HALF, C )
        GO TO 10
    END IF
END WHILE
!
IF( A.LT.LEPS )      LEPS = A
!
!   Computation of EPS complete.
!
!   Now find EMIN. Let A = + or - 1, and + or - (1 + BASE**(-3)).
!   Keep dividing A by BETA until (gradual) underflow occurs. This
!   is detected when we cannot recover the previous A.
!
RBASE = ONE / LBETA
SMALL = ONE
DO 20 I = 1, 3
    SMALL = DLAMC3( SMALL*RBASE, ZERO )
20  CONTINUE
A = DLAMC3( ONE, SMALL )
CALL DLAMC4( NGPMIN, ONE, LBETA )
CALL DLAMC4( NGNMIN, -ONE, LBETA )
CALL DLAMC4( GPMIN, A, LBETA )
CALL DLAMC4( GNMIN, -A, LBETA )
IEEE = .FALSE.
!
IF( ( NGPMIN.EQ.NGNMIN ) .AND. ( GPMIN.EQ.GNMIN ) ) THEN
    IF( NGPMIN.EQ.GPMIN ) THEN
        LEMIN = NGPMIN
!       ( Non twos-complement machines, no gradual underflow;
!       e.g., VAX )
    ELSE IF( ( GPMIN-NGPMIN ).EQ.3 ) THEN
        LEMIN = NGPMIN - 1 + LT
        IEEE = .TRUE.
!       ( Non twos-complement machines, with gradual underflow;
!       e.g., IEEE standard followers )
    ELSE
        LEMIN = MIN( NGPMIN, GPMIN )
!       ( A guess; no known machine )
        IWARN = .TRUE.
    END IF

```

```

!
      ELSE IF( ( NGPMIN.EQ.GPMIN ) .AND. ( NGNMIN.EQ.GNMIN ) ) THEN
        IF( ABS( NGPMIN-NGNMIN ).EQ.1 ) THEN
          LEMIN = MAX( NGPMIN, NGNMIN )
          ! ( Twos-complement machines, no gradual underflow;
          ! e.g., CYBER 205 )
        ELSE
          LEMIN = MIN( NGPMIN, NGNMIN )
          ! ( A guess; no known machine )
          IWARN = .TRUE.
        END IF
      ELSE IF( ( ABS( NGPMIN-NGNMIN ).EQ.1 ) .AND. ( GPMIN.EQ.GNMIN ) ) THEN
        IF( ( GPMIN-MIN( NGPMIN, NGNMIN ) ).EQ.3 ) THEN
          LEMIN = MAX( NGPMIN, NGNMIN ) - 1 + LT
          ! ( Twos-complement machines with gradual underflow;
          ! no known machine )
        ELSE
          LEMIN = MIN( NGPMIN, NGNMIN )
          ! ( A guess; no known machine )
          IWARN = .TRUE.
        END IF
      ELSE
        LEMIN = MIN( NGPMIN, NGNMIN, GPMIN, GNMIN )
        ! ( A guess; no known machine )
        IWARN = .TRUE.
      END IF
! **
! Comment out this if block if EMIN is ok
      IF( IWARN ) THEN
        FIRST = .TRUE.
        WRITE( 6, FMT = 9999 ) LEMIN
      END IF
! **
!
! Assume IEEE arithmetic if we found denormalised numbers above,
! or if arithmetic seems to round in the IEEE style, determined
! in routine DLAMC1. A true IEEE machine should have both things
! true; however, faulty machines may have one or the other.
!
      IEEE = IEEE .OR. LIEEE1
!
! Compute RMIN by successive division by BETA. We could compute
! RMIN as BASE**(( EMIN - 1 )), but some machines underflow during
! this computation.
!
      LRMIN = 1
      DO 30 I = 1, 1 - LEMIN
        LRMIN = DLAMC3( LRMIN*RBASE, ZERO )
      30 CONTINUE
!
! Finally, call DLAMC5 to compute EMAX and RMAX.
!
      CALL DLAMC5( LBETA, LT, LEMIN, IEEE, LEMAX, LRMAX )
      END IF
!
      BETA = LBETA
      T = LT
      RND = LRND
      EPS = LEPS
      EMIN = LEMIN
      RMIN = LRMIN
      EMAX = LEMAX
      RMAX = LRMAX
!
      print *, 'BETA=', BETA
!
      print *, 'T=', T
!
      print *, 'RND=', RND
!
      print *, 'EPS=', EPS
!
      print *, 'EMIN=', EMIN
!
      print *, 'RMIN=', RMIN
!
      print *, 'EMAX=', EMAX
!
      print *, 'RMAX=', RMAX
!
      STOP
!
      RETURN
!
9999 FORMAT( / / ' WARNING. The value EMIN may be incorrect: ', ' EMIN = ', I8, / &
' If, after inspection, the value EMIN looks ', ' acceptable please comment out ', / &
' the IF block as marked within the code of routine ', &
' DLAMC2, ', / ' otherwise supply EMIN explicitly.', / )
!
! End of DLAMC2
!
      END SUBROUTINE
!
! *****
!
      DOUBLE PRECISION FUNCTION DLAMC3( A, B )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
      DOUBLE PRECISION A, B
!
! ..
!
! Purpose
! =====
!
! DLAMC3 is intended to force A and B to be stored prior to doing
! the addition of A and B, for use in situations where optimizers
! might hold one of these in a register.
!
! Arguments
! =====
!
! A, B (input) DOUBLE PRECISION
! The values A and B.
!
! =====

```

```

!
! .. Executable Statements ..
!
DLAMC3 = A + B
!
RETURN
!
End of DLAMC3
!
END FUNCTION
!
*****
!
SUBROUTINE DLAMC4( EMIN, START, BASE )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
! INTEGER      BASE, EMIN
! DOUBLE PRECISION  START
! ..
!
! Purpose
! =====
!
! DLAMC4 is a service routine for DLAMC2.
!
! Arguments
! =====
!
! EMIN      (output) EMIN
!            The minimum exponent before (gradual) underflow, computed by
!            setting A = START and dividing by BASE until the previous A
!            can not be recovered.
!
! START     (input) DOUBLE PRECISION
!            The starting point for determining EMIN.
!
! BASE      (input) INTEGER
!            The base of the machine.
!
! =====
!
! .. Local Scalars ..
! INTEGER      I
! DOUBLE PRECISION  A, B1, B2, C1, C2, D1, D2, ONE, RBASE, ZERO
! ..
! .. External Functions ..
! DOUBLE PRECISION  DLAMC3
! EXTERNAL        DLAMC3
! ..
! .. Executable Statements ..
!
! A = START
! ONE = 1
! RBASE = ONE / BASE
! ZERO = 0
! EMIN = 1
! B1 = DLAMC3( A*RBASE, ZERO )
! C1 = A
! C2 = A
! D1 = A
! D2 = A
!+  WHILE( ( C1.EQ.A ).AND.( C2.EQ.A ).AND.
! $      ( D1.EQ.A ).AND.( D2.EQ.A ) ) LOOP
! 10 CONTINUE
!   IF( ( C1.EQ.A ).AND.( C2.EQ.A ).AND.( D1.EQ.A ).AND.( D2.EQ.A ) ) THEN
!     EMIN = EMIN - 1
!     A = B1
!     B1 = DLAMC3( A / BASE, ZERO )
!     C1 = DLAMC3( B1*BASE, ZERO )
!     D1 = ZERO
!     DO 20 I = 1, BASE
!       D1 = D1 + B1
!     20 CONTINUE
!     B2 = DLAMC3( A*RBASE, ZERO )
!     C2 = DLAMC3( B2 / RBASE, ZERO )
!     D2 = ZERO
!     DO 30 I = 1, BASE
!       D2 = D2 + B2
!     30 CONTINUE
!     GO TO 10
!   END IF
!+  END WHILE
!
! RETURN
!
! End of DLAMC4
!
END SUBROUTINE
!
*****
!
SUBROUTINE DLAMC5( BETA, P, EMIN, IEEE, EMAX, RMAX )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
! LOGICAL      IEEE
! INTEGER      BETA, EMAX, EMIN, P
! DOUBLE PRECISION  RMAX
! ..
!
! Purpose
! =====
!
! DLAMC5 attempts to compute RMAX, the largest machine floating-point
! number, without overflow. It assumes that EMAX + abs(EMIN) sum

```

```

! approximately to a power of 2. It will fail on machines where this
! assumption does not hold, for example, the Cyber 205 (EMIN = -28625,
! EMAX = 28718). It will also fail if the value supplied for EMIN is
! too large (i.e. too close to zero), probably with overflow.
!
! Arguments
! =====
!
! BETA      (input) INTEGER
!           The base of floating-point arithmetic.
!
! P         (input) INTEGER
!           The number of base BETA digits in the mantissa of a
!           floating-point value.
!
! EMIN      (input) INTEGER
!           The minimum exponent before (gradual) underflow.
!
! IEEE      (input) LOGICAL
!           A logical flag specifying whether or not the arithmetic
!           system is thought to comply with the IEEE standard.
!
! EMAX      (output) INTEGER
!           The largest exponent before overflow
!
! RMAX      (output) DOUBLE PRECISION
!           The largest machine floating-point number.
!
! =====
!
! .. Parameters ..
! DOUBLE PRECISION      ZERO, ONE
! PARAMETER              ( ZERO = 0.0D0, ONE = 1.0D0 )
!
! .. Local Scalars ..
! INTEGER                EXBITS, EXPSUM, I, LEXP, NBITS, TRY, UEXP
! DOUBLE PRECISION      OLDY, RECBAS, Y, Z
!
! ..
! .. External Functions ..
! DOUBLE PRECISION      DLAMC3
! EXTERNAL              DLAMC3
!
! ..
! .. Intrinsic Functions ..
! INTRINSIC              MOD
!
! .. Executable Statements ..
!
! First compute LEXP and UEXP, two powers of 2 that bound
! abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
! approximately to the bound that is closest to abs(EMIN).
! (EMAX is the exponent of the required number RMAX).
!
! LEXP = 1
! EXBITS = 1
10 CONTINUE
! TRY = LEXP*2
! IF( TRY.LE.( -EMIN ) ) THEN
!     LEXP = TRY
!     EXBITS = EXBITS + 1
!     GO TO 10
! END IF
! IF( LEXP.EQ.-EMIN ) THEN
!     UEXP = LEXP
! ELSE
!     UEXP = TRY
!     EXBITS = EXBITS + 1
! END IF
!
! Now -LEXP is less than or equal to EMIN, and -UEXP is greater
! than or equal to EMIN. EXBITS is the number of bits needed to
! store the exponent.
!
! IF( ( UEXP+EMIN ).GT.( -LEXP-EMIN ) ) THEN
!     EXPSUM = 2*LEXP
! ELSE
!     EXPSUM = 2*UEXP
! END IF
!
! EXPSUM is the exponent range, approximately equal to
! EMAX - EMIN + 1 .
!
! EMAX = EXPSUM + EMIN - 1
! NBITS = 1 + EXBITS + P
!
! NBITS is the total number of bits needed to store a
! floating-point number.
!
! IF( ( MOD( NBITS, 2 ) .EQ.1 ) .AND. ( BETA.EQ.2 ) ) THEN
!
!     Either there are an odd number of bits used to store a
!     floating-point number, which is unlikely, or some bits are
!     not used in the representation of numbers, which is possible,
!     (e.g. Cray machines) or the mantissa has an implicit bit,
!     (e.g. IEEE machines, Dec Vax machines), which is perhaps the
!     most likely. We have to assume the last alternative.
!     If this is true, then we need to reduce EMAX by one because
!     there must be some way of representing zero in an implicit-bit
!     system. On machines like Cray, we are reducing EMAX by one
!     unnecessarily.
!
!     EMAX = EMAX - 1
! END IF
!
! IF( IEEE ) THEN
!
!     Assume we are on an IEEE machine which reserves one exponent
!     for infinity and NaN.
!
!     EMAX = EMAX - 1
! END IF
!
! Now create RMAX, the largest machine number, which should
! be equal to (1.0 - BETA**(-P)) * BETA**EMAX .
!

```

```

! First compute 1.0 - BETA**(-P), being careful that the
! result is less than 1.0 .
!
      RECBAS = ONE / BETA
      Z = BETA - ONE
      Y = ZERO
      DO 20 I = 1, P
        Z = Z*RECBAS
        IF( Y.LT.ONE )      OLDY = Y
        Y = DLAMC3( Y, Z )
20    CONTINUE
      IF( Y.GE.ONE )      Y = OLDY
!
! Now multiply by BETA**EMAX to get RMAX.
!
      DO 30 I = 1, EMAX
        Y = DLAMC3( Y*BETA, ZERO )
30    CONTINUE
!
      RMAX = Y
      RETURN
!
! End of DLAMC5
!
      END SUBROUTINE
      DOUBLE PRECISION FUNCTION DSECND( )
!> \brief \b DSECND Using ETIME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!      DOUBLE PRECISION FUNCTION DSECND( )
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> DSECND returns the user time for a process in seconds.
!> This version gets the time from the EXTERNAL system function ETIME.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!> \ingroup auxOTHERauxiliary
!
! =====
!
! -- LAPACK auxiliary routine (version 3.4.0) --
! -- LAPACK is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!   November 2011
!
! =====
!
! .. Local Scalars ..
      REAL          T1
!
! ..
! .. Local Arrays ..
      REAL          TARRAY( 2 )
!
! ..
! .. External Functions ..
      REAL          ETIME
      EXTERNAL      ETIME
!
! .. Executable Statements ..
!
      T1 = ETIME( TARRAY )
      DSECND = TARRAY( 1 )
      RETURN
!
! End of DSECND
!
      END FUNCTION
      SUBROUTINE DSECNDTST
!> \brief \b DSECNDTST
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!      PROGRAM DSECNDTST
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!> \ingroup auxOTHERauxiliary

```

```

! =====
!                                     PROGRAM DSECNDTST
!
! -- LAPACK test routine (version 3.4.0) --
!
! -- LAPACK computational routine (version 3.4.0) --
! -- LAPACK is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
! =====
!
! .. Parameters ..
! INTEGER          NMAX, ITS
! PARAMETER        ( NMAX = 1000, ITS = 50000 )
!
! .. Local Scalars ..
! INTEGER          I, J
! DOUBLE PRECISION ALPHA, AVG, T1, T2, TNOSEC, TOTAL
!
! .. Local Arrays ..
! DOUBLE PRECISION X( NMAX ), Y( NMAX )
!
! .. External Functions ..
! DOUBLE PRECISION DSECND
! EXTERNAL         DSECND
!
! .. Intrinsic Functions ..
! INTRINSIC        DBLE
!
! .. Executable Statements ..
!
! .. Figure TOTAL flops ..
! TOTAL = DBLE(NMAX) * DBLE(ITS) * 2.0
!
! Initialize X and Y
!
! DO 10 I = 1, NMAX
!   X( I ) = DBLE( 1 ) / DBLE( I )
!   Y( I ) = DBLE( NMAX-I ) / DBLE( NMAX )
! 10 CONTINUE
! ALPHA = 0.315D0
!
! Time TOTAL SAXPY operations
!
! T1 = DSECND( )
! DO 30 J = 1, ITS
!   DO 20 I = 1, NMAX
!     Y( I ) = Y( I ) + ALPHA*X( I )
!   20 CONTINUE
!   ALPHA = -ALPHA
! 30 CONTINUE
! T2 = DSECND( )
! TNOSEC = T2 - T1
! WRITE( 6, 9999 )TOTAL, TNOSEC
! IF( TNOSEC.GT.0.0 ) THEN
!   WRITE( 6, 9998 ) (TOTAL/1.0D6)/TNOSEC
! ELSE
!   WRITE( 6, 9994 )
! END IF
!
! Time TOTAL DAXPY operations with DSECND in the outer loop
!
! T1 = DSECND( )
! DO 50 J = 1, ITS
!   DO 40 I = 1, NMAX
!     Y( I ) = Y( I ) + ALPHA*X( I )
!   40 CONTINUE
!   ALPHA = -ALPHA
!   T2 = DSECND( )
! 50 CONTINUE
!
! Compute the time used in milliseconds used by an average call
! to DSECND.
!
! WRITE( 6, 9997 )T2 - T1
! AVG = ( ( T2-T1 ) - TNOSEC ) * 1000.0D+00/DBLE( ITS )
! IF( AVG.GT.0.0 ) WRITE( 6, 9996 )AVG
!
! Compute the equivalent number of floating point operations used
! by an average call to DSECND.
!
! IF( ( AVG.GT.0.0 ).AND.( TNOSEC.GT.0.0 ) ) &
!   WRITE( 6, 9995 ) (AVG/1000) * TOTAL / TNOSEC
!
! 9999 FORMAT( ' Time for ', G10.3, ' DAXPY ops = ', G10.3, ' seconds' )
! 9998 FORMAT( ' DAXPY performance rate      = ', G10.3, ' mflops ' )
! 9997 FORMAT( ' Including DSECND, time       = ', G10.3, ' seconds' )
! 9996 FORMAT( ' Average time for DSECND     = ', G10.3, &
!   ' milliseconds' )
! 9995 FORMAT( ' Equivalent floating point ops = ', G10.3, ' ops' )
! 9994 FORMAT( ' *** Warning: Time for operations was less or equal', &
!   ' than zero => timing in TESTING might be dubious' )
!
! CALL MYSUB_D(NMAX,X,Y)
! END SUBROUTINE
! SUBROUTINE MYSUB_D(N,X,Y)
!   INTEGER N
!   DOUBLE PRECISION X(N), Y(N)
!   RETURN
! END SUBROUTINE
! REAL FUNCTION SLAMCH( CMACH )
! !!!DEC$ ATTRIBUTES DLLEXPORT :: SLAMCH_1
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
! CHARACTER        CMACH
!
! ..
!
! Purpose
! =====

```

```

!
! SLAMCH determines single precision machine parameters.
!
! Arguments
! =====
! CMACH      (input) CHARACTER*1
!             Specifies the value to be returned by SLAMCH:
!             = 'E' or 'e',  SLAMCH := eps
!             = 'S' or 's',  SLAMCH := sfmin
!             = 'B' or 'b',  SLAMCH := base
!             = 'P' or 'p',  SLAMCH := eps*base
!             = 'N' or 'n',  SLAMCH := t
!             = 'R' or 'r',  SLAMCH := rnd
!             = 'M' or 'm',  SLAMCH := emin
!             = 'U' or 'u',  SLAMCH := rmin
!             = 'L' or 'l',  SLAMCH := emax
!             = 'O' or 'o',  SLAMCH := rmax
!
!             where
!
!             eps = relative machine precision
!             sfmin = safe minimum, such that 1/sfmin does not overflow
!             base = base of the machine
!             prec = eps*base
!             t    = number of (base) digits in the mantissa
!             rnd  = 1.0 when rounding occurs in addition, 0.0 otherwise
!             emin = minimum exponent before (gradual) underflow
!             rmin = underflow threshold - base**(emin-1)
!             emax = largest exponent before overflow
!             rmax = overflow threshold - (base**emax)*(1-eps)
!
! =====
!
! .. Parameters ..
REAL          ONE, ZERO
PARAMETER    ( ONE = 1.0E+0, ZERO = 0.0E+0 )
!
! ..
! .. Local Scalars ..
LOGICAL      FIRST, LRND
INTEGER      BETA, IMAX, IMIN, IT
REAL         BASE, EMAX, EMIN, EPS, PREC, RMACH, RMAX, RMIN, &
             RND, SFMIN, SMALL, T
!
! .. External Functions ..
LOGICAL      LSAME
EXTERNAL     LSAME
!
! .. External Subroutines ..
EXTERNAL     SLAMC2
!
! .. Save statement ..
SAVE        FIRST, EPS, SFMIN, BASE, T, RND, EMIN, RMIN, &
             EMAX, RMAX, PREC
!
! .. Data statements ..
DATA        FIRST / .TRUE. /
!
! .. Executable Statements ..
!
IF( FIRST ) THEN
    FIRST = .FALSE.
    CALL SLAMC2( BETA, IT, LRND, EPS, IMIN, RMIN, IMAX, RMAX )
    BASE = BETA
    T = IT
    IF( LRND ) THEN
        RND = ONE
        EPS = ( BASE**( 1-IT ) ) / 2
    ELSE
        RND = ZERO
        EPS = BASE**( 1-IT )
    END IF
    PREC = EPS*BASE
    EMIN = IMIN
    EMAX = IMAX
    SFMIN = RMIN
    SMALL = ONE / RMAX
    IF( SMALL.GE.SFMIN ) THEN
!
!         Use SMALL plus a bit, to avoid the possibility of rounding
!         causing overflow when computing 1/sfmin.
!
        SFMIN = SMALL*( ONE+EPS )
    END IF
END IF
!
IF( LSAME( CMACH, 'E' ) ) THEN
    RMACH = EPS
ELSE IF( LSAME( CMACH, 'S' ) ) THEN
    RMACH = SFMIN
ELSE IF( LSAME( CMACH, 'B' ) ) THEN
    RMACH = BASE
ELSE IF( LSAME( CMACH, 'P' ) ) THEN
    RMACH = PREC
ELSE IF( LSAME( CMACH, 'N' ) ) THEN
    RMACH = T
ELSE IF( LSAME( CMACH, 'R' ) ) THEN
    RMACH = RND
ELSE IF( LSAME( CMACH, 'M' ) ) THEN
    RMACH = EMIN
ELSE IF( LSAME( CMACH, 'U' ) ) THEN
    RMACH = RMIN
ELSE IF( LSAME( CMACH, 'L' ) ) THEN
    RMACH = EMAX
ELSE IF( LSAME( CMACH, 'O' ) ) THEN
    RMACH = RMAX
END IF
!
SLAMCH = RMACH
RETURN
!
! End of SLAMCH
!
END FUNCTION

```

```

!
!-----
!
SUBROUTINE SLAMC1( BETA, T, RND, IEEE1 )
!
!  -- LAPACK auxiliary routine (version 1.1) --
!  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
!  Courant Institute, Argonne National Lab, and Rice University
!  October 31, 1992
!
!  .. Scalar Arguments ..
LOGICAL      IEEE1, RND
INTEGER      BETA, T
!
!  ..
!
! Purpose
! =====
!
! SLAMC1 determines the machine parameters given by BETA, T, RND, and
! IEEE1.
!
! Arguments
! =====
!
! BETA      (output) INTEGER
!            The base of the machine.
!
! T          (output) INTEGER
!            The number of ( BETA ) digits in the mantissa.
!
! RND       (output) LOGICAL
!            Specifies whether proper rounding ( RND = .TRUE. ) or
!            chopping ( RND = .FALSE. ) occurs in addition. This may not
!            be a reliable guide to the way in which the machine performs
!            its arithmetic.
!
! IEEE1     (output) LOGICAL
!            Specifies whether rounding appears to be done in the IEEE
!            'round to nearest' style.
!
! Further Details
! =====
!
! The routine is based on the routine ENVIRON by Malcolm and
! incorporates suggestions by Gentleman and Marovich. See
!
! Malcolm M. A. (1972) Algorithms to reveal properties of
! floating-point arithmetic. Comms. of the ACM, 15, 949-951.
!
! Gentleman W. M. and Marovich S. B. (1974) More on algorithms
! that reveal properties of floating point arithmetic units.
! Comms. of the ACM, 17, 276-277.
!
!-----
!
! .. Local Scalars ..
LOGICAL      FIRST, LIEEE1, LRND
INTEGER      LBETA, LT
REAL         A, B, C, F, ONE, QTR, SAVEC, T1, T2
!
! ..
!
! .. External Functions ..
REAL         SLAMC3
EXTERNAL     SLAMC3
!
! .. Save statement ..
SAVE        FIRST, LIEEE1, LBETA, LRND, LT
!
! .. Data statements ..
DATA        FIRST / .TRUE. /
!
! .. Executable Statements ..
!
IF( FIRST ) THEN
  FIRST = .FALSE.
  ONE = 1
!
! LBETA, LIEEE1, LT and LRND are the local values of BETA,
! IEEE1, T and RND.
!
! Throughout this routine we use the function SLAMC3 to ensure
! that relevant values are stored and not held in registers, or
! are not affected by optimizers.
!
! Compute a = 2.0**m with the smallest positive integer m such
! that
!
!      fl( a + 1.0 ) = a.
!
!      A = 1
!      C = 1
!
!+ 10  WHILE( C.EQ.ONE )LOOP
!      CONTINUE
!      IF( C.EQ.ONE ) THEN
!        A = 2*A
!        C = SLAMC3( A, ONE )
!        C = SLAMC3( C, -A )
!        GO TO 10
!      END IF
!      END WHILE
!
!      Now compute b = 2.0**m with the smallest positive integer m
!      such that
!
!      fl( a + b ) .gt. a.
!
!      B = 1
!      C = SLAMC3( A, B )
!
!+ 20  WHILE( C.EQ.A )LOOP
!      CONTINUE
!      IF( C.EQ.A ) THEN
!        B = 2*B
!        C = SLAMC3( A, B )

```

```

        GO TO 20
    END IF
END WHILE
!+
!
! Now compute the base. a and c are neighbouring floating point
! numbers in the interval ( beta**t, beta**( t + 1 ) ) and so
! their difference is beta. Adding 0.25 to c is to ensure that it
! is truncated to beta and not ( beta - 1 ).
!
    QTR = ONE / 4
    SAVEC = C
    C = SLAMC3( C, -A )
    LBETA = C + QTR
!
! Now determine whether rounding or chopping occurs, by adding a
! bit less than beta/2 and a bit more than beta/2 to a.
!
    B = LBETA
    F = SLAMC3( B / 2, -B / 100 )
    C = SLAMC3( F, A )
    IF( C.EQ.A ) THEN
        LRND = .TRUE.
    ELSE
        LRND = .FALSE.
    END IF
    F = SLAMC3( B / 2, B / 100 )
    C = SLAMC3( F, A )
    IF( ( LRND ) .AND. ( C.EQ.A ) ) &
        LRND = .FALSE.
!
! Try and decide whether rounding is done in the IEEE 'round to
! nearest' style. B/2 is half a unit in the last place of the two
! numbers A and SAVEC. Furthermore, A is even, i.e. has last bit
! zero, and SAVEC is odd. Thus adding B/2 to A should not change
! A, but adding B/2 to SAVEC should change SAVEC.
!
    T1 = SLAMC3( B / 2, A )
    T2 = SLAMC3( B / 2, SAVEC )
    LIEEE1 = ( T1.EQ.A ) .AND. ( T2.GT.SAVEC ) .AND. LRND
!
! Now find the mantissa, t. It should be the integer part of
! log to the base beta of a, however it is safer to determine t
! by powering. So we find t as the smallest positive integer for
! which
!
    fl( beta**t + 1.0 ) = 1.0.
!
    LT = 0
    A = 1
    C = 1
!
!+
30  WHILE( C.EQ.ONE ) LOOP
    CONTINUE
    IF( C.EQ.ONE ) THEN
        LT = LT + 1
        A = A*LBETA
        C = SLAMC3( A, ONE )
        C = SLAMC3( C, -A )
        GO TO 30
    END IF
END WHILE
!+
!
END IF
!
    BETA = LBETA
    T = LT
    RND = LRND
    IEEE1 = LIEEE1
    RETURN
!
! End of SLAMC1
!
END SUBROUTINE
!
!*****
!
SUBROUTINE SLAMC2( BETA, T, RND, EPS, EMIN, RMIN, EMAX, RMAX )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
    LOGICAL      RND
    INTEGER      BETA, EMAX, EMIN, T
    REAL         EPS, RMAX, RMIN
!
! ..
!
! Purpose
! =====
!
! SLAMC2 determines the machine parameters specified in its argument
! list.
!
! Arguments
! =====
!
! BETA      (output) INTEGER
!            The base of the machine.
!
! T         (output) INTEGER
!            The number of ( BETA ) digits in the mantissa.
!
! RND       (output) LOGICAL
!            Specifies whether proper rounding ( RND = .TRUE. ) or
!            chopping ( RND = .FALSE. ) occurs in addition. This may not
!            be a reliable guide to the way in which the machine performs
!            its arithmetic.
!
! EPS       (output) REAL
!            The smallest positive number such that
!
!            fl( 1.0 - EPS ) .LT. 1.0,

```

```

!
!       where fl denotes the computed value.
!
! EMIN   (output) INTEGER
!       The minimum exponent before (gradual) underflow occurs.
!
! RMIN   (output) REAL
!       The smallest normalized number for the machine, given by
!       BASE** ( EMIN - 1 ), where BASE is the floating point value
!       of BETA.
!
! EMAX   (output) INTEGER
!       The maximum exponent before overflow occurs.
!
! RMAX   (output) REAL
!       The largest positive number for the machine, given by
!       BASE**EMAX * ( 1 - EPS ), where BASE is the floating point
!       value of BETA.
!
! Further Details
! =====
!
! The computation of EPS is based on a routine PARANOIA by
! W. Kahan of the University of California at Berkeley.
!
! =====
!
! .. Local Scalars ..
LOGICAL      FIRST, IEEE, IWARN, LIEEE1, LRND
INTEGER      GNMN, GPMN, I, LBETA, LEMAX, LEMIN, LT, &
              NGNMN, NGPMN
REAL         A, B, C, HALF, LEPS, LRMAX, LRMIN, ONE, RBASE, &
              SIXTH, SMALL, THIRD, TWO, ZERO
!
! .. External Functions ..
REAL         SLAMC3
EXTERNAL     SLAMC3
!
! .. External Subroutines ..
EXTERNAL     SLAMC1, SLAMC4, SLAMC5
!
! .. Intrinsic Functions ..
INTRINSIC    ABS, MAX, MIN
!
! .. Save statement ..
SAVE        FIRST, IWARN, LBETA, LEMAX, LEMIN, LEPS, LRMAX, &
              LRMIN, LT
!
! .. Data statements ..
DATA        FIRST / .TRUE. / , IWARN / .FALSE. /
!
! .. Executable Statements ..
!
IF( FIRST ) THEN
  FIRST = .FALSE.
  ZERO = 0
  ONE = 1
  TWO = 2
!
! LBETA, LT, LRND, LEPS, LEMIN and LRMIN are the local values of
! BETA, T, RND, EPS, EMIN and RMIN.
!
! Throughout this routine we use the function SLAMC3 to ensure
! that relevant values are stored and not held in registers, or
! are not affected by optimizers.
!
! SLAMC1 returns the parameters LBETA, LT, LRND and LIEEE1.
!
CALL SLAMC1( LBETA, LT, LRND, LIEEE1 )
!
! Start to find EPS.
!
  B = LBETA
  A = B** ( -LT )
  LEPS = A
!
! Try some tricks to see whether or not this is the correct EPS.
!
  B = TWO / 3
  HALF = ONE / 2
  SIXTH = SLAMC3( B, -HALF )
  THIRD = SLAMC3( SIXTH, SIXTH )
  B = SLAMC3( THIRD, -HALF )
  B = SLAMC3( B, SIXTH )
  B = ABS( B )
  IF( B.LT.LEPS ) &
    B = LEPS
!
  LEPS = 1
!
!+ 10  WHILE( ( LEPS.GT.B ).AND.( B.GT.ZERO ) ) LOOP
!      CONTINUE
!      IF( ( LEPS.GT.B ) .AND. ( B.GT.ZERO ) ) THEN
!        LEPS = B
!        C = SLAMC3( HALF*LEPS, ( TWO**5 )*( LEPS**2 ) )
!        C = SLAMC3( HALF, -C )
!        B = SLAMC3( HALF, C )
!        C = SLAMC3( HALF, -B )
!        B = SLAMC3( HALF, C )
!        GO TO 10
!      END IF
!    END WHILE
!
!    IF( A.LT.LEPS ) &
!      LEPS = A
!
!    Computation of EPS complete.
!
!    Now find EMIN. Let A = + or - 1, and + or - ( 1 + BASE**(-3) ).
!    Keep dividing A by BETA until (gradual) underflow occurs. This
!    is detected when we cannot recover the previous A.
!
!    RBASE = ONE / LBETA
!    SMALL = ONE

```

```

DO 20 I = 1, 3
  SMALL = SLAMC3( SMALL*RBASE, ZERO )
20 CONTINUE
  A = SLAMC3( ONE, SMALL )
  CALL SLAMC4( NGPMIN, ONE, LBETA )
  CALL SLAMC4( NGNMIN, -ONE, LBETA )
  CALL SLAMC4( GPMIN, A, LBETA )
  CALL SLAMC4( GNMIN, -A, LBETA )
  IEEE = .FALSE.

!
! IF( ( NGPMIN.EQ.NGNMIN ) .AND. ( GPMIN.EQ.GNMIN ) ) THEN
!   IF( NGPMIN.EQ.GPMIN ) THEN
!     LEMIN = NGPMIN
!     ( Non twos-complement machines, no gradual underflow;
!       e.g., VAX )
!   ELSE IF( ( GPMIN-NGPMIN ).EQ.3 ) THEN
!     LEMIN = NGPMIN - 1 + LT
!     IEEE = .TRUE.
!     ( Non twos-complement machines, with gradual underflow;
!       e.g., IEEE standard followers )
!   ELSE
!     LEMIN = MIN( NGPMIN, GPMIN )
!     ( A guess; no known machine )
!     IWARN = .TRUE.
!   END IF

!   ELSE IF( ( NGPMIN.EQ.GPMIN ) .AND. ( NGNMIN.EQ.GNMIN ) ) THEN
!     IF( ABS( NGPMIN-NGNMIN ).EQ.1 ) THEN
!       LEMIN = MAX( NGPMIN, NGNMIN )
!       ( Twos-complement machines, no gradual underflow;
!       e.g., CYBER 205 )
!     ELSE
!       LEMIN = MIN( NGPMIN, NGNMIN )
!       ( A guess; no known machine )
!       IWARN = .TRUE.
!     END IF

!   ELSE IF( ( ABS( NGPMIN-NGNMIN ).EQ.1 ) .AND. &
!     ( GPMIN.EQ.GNMIN ) ) THEN
!     IF( ( GPMIN-MIN( NGPMIN, NGNMIN ) ).EQ.3 ) THEN
!       LEMIN = MAX( NGPMIN, NGNMIN ) - 1 + LT
!       ( Twos-complement machines with gradual underflow;
!       no known machine )
!     ELSE
!       LEMIN = MIN( NGPMIN, NGNMIN )
!       ( A guess; no known machine )
!       IWARN = .TRUE.
!     END IF

!   ELSE
!     LEMIN = MIN( NGPMIN, NGNMIN, GPMIN, GNMIN )
!     ( A guess; no known machine )
!     IWARN = .TRUE.
!   END IF

! **
! Comment out this if block if EMIN is ok
! IF( IWARN ) THEN
!   FIRST = .TRUE.
!   WRITE( 6, FMT = 9999 )LEMIN
! END IF

! **
!
! Assume IEEE arithmetic if we found denormalised numbers above,
! or if arithmetic seems to round in the IEEE style, determined
! in routine SLAMC1. A true IEEE machine should have both things
! true; however, faulty machines may have one or the other.
!
! IEEE = IEEE .OR. LIEEE1
!
! Compute RMIN by successive division by BETA. We could compute
! RMIN as BASE** ( EMIN - 1 ), but some machines underflow during
! this computation.
!
! LRMIN = 1
! DO 30 I = 1, 1 - LEMIN
!   LRMIN = SLAMC3( LRMIN*RBASE, ZERO )
! 30 CONTINUE
!
! Finally, call SLAMC5 to compute EMAX and RMAX.
! CALL SLAMC5( LBETA, LT, LEMIN, IEEE, LEMAX, LRMAX )
! END IF

! BETA = LBETA
! T = LT
! RND = LRND
! EPS = LEPS
! EMIN = LEMIN
! RMIN = LRMIN
! EMAX = LEMAX
! RMAX = LRMAX
!
! RETURN
!
! 9999 FORMAT( / / ' WARNING. The value EMIN may be incorrect:-', &
!   ' EMIN = ', I8, / &
!   ' If, after inspection, the value EMIN looks', &
!   ' acceptable please comment out ', &
!   / ' the IF block as marked within the code of routine', &
!   ' SLAMC2,', / ' otherwise supply EMIN explicitly.', / )
!
! End of SLAMC2
!
! END SUBROUTINE
!
! *****
!
! REAL          FUNCTION SLAMC3( A, B )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!

```

```

! .. Scalar Arguments ..
! REAL          A, B
! ..
! Purpose
! =====
! SLAMC3 is intended to force A and B to be stored prior to doing
! the addition of A and B , for use in situations where optimizers
! might hold one of these in a register.
! Arguments
! =====
! A, B          (input) REAL
!               The values A and B.
!
! .. Executable Statements ..
!
! SLAMC3 = A + B
!
! RETURN
!
! End of SLAMC3
!
! END FUNCTION
!
! *****
!
! SUBROUTINE SLAMC4( EMIN, START, BASE )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
! INTEGER        BASE, EMIN
! REAL           START
! ..
! Purpose
! =====
! SLAMC4 is a service routine for SLAMC2.
! Arguments
! =====
! EMIN          (output) EMIN
!               The minimum exponent before (gradual) underflow, computed by
!               setting A = START and dividing by BASE until the previous A
!               can not be recovered.
! START         (input) REAL
!               The starting point for determining EMIN.
! BASE          (input) INTEGER
!               The base of the machine.
!
! =====
!
! .. Local Scalars ..
! INTEGER        I
! REAL           A, B1, B2, C1, C2, D1, D2, ONE, RBASE, ZERO
! ..
! .. External Functions ..
! REAL           SLAMC3
! EXTERNAL       SLAMC3
! ..
! .. Executable Statements ..
!
! A = START
! ONE = 1
! RBASE = ONE / BASE
! ZERO = 0
! EMIN = 1
! B1 = SLAMC3( A*RBASE, ZERO )
! C1 = A
! C2 = A
! D1 = A
! D2 = A
!+  WHILE( ( C1.EQ.A ).AND.( C2.EQ.A ).AND.
! $      ( D1.EQ.A ).AND.( D2.EQ.A ) ) LOOP
10  CONTINUE
! IF( ( C1.EQ.A ).AND.( C2.EQ.A ).AND. ( D1.EQ.A ).AND. &
!   ( D2.EQ.A ) ) THEN
!   EMIN = EMIN - 1
!   A = B1
!   B1 = SLAMC3( A / BASE, ZERO )
!   C1 = SLAMC3( B1*BASE, ZERO )
!   D1 = ZERO
!   DO 20 I = 1, BASE
!     D1 = D1 + B1
!   20  CONTINUE
!   B2 = SLAMC3( A*RBASE, ZERO )
!   C2 = SLAMC3( B2 / RBASE, ZERO )
!   D2 = ZERO
!   DO 30 I = 1, BASE
!     D2 = D2 + B2
!   30  CONTINUE
!   GO TO 10
! END IF
!+  END WHILE
!
! RETURN
!
! End of SLAMC4
!
! END SUBROUTINE
!
! *****

```

```

!
! SUBROUTINE SLAMC5( BETA, P, EMIN, IEEE, EMAX, RMAX )
!
! -- LAPACK auxiliary routine (version 1.1) --
! Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
! Courant Institute, Argonne National Lab, and Rice University
! October 31, 1992
!
! .. Scalar Arguments ..
! LOGICAL      IEEE
! INTEGER      BETA, EMAX, EMIN, P
! REAL         RMAX
! ..
!
! Purpose
! =====
!
! SLAMC5 attempts to compute RMAX, the largest machine floating-point
! number, without overflow. It assumes that EMAX + abs(EMIN) sum
! approximately to a power of 2. It will fail on machines where this
! assumption does not hold, for example, the Cyber 205 (EMIN = -28625,
! EMAX = 28718). It will also fail if the value supplied for EMIN is
! too large (i.e. too close to zero), probably with overflow.
!
! Arguments
! =====
!
! BETA      (input) INTEGER
!           The base of floating-point arithmetic.
!
! P         (input) INTEGER
!           The number of base BETA digits in the mantissa of a
!           floating-point value.
!
! EMIN      (input) INTEGER
!           The minimum exponent before (gradual) underflow.
!
! IEEE      (input) LOGICAL
!           A logical flag specifying whether or not the arithmetic
!           system is thought to comply with the IEEE standard.
!
! EMAX      (output) INTEGER
!           The largest exponent before overflow
!
! RMAX      (output) REAL
!           The largest machine floating-point number.
!
! =====
!
! .. Parameters ..
! REAL      ZERO, ONE
! PARAMETER ( ZERO = 0.0E0, ONE = 1.0E0 )
! ..
! .. Local Scalars ..
! INTEGER    EXBITS, EXPSUM, I, LEXP, NBITS, TRY, UEXP
! REAL       OLDY, RECBAS, Y, Z
! ..
! .. External Functions ..
! REAL      SLAMC3
! EXTERNAL  SLAMC3
! ..
! .. Intrinsic Functions ..
! INTRINSIC MOD
! ..
! .. Executable Statements ..
!
! First compute LEXP and UEXP, two powers of 2 that bound
! abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
! approximately to the bound that is closest to abs(EMIN).
! (EMAX is the exponent of the required number RMAX).
!
! LEXP = 1
! EXBITS = 1
10 CONTINUE
! TRY = LEXP*2
! IF( TRY.LE.( -EMIN ) ) THEN
!     LEXP = TRY
!     EXBITS = EXBITS + 1
!     GO TO 10
! END IF
! IF( LEXP.EQ.-EMIN ) THEN
!     UEXP = LEXP
! ELSE
!     UEXP = TRY
!     EXBITS = EXBITS + 1
! END IF
!
! Now -LEXP is less than or equal to EMIN, and -UEXP is greater
! than or equal to EMIN. EXBITS is the number of bits needed to
! store the exponent.
!
! IF( ( UEXP+EMIN ).GT.( -LEXP-EMIN ) ) THEN
!     EXPSUM = 2*LEXP
! ELSE
!     EXPSUM = 2*UEXP
! END IF
!
! EXPSUM is the exponent range, approximately equal to
! EMAX - EMIN + 1 .
!
! EMAX = EXPSUM + EMIN - 1
! NBITS = 1 + EXBITS + P
!
! NBITS is the total number of bits needed to store a
! floating-point number.
!
! IF( ( MOD( NBITS, 2 ) .EQ.1 ) .AND. ( BETA.EQ.2 ) ) THEN
!
!     Either there are an odd number of bits used to store a
!     floating-point number, which is unlikely, or some bits are
!     not used in the representation of numbers, which is possible,
!     (e.g. Cray machines) or the mantissa has an implicit bit,
!     (e.g. IEEE machines, Dec Vax machines), which is perhaps the
!     most likely. We have to assume the last alternative.
!

```

```

!      If this is true, then we need to reduce EMAX by one because
!      there must be some way of representing zero in an implicit-bit
!      system. On machines like Cray, we are reducing EMAX by one
!      unnecessarily.
!
      EMAX = EMAX - 1
END IF
!
!      IF( IEEE ) THEN
!
!      Assume we are on an IEEE machine which reserves one exponent
!      for infinity and NaN.
!
      EMAX = EMAX - 1
END IF
!
!      Now create RMAX, the largest machine number, which should
!      be equal to (1.0 - BETA**(-P)) * BETA**EMAX .
!
!      First compute 1.0 - BETA**(-P), being careful that the
!      result is less than 1.0 .
!
      RECBAS = ONE / BETA
      Z = BETA - ONE
      Y = ZERO
      DO 20 I = 1, P
         Z = Z*RECBAS
         IF( Y.LT.ONE ) &          OLDY = Y
         Y = SLAMC3( Y, Z )
      20 CONTINUE
      IF( Y.GE.ONE ) &
         Y = OLDY
!
!      Now multiply by BETA**EMAX to get RMAX.
!
      DO 30 I = 1, EMAX
         Y = SLAMC3( Y*BETA, ZERO )
      30 CONTINUE
!
      RMAX = Y
      RETURN
!
!      End of SLAMC5
!
END SUBROUTINE
REAL FUNCTION SECOND( )
!> \brief \b SECOND Using ETIME
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      REAL FUNCTION SECOND( )
!
!
!> \par Purpose:
!      =====
!>
!> \verbatim
!>
!> SECOND returns the user time for a process in seconds.
!> This version gets the time from the EXTERNAL system function ETIME.
!> \endverbatim
!
!      Authors:
!      =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup auxOTHERauxiliary
!
!      =====
!
!      -- LAPACK auxiliary routine (version 3.4.0) --
!      -- LAPACK is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!      November 2011
!
!      =====
!
!      .. Local Scalars ..
      REAL          T1
!
!      .. Local Arrays ..
      REAL          TARRAY( 2 )
!
!      .. External Functions ..
      REAL          ETIME
      EXTERNAL      ETIME
!
!      .. Executable Statements ..
!
      T1 = ETIME( TARRAY )
      SECOND = TARRAY( 1 )
      RETURN
!
!      End of SECOND
!
      END FUNCTION
!
SUBROUTINE SECONDTST
!> \brief \b SECONDTST
!
!      ===== DOCUMENTATION =====

```

```

!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup auxOTHERcomputational
!
! ===== PROGRAM SECONDTST =====
!
! -- LAPACK test routine (version 3.4.0) --
!
! -- LAPACK computational routine (version 3.4.0) --
! -- LAPACK is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
! =====
!
! .. Parameters ..
! INTEGER          NMAX, ITS
! PARAMETER        ( NMAX = 1000, ITS = 50000 )
!
! .. Local Scalars ..
! INTEGER          I, J
! REAL             ALPHA, AVG, T1, T2, TNOSEC, TOTAL
!
! .. Local Arrays ..
! REAL             X( NMAX ), Y( NMAX )
!
! .. External Functions ..
! REAL            SECOND
! EXTERNAL         SECOND
!
! .. Intrinsic Functions ..
! INTRINSIC        REAL
!
! .. Executable Statements ..
!
! .. Figure TOTAL flops ..
! TOTAL = REAL(NMAX) * REAL(ITS) * 2.0
!
! Initialize X and Y
!
! DO 10 I = 1, NMAX
!   X( I ) = REAL( 1 ) / REAL( I )
!   Y( I ) = REAL( NMAX-I ) / REAL( NMAX )
10 CONTINUE
! ALPHA = 0.315
!
! Time TOTAL SAXPY operations
!
! T1 = SECOND( )
! DO 30 J = 1, ITS
!   DO 20 I = 1, NMAX
!     Y( I ) = Y( I ) + ALPHA*X( I )
20   CONTINUE
!   ALPHA = -ALPHA
30 CONTINUE
! T2 = SECOND( )
! TNOSEC = T2 - T1
! WRITE( 6, 9999 )TOTAL, TNOSEC
! IF( TNOSEC.GT.0.0 ) THEN
!   WRITE( 6, 9998 ) (TOTAL/1.0E6)/TNOSEC
! ELSE
!   WRITE( 6, 9994 )
! END IF
!
! Time TOTAL SAXPY operations with SECOND in the outer loop
!
! T1 = SECOND( )
! DO 50 J = 1, ITS
!   DO 40 I = 1, NMAX
!     Y( I ) = Y( I ) + ALPHA*X( I )
40   CONTINUE
!   ALPHA = -ALPHA
!   T2 = SECOND( )
50 CONTINUE
!
! Compute the time used in milliseconds used by an average call
! to SECOND.
!
! WRITE( 6, 9997 )T2 - T1
! AVG = ( ( T2-T1 ) - TNOSEC ) * 1000.0E+00/REAL( ITS )
! IF( AVG.GT.0.0 ) & WRITE( 6, 9996 )AVG
!
! Compute the equivalent number of floating point operations used
! by an average call to SECOND.
!
! IF( ( AVG.GT.0.0 ).AND.( TNOSEC.GT.0.0 ) ) &
!   WRITE( 6, 9995 ) (AVG/1000) * TOTAL / TNOSEC
!
! 9999 FORMAT( ' Time for ', G10.3, ' SAXPY ops = ', G10.3, ' seconds' )
! 9998 FORMAT( ' SAXPY performance rate = ', G10.3, ' mflops ' )
! 9997 FORMAT( ' Including SECOND, time = ', G10.3, ' seconds' )
! 9996 FORMAT( ' Average time for SECOND = ', G10.3, &
!   ' milliseconds' )
! 9995 FORMAT( ' Equivalent floating point ops = ', G10.3, ' ops' )
! 9994 FORMAT( ' *** Warning: Time for operations was less or equal', &
!   ' than zero => timing in TESTING might be dubious' )
!
! CALL MYSUB_S(NMAX,X,Y)
! !print *, 'done with secondtst'
! END SUBROUTINE
!
! SUBROUTINE MYSUB_S(N,X,Y)

```

```

        INTEGER, INTENT(IN):: N
        REAL X(N), Y(N)
        RETURN
    END SUBROUTINE
    SUBROUTINE TSTIEE
!> \brief \b TSTIEE
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup auxOTHERauxiliary
!
! =====
!
! PROGRAM TSTIEE
!
! -- LAPACK test routine (version 3.4.0) --
! Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
! November 2006
!
! .. External Functions ..
!   INTEGER      ILAENV
!   EXTERNAL     ILAENV
! ..
! .. Local Scalars ..
!   INTEGER      IEEEOK
! ..
! .. Executable Statements ..
!
!   WRITE( 6, FMT = * ) 'We are about to check whether infinity arithmetic'
!   WRITE( 6, FMT = * ) 'can be trusted. If this test hangs, set'
!   WRITE( 6, FMT = * ) 'ILAENV = 0 for ISPEC = 10 in LAPACK/SRC/ilaenv.f'
!
!   IEEEOK = ILAENV( 10, 'ILAENV', 'N', 1, 2, 3, 4 )
!   WRITE( 6, FMT = * )
!
!   IF( IEEEOK.EQ.0 ) THEN
!       WRITE( 6, FMT = * ) 'Infinity arithmetic did not perform per the ieee spec'
!   ELSE
!       WRITE( 6, FMT = * ) 'Infinity arithmetic performed as per the ieee spec.'
!       WRITE( 6, FMT = * ) 'However, this is not an exhaustive test and does not'
!       WRITE( 6, FMT = * ) 'guarantee that infinity arithmetic meets the', ' ieee spec.'
!   END IF
!
!   WRITE( 6, FMT = * )
!   WRITE( 6, FMT = * ) 'We are about to check whether NaN arithmetic'
!   WRITE( 6, FMT = * ) 'can be trusted. If this test hangs, set'
!   WRITE( 6, FMT = * ) 'ILAENV = 0 for ISPEC = 11 in LAPACK/SRC/ilaenv.f'
!   IEEEOK = ILAENV( 11, 'ILAENV', 'N', 1, 2, 3, 4 )
!
!   WRITE( 6, FMT = * )
!   IF( IEEEOK.EQ.0 ) THEN
!       WRITE( 6, FMT = * ) 'NaN arithmetic did not perform per the ieee spec'
!   ELSE
!       WRITE( 6, FMT = * ) 'NaN arithmetic performed as per the ieee', ' spec.'
!       WRITE( 6, FMT = * ) 'However, this is not an exhaustive test and does not'
!       WRITE( 6, FMT = * ) 'guarantee that NaN arithmetic meets the', ' ieee spec.'
!   END IF
!   WRITE( 6, FMT = * )
!
! END SUBROUTINE
!
!
! INTEGER      FUNCTION ILAENV( ISPEC, NAME, OPTS, N1, N2, N3, N4 )
!
! -- LAPACK auxiliary routine (version 3.4.0) --
! Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
! November 2006
!
! .. Scalar Arguments ..
! CHARACTER*( *) NAME, OPTS
! INTEGER      ISPEC, N1, N2, N3, N4
! ..
!
! Purpose
! =====
!
! ILAENV is called from the LAPACK routines to choose problem-dependent
! parameters for the local environment. See ISPEC for a description of
! the parameters.
!
! This version provides a set of parameters which should give good,
! but not optimal, performance on many of the currently available
! computers. Users are encouraged to modify this subroutine to set
! the tuning parameters for their particular machine using the option
! and problem size information in the arguments.
!
! This routine will not function correctly if it is converted to all
! lower case. Converting it to all upper case is allowed.
!
! Arguments:
! =====
!
! ISPEC (input) INTEGER
!   Specifies the parameter to be returned as the value of
!   ILAENV.
!   = 1: the optimal blocksize; if this value is 1, an unblocked
!       algorithm will give the best performance.
!   = 2: the minimum block size for which the block routine
!       should be used; if the usable block size is less than
!       this value, an unblocked routine should be used.
!   = 3: the crossover point (in a block routine, for N less
!       than this value, an unblocked routine should be used)

```

```

!      = 4: the number of shifts, used in the nonsymmetric
!      eigenvalue routines
!      = 5: the minimum column dimension for blocking to be used;
!      rectangular blocks must have dimension at least k by m,
!      where k is given by ILAENV(2,...) and m by ILAENV(5,...)
!      = 6: the crossover point for the SVD (when reducing an m by n
!      matrix to bidiagonal form, if max(m,n)/min(m,n) exceeds
!      this value, a QR factorization is used first to reduce
!      the matrix to a triangular form.)
!      = 7: the number of processors
!      = 8: the crossover point for the multishift QR and QZ methods
!      for nonsymmetric eigenvalue problems.
!      = 9: maximum size of the subproblems at the bottom of the
!      computation tree in the divide-and-conquer algorithm
!      (used by xGELSD and xGESDD)
!      =10: ieee NaN arithmetic can be trusted not to trap
!      =11: infinity arithmetic can be trusted not to trap
!
! NAME      (input) CHARACTER*(*)
!           The name of the calling subroutine, in either upper case or
!           lower case.
!
! OPTS      (input) CHARACTER*(*)
!           The character options to the subroutine NAME, concatenated
!           into a single character string. For example, UPLO = 'U',
!           TRANS = 'T', and DIAG = 'N' for a triangular routine would
!           be specified as OPTS = 'UTN'.
!
! N1        (input) INTEGER
! N2        (input) INTEGER
! N3        (input) INTEGER
! N4        (input) INTEGER
!           Problem dimensions for the subroutine NAME; these may not all
!           be required.
!
! (ILAENV)  (output) INTEGER
!           >= 0: the value of the parameter specified by ISPEC
!           < 0: if ILAENV = -k, the k-th argument had an illegal value.
!
! Further Details
! =====
!
! The following conventions have been used when calling ILAENV from the
! LAPACK routines:
! 1) OPTS is a concatenation of all of the character options to
!    subroutine NAME, in the same order that they appear in the
!    argument list for NAME, even if they are not used in determining
!    the value of the parameter specified by ISPEC.
! 2) The problem dimensions N1, N2, N3, N4 are specified in the order
!    that they appear in the argument list for NAME. N1 is used
!    first, N2 second, and so on, and unused problem dimensions are
!    passed a value of -1.
! 3) The parameter value returned by ILAENV is checked for validity in
!    the calling subroutine. For example, ILAENV is used to retrieve
!    the optimal blocksize for STRTRI as follows:
!
!     NB = ILAENV( 1, 'STRTRI', UPLO // DIAG, N, -1, -1, -1 )
!     IF( NB.LE.1 ) NB = MAX( 1, N )
!
! =====
!
! .. Local Scalars ..
LOGICAL          CNAME, SNAME
CHARACTER*1      C1
CHARACTER*2      C2, C4
CHARACTER*3      C3
CHARACTER*6      SUBNAM
INTEGER          I, IC, IZ, NB, NBMIN, NX
!
! .. Intrinsic Functions ..
INTRINSIC        CHAR, ICHAR, INT, MIN, REAL
!
! .. External Functions ..
INTEGER          IEEECK
EXTERNAL         IEEECK
!
! .. Executable Statements ..
!
GO TO ( 100, 100, 100, 400, 500, 600, 700, 800, 900, 1000, &
1100 ) ISPEC
!
Invalid value for ISPEC
!
ILAENV = -1
RETURN
!
100 CONTINUE
!
Convert NAME to upper case if the first character is lower case.
!
ILAENV = 1
SUBNAM = NAME
IC = ICHAR( SUBNAM( 1:1 ) )
IZ = ICHAR( 'Z' )
IF( IZ.EQ.90 .OR. IZ.EQ.122 ) THEN
!
ASCII character set
!
IF( IC.GE.97 .AND. IC.LE.122 ) THEN
SUBNAM( 1:1 ) = CHAR( IC-32 )
DO 10 I = 2, 6
IC = ICHAR( SUBNAM( I:I ) )
IF( IC.GE.97 .AND. IC.LE.122 ) SUBNAM( I:I ) = CHAR( IC-32 )
10 CONTINUE
END IF
!
ELSE IF( IZ.EQ.233 .OR. IZ.EQ.169 ) THEN
!
EBCDIC character set
!
IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR. &
( IC.GE.145 .AND. IC.LE.153 ) .OR. &
( IC.GE.162 .AND. IC.LE.169 ) ) THEN
SUBNAM( 1:1 ) = CHAR( IC+64 )

```

```

DO 20 I = 2, 6
  IC = ICHAR( SUBNAM( I:I ) )
  IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR. &
    ( IC.GE.145 .AND. IC.LE.153 ) .OR. &
    ( IC.GE.162 .AND. IC.LE.169 ) ) &
    SUBNAM( I:I ) = CHAR( IC+64 )
20  CONTINUE
END IF
!
! ELSE IF( IZ.EQ.218 .OR. IZ.EQ.250 ) THEN
!
!   Prime machines: ASCII+128
!
  IF( IC.GE.225 .AND. IC.LE.250 ) THEN
    SUBNAM( 1:1 ) = CHAR( IC-32 )
    DO 30 I = 2, 6
      IC = ICHAR( SUBNAM( I:I ) )
      IF( IC.GE.225 .AND. IC.LE.250 ) &
        SUBNAM( I:I ) = CHAR( IC-32 )
30    CONTINUE
    END IF
  END IF
!
! C1 = SUBNAM( 1:1 )
! SNAME = C1.EQ.'S' .OR. C1.EQ.'D'
! CNAME = C1.EQ.'C' .OR. C1.EQ.'Z'
! IF( .NOT.( CNAME .OR. SNAME ) ) &
!   RETURN
! C2 = SUBNAM( 2:3 )
! C3 = SUBNAM( 4:6 )
! C4 = C3( 2:3 )
!
! GO TO ( 110, 200, 300 ) ISPEC
!
110 CONTINUE
!
! ISPEC = 1: block size
!
! In these examples, separate code is provided for setting NB for
! real and complex. We assume that NB will take the same value in
! single or double precision.
!
NB = 1
!
IF( C2.EQ.'GE' ) THEN
  IF( C3.EQ.'TRF' ) THEN
    IF( SNAME ) THEN
      NB = 64
    ELSE
      NB = 64
    END IF
  ELSE IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR. &
    C3.EQ.'QLF' ) THEN
    IF( SNAME ) THEN
      NB = 32
    ELSE
      NB = 32
    END IF
  ELSE IF( C3.EQ.'HRD' ) THEN
    IF( SNAME ) THEN
      NB = 32
    ELSE
      NB = 32
    END IF
  ELSE IF( C3.EQ.'BRD' ) THEN
    IF( SNAME ) THEN
      NB = 32
    ELSE
      NB = 32
    END IF
  ELSE IF( C3.EQ.'TRI' ) THEN
    IF( SNAME ) THEN
      NB = 64
    ELSE
      NB = 64
    END IF
  ELSE IF( C2.EQ.'PO' ) THEN
    IF( C3.EQ.'TRF' ) THEN
      IF( SNAME ) THEN
        NB = 64
      ELSE
        NB = 64
      END IF
    END IF
  ELSE IF( C2.EQ.'SY' ) THEN
    IF( C3.EQ.'TRF' ) THEN
      IF( SNAME ) THEN
        NB = 64
      ELSE
        NB = 64
      END IF
    ELSE IF( SNAME .AND. C3.EQ.'TRD' ) THEN
      NB = 32
    ELSE IF( SNAME .AND. C3.EQ.'GST' ) THEN
      NB = 64
    END IF
  ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
    IF( C3.EQ.'TRF' ) THEN
      NB = 64
    ELSE IF( C3.EQ.'TRD' ) THEN
      NB = 32
    ELSE IF( C3.EQ.'GST' ) THEN
      NB = 64
    END IF
  ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
    IF( C3( 1:1 ).EQ.'G' ) THEN
      IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
        C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
        C4.EQ.'BR' ) THEN
        NB = 32
      END IF
    ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
      IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &

```

```

        C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
        C4.EQ.'BR' ) THEN
        NB = 32
    END IF
END IF
ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
    IF( C3( 1:1 ).EQ.'G' ) THEN
        IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
            C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
            C4.EQ.'BR' ) THEN
            NB = 32
        END IF
    ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
        IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
            C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
            C4.EQ.'BR' ) THEN
            NB = 32
        END IF
    END IF
ELSE IF( C2.EQ.'GB' ) THEN
    IF( C3.EQ.'TRF' ) THEN
        IF( SNAME ) THEN
            IF( N4.LE.64 ) THEN
                NB = 1
            ELSE
                NB = 32
            END IF
        ELSE
            IF( N4.LE.64 ) THEN
                NB = 1
            ELSE
                NB = 32
            END IF
        END IF
    END IF
ELSE IF( C2.EQ.'PB' ) THEN
    IF( C3.EQ.'TRF' ) THEN
        IF( SNAME ) THEN
            IF( N2.LE.64 ) THEN
                NB = 1
            ELSE
                NB = 32
            END IF
        ELSE
            IF( N2.LE.64 ) THEN
                NB = 1
            ELSE
                NB = 32
            END IF
        END IF
    END IF
ELSE IF( C2.EQ.'TR' ) THEN
    IF( C3.EQ.'TRI' ) THEN
        IF( SNAME ) THEN
            NB = 64
        ELSE
            NB = 64
        END IF
    END IF
ELSE IF( C2.EQ.'LA' ) THEN
    IF( C3.EQ.'UUM' ) THEN
        IF( SNAME ) THEN
            NB = 64
        ELSE
            NB = 64
        END IF
    END IF
ELSE IF( SNAME .AND. C2.EQ.'ST' ) THEN
    IF( C3.EQ.'EBZ' ) THEN
        NB = 1
    END IF
END IF
ILAENV = NB
RETURN
!
200 CONTINUE
!
!
ISPEC = 2: minimum block size
!
NBMIN = 2
IF( C2.EQ.'GE' ) THEN
    IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR. &
        C3.EQ.'QLF' ) THEN
        IF( SNAME ) THEN
            NBMIN = 2
        ELSE
            NBMIN = 2
        END IF
    ELSE IF( C3.EQ.'HRD' ) THEN
        IF( SNAME ) THEN
            NBMIN = 2
        ELSE
            NBMIN = 2
        END IF
    ELSE IF( C3.EQ.'BRD' ) THEN
        IF( SNAME ) THEN
            NBMIN = 2
        ELSE
            NBMIN = 2
        END IF
    ELSE IF( C3.EQ.'TRI' ) THEN
        IF( SNAME ) THEN
            NBMIN = 2
        ELSE
            NBMIN = 2
        END IF
    ELSE IF( C2.EQ.'SY' ) THEN
        IF( C3.EQ.'TRF' ) THEN
            IF( SNAME ) THEN
                NBMIN = 8
            ELSE
                NBMIN = 8
            END IF
        END IF
    END IF

```

```

        ELSE IF( SNAME .AND. C3.EQ.'TRD' ) THEN
            NBMIN = 2
        END IF
    ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
        IF( C3.EQ.'TRD' ) THEN
            NBMIN = 2
        END IF
    ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        END IF
    ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        END IF
    END IF
    ILAENV = NBMIN
    RETURN
!
300 CONTINUE
!
!   ISPEC = 3:  crossover point
!
NX = 0
IF( C2.EQ.'GE' ) THEN
    IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR. &
        C3.EQ.'QLF' ) THEN
        IF( SNAME ) THEN
            NX = 128
        ELSE
            NX = 128
        END IF
    ELSE IF( C3.EQ.'HRD' ) THEN
        IF( SNAME ) THEN
            NX = 128
        ELSE
            NX = 128
        END IF
    ELSE IF( C3.EQ.'BRD' ) THEN
        IF( SNAME ) THEN
            NX = 128
        ELSE
            NX = 128
        END IF
    ELSE IF( C2.EQ.'SY' ) THEN
        IF( SNAME .AND. C3.EQ.'TRD' ) THEN
            NX = 32
        END IF
    ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
        IF( C3.EQ.'TRD' ) THEN
            NX = 32
        END IF
    ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                C4.EQ.'BR' ) THEN
                NX = 128
            END IF
        ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
            IF( C3( 1:1 ).EQ.'G' ) THEN
                IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR. &
                    C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR. &
                    C4.EQ.'BR' ) THEN
                    NX = 128
                END IF
            END IF
        END IF
    END IF
    ILAENV = NX
    RETURN
!
400 CONTINUE
!
!   ISPEC = 4:  number of shifts (used by xHSEQR)
!
ILAENV = 6
RETURN
!
500 CONTINUE
!
!   ISPEC = 5:  minimum column dimension (not used)
!
ILAENV = 2
RETURN
!
600 CONTINUE
!
!   ISPEC = 6:  crossover point for SVD (used by xGELSS and xGESVD)
!
ILAENV = INT( REAL( MIN( N1, N2 ) ) * 1.6E0 )
RETURN
!

```

```

700 CONTINUE
!
!   ISPEC = 7:  number of processors (not used)
!
!   ILAENV = 1
!   RETURN
!
800 CONTINUE
!
!   ISPEC = 8:  crossover point for multishift (used by xHSEQR)
!
!   ILAENV = 50
!   RETURN
!
900 CONTINUE
!
!   ISPEC = 9:  maximum size of the subproblems at the bottom of the
!               computation tree in the divide-and-conquer algorithm
!               (used by xGELSD and xGESDD)
!
!   ILAENV = 25
!   RETURN
!
1000 CONTINUE
!
!   ISPEC = 10: ieee NaN arithmetic can be trusted not to trap
!
!   ILAENV = 1
!   IF (ILAENV.EQ. 1) THEN
!       ILAENV = IEEECK( 0, 0.0, 1.0 )
!   ENDIF
!   RETURN
!
1100 CONTINUE
!
!   ISPEC = 11: infinity arithmetic can be trusted not to trap
!
!   ILAENV = 1
!   IF (ILAENV.EQ. 1) THEN
!       ILAENV = IEEECK( 1, 0.0, 1.0 )
!   ENDIF
!   RETURN
!
!   End of ILAENV
!
END FUNCTION

INTEGER          FUNCTION IEEECK( ISPEC, ZERO, ONE )
!
! -- LAPACK auxiliary routine (version 3.4.0) --
! Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
! November 2006
!
! .. Scalar Arguments ..
INTEGER          ISPEC
REAL             ZERO, ONE
!
! ..
!
! Purpose
! =====
!
! IEEECK is called from the ILAENV to verify that Inifinity and
! possibly NaN arithmetic is safe (i.e. will not trap).
!
! Arguments:
! =====
!
! ISPEC      (input) INTEGER
!             Specifies whether to test just for inifinity arithmetic
!             or whether to test for infinity and NaN arithmetic.
!             = 0: Verify inifinity arithmetic only.
!             = 1: Verify infinity and NaN arithmetic.
!
! ZERO       (input) REAL
!             Must contain the value 0.0
!             This is passed to prevent the compiler from optimizing
!             away this code.
!
! ONE        (input) REAL
!             Must contain the value 1.0
!             This is passed to prevent the compiler from optimizing
!             away this code.
!
! RETURN VALUE:  INTEGER
!             = 0: Arithmetic failed to produce the correct answers
!             = 1: Arithmetic produced the correct answers
!
! .. Local Scalars ..
REAL POSINF, NEGINF, NAN1, NAN2, NAN3, NAN4, NAN5, NAN6, NEGZRO, &
NEWZRO
!
! .. Executable Statements ..
IEEECK = 1

POSINF = ONE / ZERO
IF ( POSINF .LE. ONE ) THEN
    IEEECK = 0
    RETURN
ENDIF

NEGINF = -ONE / ZERO
IF ( NEGINF .GE. ZERO ) THEN
    IEEECK = 0
    RETURN
ENDIF

NEGZRO = ONE / ( NEGINF + ONE )
IF ( NEGZRO .NE. ZERO ) THEN
    IEEECK = 0
    RETURN
ENDIF

NEGINF = ONE / NEGZRO
IF ( NEGINF .GE. ZERO ) THEN

```

```

        IEEECK = 0
        RETURN
    ENDIF

    NEWZRO = NEGZRO + ZERO
    IF ( NEWZRO .NE. ZERO ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    POSINF = ONE / NEWZRO
    IF ( POSINF .LE. ONE ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    NEGINF = NEGINF * POSINF
    IF ( NEGINF .GE. ZERO ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    POSINF = POSINF * POSINF
    IF ( POSINF .LE. ONE ) THEN
        IEEECK = 0
        RETURN
    ENDIF

!
! Return if we were only asked to check infinity arithmetic
!
    IF (ISPEC .EQ. 0 ) RETURN

    NAN1 = POSINF + NEGINF
    NAN2 = POSINF / NEGINF
    NAN3 = POSINF / POSINF
    NAN4 = POSINF * ZERO
    NAN5 = NEGINF * NEGZRO
    NAN6 = NAN5 * 0.0

    IF ( NAN1 .EQ. NAN1 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    IF ( NAN2 .EQ. NAN2 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    IF ( NAN3 .EQ. NAN3 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    IF ( NAN4 .EQ. NAN4 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    IF ( NAN5 .EQ. NAN5 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    IF ( NAN6 .EQ. NAN6 ) THEN
        IEEECK = 0
        RETURN
    ENDIF

    RETURN
END FUNCTION

SUBROUTINE DBLAT1
    *> \brief \b DBLAT1
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!   PROGRAM DBLAT1
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!>   Test program for the DOUBLE PRECISION Level 1 BLAS.
!>
!>   Based upon the original BLAS test routine together with:
!>   F06EAF Example Program Text
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date April 2012

```

```

!
!> \ingroup double_blas_testing
!
! =====
!
! -- Reference BLAS test routine (version 3.4.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! -- April 2012
!
! =====
!
! .. Parameters ..
INTEGER      NOUT
PARAMETER    (NOUT=6)
! .. Scalars in Common ..
INTEGER      ICASE, INCX, INCY, N
LOGICAL      PASS
! .. Local Scalars ..
DOUBLE PRECISION SFAC
INTEGER      IC
! .. External Subroutines ..
EXTERNAL     CHECK0, CHECK1, CHECK2, CHECK3, HEADER
! .. Common blocks ..
COMMON       /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Data statements ..
DATA         SFAC/9.765625D-4/
! .. Executable Statements ..
WRITE (NOUT,99999)
DO 20 IC = 1, 13
  ICASE = IC
  CALL HEADER
!
!   .. Initialize PASS, INCX, and INCY for a new case. ..
!   .. the value 9999 for INCX or INCY will appear in the ..
!   .. detailed output, if any, for cases that do not involve ..
!   .. these parameters ..
!
  PASS = .TRUE.
  INCX = 9999
  INCY = 9999
  IF (ICASE.EQ.3 .OR. ICASE.EQ.11) THEN
    CALL CHECK0(SFAC)
!
!   STOP
  ELSE IF (ICASE.EQ.7 .OR. ICASE.EQ.8 .OR. ICASE.EQ.9 .OR. &
    ICASE.EQ.10) THEN
    CALL CHECK1(SFAC)
  ELSE IF (ICASE.EQ.1 .OR. ICASE.EQ.2 .OR. ICASE.EQ.5 .OR. &
    ICASE.EQ.6 .OR. ICASE.EQ.12 .OR. ICASE.EQ.13) THEN
    CALL CHECK2(SFAC)
  ELSE IF (ICASE.EQ.4) THEN
    CALL CHECK3(SFAC)
  END IF
  -- Print
  IF (PASS) WRITE (NOUT,99998)
20 CONTINUE
RETURN      ! from STOP
!
99999 FORMAT (' Real BLAS Test Program Results',/1X)
99998 FORMAT ('          ----- PASS -----')
END SUBROUTINE
SUBROUTINE HEADER
! .. Parameters ..
INTEGER      NOUT
PARAMETER    (NOUT=6)
! .. Scalars in Common ..
INTEGER      ICASE, INCX, INCY, N
LOGICAL      PASS
! .. Local Arrays ..
CHARACTER*6  L(13)
! .. Common blocks ..
COMMON       /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Data statements ..
DATA         L(1)/' DDOT '/
DATA         L(2)/' DAXPY '/
DATA         L(3)/' DROTG '/
DATA         L(4)/' DROT '/
DATA         L(5)/' DCPY '/
DATA         L(6)/' DSWAP '/
DATA         L(7)/' DNRN2 '/
DATA         L(8)/' DASUM '/
DATA         L(9)/' DSCAL '/
DATA         L(10)/' IDAMAX'/
DATA         L(11)/' DROTMG'/
DATA         L(12)/' DROTM '/
DATA         L(13)/' DSDOT '/
! .. Executable Statements ..
WRITE (NOUT,99999) ICASE, L(ICASE)
RETURN
!
99999 FORMAT (' Test of subprogram number',I3,12X,A6)
END SUBROUTINE
SUBROUTINE CHECK0(SFAC)
! .. Parameters ..
INTEGER      NOUT
PARAMETER    (NOUT=6)
! .. Scalar Arguments ..
DOUBLE PRECISION SFAC
! .. Scalars in Common ..
INTEGER      ICASE, INCX, INCY, N
LOGICAL      PASS
! .. Local Scalars ..
DOUBLE PRECISION SA, SB, SC, SS, D12
INTEGER      I, K
! .. Local Arrays ..
DOUBLE PRECISION DA1(8), DATRUE(8), DB1(8), DBTRUE(8), DC1(8), &
  DS1(8), DAB(4,9), DTEMP(9), DTRUE(9,9)
! .. External Subroutines ..
EXTERNAL     DROTG, DROTMG, STEST1
! .. Common blocks ..
COMMON       /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Data statements ..
DATA         DA1/0.3D0, 0.4D0, -0.3D0, -0.4D0, -0.3D0, 0.0D0, &

```

```

DATA          0.0D0, 1.0D0/
              DB1/0.4D0, 0.3D0, 0.4D0, 0.3D0, -0.4D0, 0.0D0, &
DATA          1.0D0, 0.0D0/
              DC1/0.6D0, 0.8D0, -0.6D0, 0.8D0, 0.6D0, 1.0D0, &
DATA          0.0D0, 1.0D0/
              DS1/0.8D0, 0.6D0, 0.8D0, -0.6D0, 0.8D0, 0.0D0, &
DATA          1.0D0, 0.0D0/
              DATRUE/0.5D0, 0.5D0, -0.5D0, -0.5D0, &
DATA          0.0D0, 1.0D0, 1.0D0/
              DBTRUE/0.0D0, 0.6D0, 0.0D0, -0.6D0, 0.0D0, &
DATA          0.0D0, 1.0D0, 0.0D0/
! INPUT FOR MODIFIED GIVENs
DATA DAB/ .1D0, .3D0, 1.2D0, .2D0, &
          .7D0, .2D0, .6D0, 4.2D0, &
          0.0D0, 0.0D0, 0.0D0, 0.0D0, &
          4.0D, -1.0D, 2.0D, 4.0D, &
          6.D-10, 2.D-2, 1.D5, 10.D0, &
          4.D10, 2.D-2, 1.D-5, 10.D0, &
          2.D-10, 4.D-2, 1.D5, 10.D0, &
          2.D10, 4.D-2, 1.D-5, 10.D0, &
          4.D0, -2.D0, 8.D0, 4.D0 /
! TRUE RESULTS FOR MODIFIED GIVENs
DATA DTRUE/0.0D0, 0.0D0, 1.3D0, .2D0, 0.0D0, 0.0D0, 0.0D0, .5D0, 0.0D0, &
          0.0D0, 0.0D0, 4.5D0, 4.2D0, 1.D0, .5D0, 0.0D0, 0.0D0, 0.0D0, &
          0.0D0, 0.0D0, 0.0D0, 0.0D0, -2.D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
          0.0D0, 0.0D0, 0.0D0, 4.0D, -1.D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
          0.0D, 15.D-3, 0.0D, 10.D0, -1.D0, 0.0D, -1.D-4, &
          0.0D, 1.D0, &
          0.0D0, 0.0D, 6144.D-5, 10.D0, -1.D0, 4096.D0, -1.D6, &
          0.0D, 1.D0, &
          0.0D0, 0.0D, 15.D0, 10.D0, -1.D0, 5.D-5, 0.0D, 1.D0, 0.0D, &
          0.0D0, 0.0D, 15.D0, 10.D0, -1.D0, 5.D5, -4096.D0, &
          1.D0, 4096.D-6, &
          0.0D0, 0.0D, 7.D0, 4.D0, 0.0D0, 0.0D, -.5D0, -.25D0, 0.0D/
! 4096 = 2 ** 12
DATA D12 /4096.D0/
DTRUE(1,1) = 12.D0 / 130.D0
DTRUE(2,1) = 36.D0 / 130.D0
DTRUE(7,1) = -1.D0 / 6.D0
DTRUE(1,2) = 14.D0 / 75.D0
DTRUE(2,2) = 49.D0 / 75.D0
DTRUE(9,2) = 1.D0 / 7.D0
DTRUE(1,5) = 45.D-11 * (D12 * D12)
DTRUE(3,5) = 4.D5 / (3.D0 * D12)
DTRUE(6,5) = 1.D0 / D12
DTRUE(8,5) = 1.D4 / (3.D0 * D12)
DTRUE(1,6) = 4.D10 / (1.5D0 * D12 * D12)
DTRUE(2,6) = 2.D-2 / 1.5D0
DTRUE(8,6) = 5.D-7 * D12
DTRUE(1,7) = 4.D0 / 150.D0
DTRUE(2,7) = (2.D-10 / 1.5D0) * (D12 * D12)
DTRUE(7,7) = -DTRUE(6,5)
DTRUE(9,7) = 1.D4 / D12
DTRUE(1,8) = DTRUE(1,7)
DTRUE(2,8) = 2.D10 / (1.5D0 * D12 * D12)
DTRUE(1,9) = 32.D0 / 7.D0
DTRUE(2,9) = -16.D0 / 7.D0
! .. Executable Statements ..
!
! Compute true values which cannot be prestored
! in decimal notation
!
DBTRUE(1) = 1.0D0/0.6D0
DBTRUE(3) = -1.0D0/0.6D0
DBTRUE(5) = 1.0D0/0.6D0
!
DO 20 K = 1, 8
! .. Set N=K for identification in output if any ..
  N = K
  IF (ICASE.EQ.3) THEN
! .. DROTG ..
    IF (K.GT.8) GO TO 40
    SA = DAL(K)
! print *,sa
    SB = DB1(K)
! print *,sa,sb,sc,ss
    CALL DROTG(SA,SB,SC,SS)
! print *,sa,sb,sc,ss
! stop
    CALL STEST1(SA,DATRUE(K),DATRUE(K),SFAC)
    CALL STEST1(SB,DBTRUE(K),DBTRUE(K),SFAC)
    CALL STEST1(SC,DC1(K),DC1(K),SFAC)
    CALL STEST1(SS,DSL(K),DSL(K),SFAC)
! ELSEIF (ICASE.EQ.11) THEN
! .. DROTMG ..
    DO I=1,4
      DTEMP(I) = DAB(I,K)
      DTEMP(I+4) = 0.0
    END DO
    DTEMP(9) = 0.0
! print *,DTEMP(1)
! print *,DTEMP(2)
! print *,DTEMP(3)
! print *,DTEMP(4)
! print *,DTEMP(5)
! print *,DTEMP(6)
! print *,DTEMP(7)
! print *,DTEMP(8)
! print *,DTEMP(9)
! CALL DROTMG(DTEMP(1),DTEMP(2),DTEMP(3),DTEMP(4),DTEMP(5))
! print *,DTEMP(1)
! print *,DTEMP(2)
! print *,DTEMP(3)
! print *,DTEMP(4)
! print *,DTEMP(5)
! print *,DTEMP(6)
! print *,DTEMP(7)
! print *,DTEMP(8)
! print *,DTEMP(9)
! STOP
! CALL STEST(9,DTEMP,DTRUE(1,K),DTRUE(1,K),SFAC)
! ELSE

```

```

        WRITE (NOUT,*) ' Shouldn''t be here in CHECK0'
        STOP
    END IF
20 CONTINUE
40 RETURN
END SUBROUTINE
SUBROUTINE CHECK1(SFAC)
! .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
! .. Scalar Arguments ..
DOUBLE PRECISION SFAC
! .. Scalars in Common ..
INTEGER          ICASE, INCX, INCY, N
LOGICAL          PASS
! .. Local Scalars ..
INTEGER          I, LEN, NP1
! .. Local Arrays ..
DOUBLE PRECISION DTRUE1(5), DTRUE3(5), DTRUE5(8,5,2), DV(8,5,2), &
                SA(10), STEMP(1), STRUE(8), SX(8)
INTEGER          ITRUE2(5)
! .. External Functions ..
DOUBLE PRECISION DASUM, DNRM2
INTEGER          IDAMAX
EXTERNAL         DASUM, DNRM2, IDAMAX
! .. External Subroutines ..
EXTERNAL         ITEST1, DSCAL, STEST, STEST1
! .. Intrinsic Functions ..
INTRINSIC        MAX
! .. Common blocks ..
COMMON           /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Data statements ..
DATA            SA/0.3D0, -1.0D0, 0.0D0, 1.0D0, 0.3D0, 0.3D0, &
                0.3D0, 0.3D0, 0.3D0, 0.3D0/
DATA            DV/0.1D0, 2.0D0, 2.0D0, 2.0D0, 2.0D0, 2.0D0, &
                2.0D0, 2.0D0, 0.3D0, 3.0D0, 3.0D0, 3.0D0, 3.0D0, &
                3.0D0, 3.0D0, 3.0D0, 0.3D0, -0.4D0, 4.0D0, &
                4.0D0, 4.0D0, 4.0D0, 4.0D0, 4.0D0, 0.2D0, &
                -0.6D0, 0.3D0, 5.0D0, 5.0D0, 5.0D0, 5.0D0, &
                5.0D0, 0.1D0, -0.3D0, 0.5D0, -0.1D0, 6.0D0, &
                6.0D0, 6.0D0, 6.0D0, 0.1D0, 8.0D0, 8.0D0, 8.0D0, &
                8.0D0, 8.0D0, 8.0D0, 8.0D0, 0.3D0, 9.0D0, 9.0D0, &
                9.0D0, 9.0D0, 9.0D0, 9.0D0, 9.0D0, 0.3D0, 2.0D0, &
                -0.4D0, 2.0D0, 2.0D0, 2.0D0, 2.0D0, 2.0D0, &
                0.2D0, 3.0D0, -0.6D0, 5.0D0, 0.3D0, 2.0D0, &
                2.0D0, 2.0D0, 0.1D0, 4.0D0, -0.3D0, 6.0D0, &
                -0.5D0, 7.0D0, -0.1D0, 3.0D0/
DATA            DTRUE1/0.0D0, 0.3D0, 0.5D0, 0.7D0, 0.6D0/
DATA            DTRUE3/0.0D0, 0.3D0, 0.7D0, 1.1D0, 1.0D0/
DATA            DTRUE5/0.1D0, 2.0D0, 2.0D0, 2.0D0, 2.0D0, &
                2.0D0, 2.0D0, 2.0D0, -0.3D0, 3.0D0, 3.0D0, &
                3.0D0, 3.0D0, 3.0D0, 3.0D0, 3.0D0, 0.0D0, 0.0D0, &
                4.0D0, 4.0D0, 4.0D0, 4.0D0, 4.0D0, 4.0D0, &
                0.2D0, -0.6D0, 0.3D0, 5.0D0, 5.0D0, 5.0D0, &
                5.0D0, 5.0D0, 0.03D0, -0.09D0, 0.15D0, -0.03D0, &
                6.0D0, 6.0D0, 6.0D0, 6.0D0, 0.1D0, 8.0D0, &
                8.0D0, 8.0D0, 8.0D0, 8.0D0, 8.0D0, 8.0D0, &
                0.09D0, 9.0D0, 9.0D0, 9.0D0, 9.0D0, 9.0D0, &
                9.0D0, 9.0D0, 0.09D0, 2.0D0, -0.12D0, 2.0D0, &
                2.0D0, 2.0D0, 2.0D0, 2.0D0, 0.06D0, 3.0D0, &
                -0.18D0, 5.0D0, 0.09D0, 2.0D0, 2.0D0, 2.0D0, &
                0.03D0, 4.0D0, -0.09D0, 6.0D0, -0.15D0, 7.0D0, &
                -0.03D0, 3.0D0/
DATA            ITRUE2/0, 1, 2, 2, 3/
! .. Executable Statements ..
DO 80 INCX = 1, 2
    DO 60 NP1 = 1, 5
        N = NP1 - 1
        LEN = 2*MAX(N,1)
        ! .. Set vector arguments ..
        DO 20 I = 1, LEN
            SX(I) = DV(I,NP1,INCX)
20        CONTINUE
        IF (ICASE.EQ.7) THEN
            ! .. DNRM2 ..
            STEMP(1) = DTRUE1(NP1)
            CALL STEST1(DNRM2(N,SX,INCX),STEMP(1),STEMP,SFAC)
        ELSE IF (ICASE.EQ.8) THEN
            ! .. DASUM ..
            STEMP(1) = DTRUE3(NP1)
            CALL STEST1(DASUM(N,SX,INCX),STEMP(1),STEMP,SFAC)
        ELSE IF (ICASE.EQ.9) THEN
            ! .. DSCAL ..
            CALL DSCAL(N,SA((INCX-1)*5+NP1),SX,INCX)
            DO 40 I = 1, LEN
                STRUE(I) = DTRUE5(I,NP1,INCX)
40            CONTINUE
            CALL STEST(LEN,SX,STRUE,STEMP,SFAC)
        ELSE IF (ICASE.EQ.10) THEN
            ! .. IDAMAX ..
            CALL ITEST1(IDAMAX(N,SX,INCX),ITRUE2(NP1))
        ELSE
            WRITE (NOUT,*) ' Shouldn''t be here in CHECK1'
            STOP
        END IF
    END IF
60 CONTINUE
80 CONTINUE
RETURN
END SUBROUTINE
SUBROUTINE CHECK2(SFAC)
! .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
! .. Scalar Arguments ..
DOUBLE PRECISION SFAC
! .. Scalars in Common ..
INTEGER          ICASE, INCX, INCY, N
LOGICAL          PASS
! .. Local Scalars ..
DOUBLE PRECISION SA
INTEGER          I, J, KI, KN, KNI, KPAR, KSIZE, LENX, LENY, &
                MX, MY
! .. Local Arrays ..

```



```

- .9D0, 2.8D0, -1.4D0, -1.3D0, 0.D0, 0.D0, 0.D0, &
3.5D0, -.4D0, -2.2D0, 4.7D0, 0.D0, 0.D0, 0.D0/

!
DATA DT19XB/.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.9D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, -.5D0, 0.D0, 0.D0, 0.D0, 0.D0, &
0.D0, .1D0, -3.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.3D0, .1D0, -2.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.3D0, .1D0, -2.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, -.5D0, .8D0, .9D0, -.3D0, -.4D0, &
-2.0D0, .1D0, 1.4D0, .8D0, .6D0, -.3D0, -2.8D0, &
-1.8D0, .1D0, 1.3D0, .8D0, 0.D0, -.3D0, -1.9D0, &
3.8D0, .1D0, -3.1D0, .8D0, 4.8D0, -.3D0, -1.5D0 /

!
DATA DT19XC/.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.9D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, -.5D0, 0.D0, 0.D0, 0.D0, 0.D0, &
4.8D0, .1D0, -3.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.3D0, .1D0, -2.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
2.1D0, .1D0, -2.0D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, -.5D0, .8D0, .9D0, -.3D0, -.4D0, &
-1.6D0, .1D0, -2.2D0, .8D0, 5.4D0, -.3D0, -2.8D0, &
-1.5D0, .1D0, -1.4D0, .8D0, 3.6D0, -.3D0, -1.9D0, &
3.7D0, .1D0, -2.2D0, .8D0, 3.6D0, -.3D0, -1.5D0 /

!
DATA DT19XD/.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.9D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.8D0, -1.0D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.9D0, -.8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
3.5D0, .8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.6D0, .1D0, -.5D0, .8D0, 0.D0, 0.D0, 0.D0, &
-.8D0, -1.0D0, 1.4D0, -1.6D0, 0.D0, 0.D0, 0.D0, &
-.9D0, -.8D0, 1.3D0, -1.6D0, 0.D0, 0.D0, 0.D0, &
3.5D0, .8D0, -3.1D0, 4.8D0, 0.D0, 0.D0, 0.D0/

!
TRUE Y RESULTS FOR ROTATIONS DROT M
DATA DT19YA/.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
1.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.7D0, -4.8D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
1.7D0, -.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 3.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, .3D0, .7D0, 0.D0, 0.D0, 0.D0, &
.7D0, -4.8D0, 3.0D0, 1.1D0, 0.D0, 0.D0, 0.D0, &
1.7D0, -.7D0, -.7D0, 2.3D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 3.5D0, -.7D0, -3.6D0, 0.D0, 0.D0, 0.D0/

!
DATA DT19YB/.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
1.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, .3D0, .7D0, 0.D0, 0.D0, 0.D0, &
4.0D0, -.9D0, -.3D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.5D0, -.9D0, 1.5D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-1.5D0, -.9D0, -1.8D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, .3D0, .7D0, -.6D0, .2D0, .8D0, &
3.7D0, -.9D0, -1.2D0, .7D0, -1.5D0, .2D0, 2.2D0, &
-.3D0, -.9D0, 2.1D0, .7D0, -1.6D0, .2D0, 2.0D0, &
-1.6D0, -.9D0, -2.1D0, .7D0, 2.9D0, .2D0, -3.8D0 /

!
DATA DT19YC/.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
1.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
4.0D0, -6.3D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-.5D0, -.3D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-1.5D0, 3.0D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, .3D0, .7D0, 0.D0, 0.D0, 0.D0, &
3.7D0, -7.2D0, 3.0D0, 1.7D0, 0.D0, 0.D0, 0.D0, &
-.3D0, .9D0, -.7D0, 1.9D0, 0.D0, 0.D0, 0.D0, &
-1.6D0, 2.7D0, -.7D0, -3.4D0, 0.D0, 0.D0, 0.D0/

!
DATA DT19YD/.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
1.7D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
-2.6D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, 0.D0, &
.5D0, -.9D0, .3D0, .7D0, 0.D0, 0.D0, 0.D0, &

```

```

      .7D0,   -.9D0,   1.2D0,           0.D0,0.D0,0.D0,0.D0, &
      1.7D0,   -.9D0,   .5D0,           0.D0,0.D0,0.D0,0.D0, &
     -2.6D0,   -.9D0,  -1.3D0,           0.D0,0.D0,0.D0,0.D0, &
      .5D0,   -.9D0,   .3D0,   .7D0,   -.6D0,   .2D0,   .8D0, &
      .7D0,   -.9D0,   1.2D0,   .7D0,  -1.5D0,   .2D0,   1.6D0, &
      1.7D0,   -.9D0,   .5D0,   .7D0,  -1.6D0,   .2D0,   2.4D0, &
     -2.6D0,   -.9D0,  -1.3D0,   .7D0,   2.9D0,   .2D0,  -4.0D0 /
!
! .. Executable Statements ..
!
INTEGER II,JJ,KK
DO II=1,7
DO JJ=1,4
DO KK=1,4
!   write(6,1017) II,JJ,KK,DT19X(II,JJ,KK),DT19Y(II,JJ,KK)
END DO
END DO
ENDDO
! STOP
1017 FORMAT (I3,I3,I3,2F12.3)
DO 120 KI = 1, 4
  INCX = INCXS(KI)
  INCY = INCYS(KI)
  MX = ABS(INCX)
  MY = ABS(INCY)
!
DO 100 KN = 1, 4
  N = NS(KN)
  KSIZE = MIN(2,KN)
  LENX = LENS(KN,MX)
  LENY = LENS(KN,MY)
!   .. Initialize all argument arrays ..
DO 20 I = 1, 7
  SX(I) = DX1(I)
  SY(I) = DY1(I)
20 CONTINUE
!
IF (ICASE.EQ.1) THEN
!   .. DDOT ..
CALL STEST1(DDOT(N,SX,INCX,SY,INCY),DT7(KN,KI),SSIZE1(KN),SFAC)
ELSE IF (ICASE.EQ.2) THEN
!   .. DAXPY ..
CALL DAXPY(N,SA,SX,INCX,SY,INCY)
DO 40 J = 1, LENY
  STY(J) = DT8(J,KN,KI)
40 CONTINUE
CALL STEST(LENY,SY,STY,SSIZE2(1,KSIZE),SFAC)
ELSE IF (ICASE.EQ.5) THEN
!   .. DCOPY ..
DO 60 I = 1, 7
  STY(I) = DT10Y(I,KN,KI)
60 CONTINUE
CALL DCOPY(N,SX,INCX,SY,INCY)
CALL STEST(LENY,SY,STY,SSIZE2(1,1),1.0D0)
ELSE IF (ICASE.EQ.6) THEN
!   .. DSWAP ..
CALL DSWAP(N,SX,INCX,SY,INCY)
DO 80 I = 1, 7
  STX(I) = DT10X(I,KN,KI)
  STY(I) = DT10Y(I,KN,KI)
80 CONTINUE
CALL STEST(LENX,SX,STX,SSIZE2(1,1),1.0D0)
CALL STEST(LENY,SY,STY,SSIZE2(1,1),1.0D0)
ELSE IF (ICASE.EQ.12) THEN
!   .. DROTM ...
KNI=KN+4*(KI-1)
!   print *, 'start kpar 1,4 I=1,7'
DO KPAR=1,4
DO I=1,7
  SX(I) = DX1(I)
  SY(I) = DY1(I)
  STX(I) = DT19X(I,KPAR,KNI)
  STY(I) = DT19Y(I,KPAR,KNI)
END DO
!
DO I=1,5
  DTEMP(I) = DPAR(I,KPAR)
END DO
!
DO I=1,LENX
  SSIZE(I)=STX(I)
END DO
!   SEE REMARK ABOVE ABOUT DT11X(1,2,7)
!   AND DT11X(5,3,8).
IF ((KPAR .EQ. 2) .AND. (KNI .EQ. 7)) SSIZE(1) = 2.4D0
IF ((KPAR .EQ. 3) .AND. (KNI .EQ. 8)) SSIZE(5) = 1.8D0
!
print *, 'here in CHECK2',N
DO I=1,7
!   print *,I,SX(I),SY(I)
END DO
!   CALL DROTM(N,SX,INCX,SY,INCY,DTEMP)
!   print *, 'aft'
DO I=1,7
!   print *,I,SX(I),SY(I)
END DO
!   STOP
DO I=1,LENX
!   write(*,1018) KPAR,I,SX(I),STX(I),SSIZE(I),SFAC
END DO
CALL STEST(LENX,SX,STX,SSIZE,SFAC)
CALL STEST(LENY,SY,STY,STY,SFAC)
!
END DO
! STOP
ELSE IF (ICASE.EQ.13) THEN
!   .. DSDOT ..
CALL TESTDSDOT(REAL(DSDOT(N,REAL(SX),INCX,REAL(SY),INCY)),
REAL(DT7(KN,KI)),REAL(SSIZE1(KN)),.3125E-1)
ELSE
  WRITE(NOUT,*) ' Shouldn''t be here in CHECK2'
  STOP
END IF
100 CONTINUE

```

```

120 CONTINUE
1018 FORMAT (2I3,4F12.3)
RETURN
END SUBROUTINE
SUBROUTINE CHECK3(SFAC)
! .. Parameters ..
INTEGER      NOUT
PARAMETER    (NOUT=6)
! .. Scalar Arguments ..
DOUBLE PRECISION SFAC
! .. Scalars in Common ..
INTEGER      ICASE, INCX, INCY, N
LOGICAL      PASS
! .. Local Scalars ..
DOUBLE PRECISION SC, SS
INTEGER      I, K, KI, KN, KSIZE, LENX, LENY, MX, MY
! .. Local Arrays ..
DOUBLE PRECISION COPYX(5), COPYY(5), DT9X(7,4,4), DT9Y(7,4,4), &
DX1(7), DY1(7), MWPC(11), MWPS(11), MWPSTX(5), &
MWPSTY(5), MWPTX(11,5), MWPTY(11,5), MWFX(5), &
MWPY(5), SSIZE2(14,2), STX(7), STY(7), SX(7), &
SY(7)
INTEGER      INCXS(4), INCYS(4), LENS(4,2), MWPINX(11), &
MWPINY(11), MWPN(11), NS(4)
! .. External Subroutines ..
EXTERNAL     DROT, STEST
! .. Intrinsic Functions ..
INTRINSIC    ABS, MIN
! .. Common blocks ..
COMMON       /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Data statements ..
DATA         INCXS/1, 2, -2, -1/
DATA         INCYS/1, -2, 1, -2/
DATA         LENS/1, 1, 2, 4, 1, 1, 3, 7/
DATA         NS/0, 1, 2, 4/
DATA         DX1/0.6D0, 0.1D0, -0.5D0, 0.8D0, 0.9D0, -0.3D0,      -0.4D0/
DATA         DY1/0.5D0, -0.9D0, 0.3D0, 0.7D0, -0.6D0, 0.2D0,      0.8D0/
DATA         SC, SS/0.8D0, 0.6D0/
DATA         DT9X/0.6D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.78D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.78D0, -0.46D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.78D0, -0.46D0, -0.22D0, &
1.06D0, 0.0D0, 0.0D0, 0.0D0, 0.6D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.78D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.66D0, 0.1D0, -0.1D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.96D0, 0.1D0, -0.76D0, 0.8D0, 0.90D0, &
-0.3D0, -0.02D0, 0.6D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.78D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, -0.06D0, 0.1D0, &
-0.1D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.90D0, &
0.1D0, -0.22D0, 0.8D0, 0.18D0, -0.3D0, -0.02D0, &
0.6D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.78D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.78D0, 0.26D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.78D0, 0.26D0, -0.76D0, 1.12D0, &
0.0D0, 0.0D0, 0.0D0/
DATA         DT9Y/0.5D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.04D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.04D0, -0.78D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.04D0, -0.78D0, 0.54D0, &
0.08D0, 0.0D0, 0.0D0, 0.0D0, 0.5D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.04D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.7D0, &
-0.9D0, -0.12D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.64D0, -0.9D0, -0.3D0, 0.7D0, -0.18D0, 0.2D0, &
0.28D0, 0.5D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.04D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.7D0, -1.08D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.64D0, -1.26D0, &
0.54D0, 0.2D0, 0.0D0, 0.0D0, 0.0D0, 0.5D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.04D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.04D0, -0.9D0, 0.18D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.04D0, -0.9D0, 0.18D0, 0.7D0, &
-0.18D0, 0.2D0, 0.16D0/
DATA         SSIZE2/0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, &
0.0D0, 1.17D0, 1.17D0, 1.17D0, 1.17D0, 1.17D0, &
1.17D0, 1.17D0, 1.17D0, 1.17D0, 1.17D0, 1.17D0, &
1.17D0, 1.17D0, 1.17D0/
! .. Executable Statements ..
DO 60 KI = 1, 4
  INCX = INCXS(KI)
  INCY = INCYS(KI)
  MX = ABS(INCX)
  MY = ABS(INCY)
  DO 40 KN = 1, 4
    N = NS(KN)
    KSIZE = MIN(2,KN)
    LENX = LENS(KN,MX)
    LENY = LENS(KN,MY)
    IF (ICASE.EQ.4) THEN
      .. DROT ..
      DO 20 I = 1, 7
        SX(I) = DX1(I)
        SY(I) = DY1(I)
        STX(I) = DT9X(I,KN,KI)
        STY(I) = DT9Y(I,KN,KI)
20      CONTINUE
      CALL DROT(N,SX,INCX,SY,INCY,SC,SS)
      CALL STEST(LENX,SX,STX,SSIZE2(1,KSIZE),SFAC)
      CALL STEST(LENY,SY,STY,SSIZE2(1,KSIZE),SFAC)
    ELSE
      WRITE (NOUT,*) ' Shouldn''t be here in CHECK3'
      STOP
    END IF
  CONTINUE
40 CONTINUE
60 CONTINUE
!
MWPC(1) = 1

```

```

      DO 80 I = 2, 11
        MWPC(I) = 0
80    CONTINUE
      MWPS(1) = 0
      DO 100 I = 2, 6
        MWPS(I) = 1
100   CONTINUE
      DO 120 I = 7, 11
        MWPS(I) = -1
120   CONTINUE
      MWPINX(1) = 1
      MWPINX(2) = 1
      MWPINX(3) = 1
      MWPINX(4) = -1
      MWPINX(5) = 1
      MWPINX(6) = -1
      MWPINX(7) = 1
      MWPINX(8) = 1
      MWPINX(9) = -1
      MWPINX(10) = 1
      MWPINX(11) = -1
      MWPINY(1) = 1
      MWPINY(2) = 1
      MWPINY(3) = -1
      MWPINY(4) = -1
      MWPINY(5) = 2
      MWPINY(6) = 1
      MWPINY(7) = 1
      MWPINY(8) = -1
      MWPINY(9) = -1
      MWPINY(10) = 2
      MWPINY(11) = 1
      DO 140 I = 1, 11
        MWPN(I) = 5
140   CONTINUE
      MWPN(5) = 3
      MWPN(10) = 3
      DO 160 I = 1, 5
        MWPTX(I) = I
        MWPTY(I) = I
        MWPTX(1,I) = I
        MWPTY(1,I) = I
        MWPTX(2,I) = I
        MWPTY(2,I) = -I
        MWPTX(3,I) = 6 - I
        MWPTY(3,I) = I - 6
        MWPTX(4,I) = I
        MWPTY(4,I) = -I
        MWPTX(6,I) = 6 - I
        MWPTY(6,I) = I - 6
        MWPTX(7,I) = -I
        MWPTY(7,I) = I
        MWPTX(8,I) = I - 6
        MWPTY(8,I) = 6 - I
        MWPTX(9,I) = -I
        MWPTY(9,I) = I
        MWPTX(11,I) = I - 6
        MWPTY(11,I) = 6 - I
160   CONTINUE
      MWPTX(5,1) = 1
      MWPTX(5,2) = 3
      MWPTX(5,3) = 5
      MWPTX(5,4) = 4
      MWPTX(5,5) = 5
      MWPTY(5,1) = -1
      MWPTY(5,2) = 2
      MWPTY(5,3) = -2
      MWPTY(5,4) = 4
      MWPTY(5,5) = -3
      MWPTX(10,1) = -1
      MWPTX(10,2) = -3
      MWPTX(10,3) = -5
      MWPTX(10,4) = 4
      MWPTX(10,5) = 5
      MWPTY(10,1) = 1
      MWPTY(10,2) = 2
      MWPTY(10,3) = 2
      MWPTY(10,4) = 4
      MWPTY(10,5) = 3
      DO 200 I = 1, 11
        INCX = MWPINX(I)
        INCY = MWPINY(I)
        DO 180 K = 1, 5
          COPYX(K) = MWPTX(K)
          COPYY(K) = MWPTY(K)
          MWPSTX(K) = MWPTX(I,K)
          MWPSTY(K) = MWPTY(I,K)
180    CONTINUE
      CALL DROT(MWPN(I), COPYX, INCX, COPYY, INCY, MWPC(I), MWPS(I))
      CALL STEST(5, COPYX, MWPSTX, MWPSTY, SFAC)
      CALL STEST(5, COPYY, MWPSTY, MWPSTY, SFAC)
200   CONTINUE
      RETURN
      END SUBROUTINE
      SUBROUTINE STEST(LEN,SCOMP,STRUE,SSIZE,SFAC)
      ***** STEST *****
      !
      ! THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO
      ! SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
      ! NEGLIGIBLE.
      !
      ! C. L. LAWSON, JPL, 1974 DEC 10
      !
      ! .. Parameters ..
      INTEGER          NOUT
      DOUBLE PRECISION ZERO
      PARAMETER        (NOUT=6, ZERO=0.0D0)
      ! .. Scalar Arguments ..
      DOUBLE PRECISION SFAC
      INTEGER          LEN
      ! .. Array Arguments ..
      DOUBLE PRECISION SCOMP(LEN), SSIZE(LEN), STRUE(LEN)
      ! .. Scalars in Common ..
      INTEGER          ICASE, INCX, INCY, N

```

```

LOGICAL      PASS
! .. Local Scalars ..
DOUBLE PRECISION SD
INTEGER      I
! .. External Functions ..
DOUBLE PRECISION SDIFF
EXTERNAL     SDIFF
! .. Intrinsic Functions ..
INTRINSIC   ABS
! .. Common blocks ..
COMMON      /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Executable Statements ..
!
DO I=1,LEN
!   write(*,1021),I,SCOMP(I),STRUE(I),SSIZE(I),SFAC,LEN
END DO

DO 40 I = 1, LEN
  SD = SCOMP(I) - STRUE(I)
  IF (ABS(SFAC*SD) .LE. ABS(SSIZE(I))*EPSILON(ZERO)) GO TO 40
!
!   HERE      SCOMP(I) IS NOT CLOSE TO STRUE(I).
!
  IF ( .NOT. PASS) GO TO 20
!   PRINT FAIL MESSAGE AND HEADER.
  PASS = .FALSE.
  WRITE (NOUT,99999)
  WRITE (NOUT,99998)
20  WRITE (NOUT,99997) ICASE, N, INCX, INCY, I, SCOMP(I), STRUE(I), SD, SSIZE(I)
40 CONTINUE
RETURN
!
1021 FORMAT (I3,4F12.3,I3)
99999 FORMAT ('          FAIL')
99998 FORMAT ('/' CASE N INCX INCY I          ', &
' COMP(I)          TRUE(I) DIFFERENCE', &
' SIZE(I)',/1X)
99997 FORMAT (1X,I4,I3,2I5,I3,2D36.8,2D12.4)
END SUBROUTINE
SUBROUTINE TESTSDOT(SCOMP,STRUE,SSIZE,SFAC)
***** TEST *****
!
! THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO
! SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
! NEGLIGIBLE.
!
! C. L. LAWSON, JPL, 1974 DEC 10
!
! .. Parameters ..
INTEGER      NOUT
REAL         ZERO
PARAMETER    (NOUT=6, ZERO=0.0E0)
! .. Scalar Arguments ..
REAL         SFAC, SCOMP, SSIZE, STRUE
! .. Scalars in Common ..
INTEGER      ICASE, INCX, INCY, N
LOGICAL      PASS
! .. Local Scalars ..
REAL         SD
! .. Intrinsic Functions ..
INTRINSIC   ABS
! .. Common blocks ..
COMMON      /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Executable Statements ..
!
  SD = SCOMP - STRUE
  IF (ABS(SFAC*SD) .LE. ABS(SSIZE) * EPSILON(ZERO)) GO TO 40
!
!   HERE      SCOMP(I) IS NOT CLOSE TO STRUE(I).
!
  IF ( .NOT. PASS) GO TO 20
!   PRINT FAIL MESSAGE AND HEADER.
  PASS = .FALSE.
  WRITE (NOUT,99999)
  WRITE (NOUT,99998)
20  WRITE (NOUT,99997) ICASE, N, INCX, INCY, SCOMP, STRUE, SD, SSIZE
40 CONTINUE
RETURN
!
99999 FORMAT ('          FAIL')
99998 FORMAT ('/' CASE N INCX INCY          ', &
' COMP(I)          TRUE(I) DIFFERENCE', &
' SIZE(I)',/1X)
99997 FORMAT (1X,I4,I3,I15,I3,2E36.8,2E12.4)
END SUBROUTINE
SUBROUTINE TEST1(SCOMP1,STRUE1,SSIZE,SFAC)
***** TEST1 *****
!
! THIS IS AN INTERFACE SUBROUTINE TO ACCOMODATE THE FORTRAN
! REQUIREMENT THAT WHEN A DUMMY ARGUMENT IS AN ARRAY, THE
! ACTUAL ARGUMENT MUST ALSO BE AN ARRAY OR AN ARRAY ELEMENT.
!
! C.L. LAWSON, JPL, 1978 DEC 6
!
! .. Scalar Arguments ..
DOUBLE PRECISION SCOMP1, SFAC, STRUE1
! .. Array Arguments ..
DOUBLE PRECISION SSIZE(*)
! .. Local Arrays ..
DOUBLE PRECISION SCOMP(1), STRUE(1)
! .. External Subroutines ..
EXTERNAL     TEST
! .. Executable Statements ..
!
  SCOMP(1) = SCOMP1
  STRUE(1) = STRUE1
  CALL TEST(1,SCOMP,STRUE,SSIZE,SFAC)
!
RETURN
END SUBROUTINE
DOUBLE PRECISION FUNCTION SDIFF(SA,SB)
***** SDIFF *****
! COMPUTES DIFFERENCE OF TWO NUMBERS. C. L. LAWSON, JPL 1974 FEB 15

```

```

!
! .. Scalar Arguments ..
DOUBLE PRECISION          SA, SB
! .. Executable Statements ..
SDIFF = SA - SB
RETURN
END FUNCTION

SUBROUTINE ITEST1(ICOMP,ITRUE)
***** ITEST1 *****
!
! THIS SUBROUTINE COMPARES THE VARIABLES ICOMP AND ITRUE FOR
! EQUALITY.
! C. L. LAWSON, JPL, 1974 DEC 10
!
! .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
! .. Scalar Arguments ..
INTEGER          ICOMP, ITRUE
! .. Scalars in Common ..
INTEGER          ICASE, INCX, INCY, N
LOGICAL          PASS
! .. Local Scalars ..
INTEGER          ID
! .. Common blocks ..
COMMON           /COMBLA/ICASE, N, INCX, INCY, PASS
! .. Executable Statements ..

IF (ICOMP.EQ.ITRUE) GO TO 40

!
!             HERE ICOMP IS NOT EQUAL TO ITRUE.
!
IF ( .NOT. PASS) GO TO 20
PRINT FAIL MESSAGE AND HEADER.
PASS = .FALSE.
WRITE (NOUT,99999)
WRITE (NOUT,99998)
20 ID = ICOMP - ITRUE
WRITE (NOUT,99997) ICASE, N, INCX, INCY, ICOMP, ITRUE, ID
40 CONTINUE
RETURN
!
99999 FORMAT ('                FAIL')
99998 FORMAT ('/ CASE N INCX INCY          ', &
' COMP                TRUE    DIFFERENCE', &
'/IX)')
99997 FORMAT (1X,I4,I3,2I5,2I36,I12)
END SUBROUTINE

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DATA          L(1)/' DDOT '/'
! DATA          L(2)/'DAXPY '/'
! DATA          L(3)/'DROTG '/'
! DATA          L(4)/' DROT '/'
! DATA          L(5)/'DCOPY '/'
! DATA          L(6)/'DSWAP '/'
! DATA          L(7)/'DNRM2 '/'
! DATA          L(8)/'DASUM '/'
! DATA          L(9)/'DSCAL '/'
! DATA          L(10)/'IDAMAX'/'
! DATA          L(11)/'DROTMG'/'
! DATA          L(12)/'DROTM '/'
! DATA          L(13)/'DSDOT '/'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
!> \brief \b DDOT
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!   DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,INCY,N
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION DX(*),DY(*)
!   ..
!
!> \par Purpose:
! =====
!> \verbatim
!> DDOT forms the dot product of two vectors.
!> uses unrolled loops for increments equal to one.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!> jack dongarra, linpack, 3/11/78.

```

```

!>      modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
!
=====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! -- November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX, INCY, N
!
! .. Array Arguments ..
! DOUBLE PRECISION DX(*), DY(*)
! ..
!
=====
!
! .. Local Scalars ..
! DOUBLE PRECISION DTEMP
! INTEGER I, IX, IY, M, MP1
!
! .. Intrinsic Functions ..
! INTRINSIC MOD
! ..
! DDOT = 0.0d0
! DTEMP = 0.0d0
! IF (N.LE.0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!     code for both increments equal to 1
!
!
!     clean-up loop
!
!     M = MOD(N,5)
!     IF (M.NE.0) THEN
!       DO I = 1, M
!         DTEMP = DTEMP + DX(I)*DY(I)
!       END DO
!     IF (N.LT.5) THEN
!       DDOT=DTEMP
!     RETURN
!   END IF
!   END IF
!   MP1 = M + 1
!   DO I = MP1, N, 5
!     DTEMP = DTEMP + DX(I)*DY(I) + DX(I+1)*DY(I+1) + &
!       DX(I+2)*DY(I+2) + DX(I+3)*DY(I+3) + DX(I+4)*DY(I+4)
!   END DO
! ELSE
!
!     code for unequal increments or equal increments
!     not equal to 1
!
!     IX = 1
!     IY = 1
!     IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!     IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!     DO I = 1, N
!       DTEMP = DTEMP + DX(IX)*DY(IY)
!       IX = IX + INCX
!       IY = IY + INCY
!     END DO
!   END IF
!   DDOT = DTEMP
!   RETURN
! END FUNCTION
! SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
!> \brief \b DAXPY
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION DA
! INTEGER INCX, INCY, N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*), DY(*)
! ..
!
!> \par Purpose:
!> =====
!> \endverbatim
!>
!> DAXPY constant times a vector plus a vector.
!> uses unrolled loops for increments equal to one.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:

```

```

! =====
!>
!> \verbatim
!>
!>     jack dongarra, linpack, 3/11/78.
!>     modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION DA
! INTEGER INCX, INCY, N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*), DY(*)
! ..
! =====
!
! .. Local Scalars ..
! INTEGER I, IX, IY, M, MP1
! ..
! .. Intrinsic Functions ..
! INTRINSIC MOD
! ..
! IF (N.LE.0) RETURN
! IF (DA.EQ.0.0d0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!     code for both increments equal to 1
!
!
!     clean-up loop
!
!     M = MOD(N,4)
!     IF (M.NE.0) THEN
!       DO I = 1,M
!         DY(I) = DY(I) + DA*DX(I)
!       END DO
!     END IF
!     IF (N.LT.4) RETURN
!     MP1 = M + 1
!     DO I = MP1,N,4
!       DY(I) = DY(I) + DA*DX(I)
!       DY(I+1) = DY(I+1) + DA*DX(I+1)
!       DY(I+2) = DY(I+2) + DA*DX(I+2)
!       DY(I+3) = DY(I+3) + DA*DX(I+3)
!     END DO
! ELSE
!
!     code for unequal increments or equal increments
!     not equal to 1
!
!     IX = 1
!     IY = 1
!     IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!     IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!     DO I = 1,N
!       DY(IY) = DY(IY) + DA*DX(IX)
!       IX = IX + INCX
!       IY = IY + INCY
!     END DO
! END IF
! RETURN
! END SUBROUTINE
! SUBROUTINE DROTG(DA,DB,C,S)
!> \brief \b DROTG
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DROTG(DA,DB,C,S)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION C,DA,DB,S
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!>     DROTG construct gives plane rotation.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!>
!> \verbatim

```

```

!>
!>      jack dongarra, linpack, 3/11/78.
!> \endverbatim
!>
!> =====
!
! -- Reference BLAS levell routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! -- November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION C,DA,DB,S
! ..
!
! =====
!
! .. Local Scalars ..
! DOUBLE PRECISION R,ROE,SCALE,Z
! ..
! .. Intrinsic Functions ..
! INTRINSIC DABS,DSIGN,DSQRT
! ..
! ROE = DB
! IF (DABS(DA).GT.DABS(DB)) ROE = DA
! SCALE = DABS(DA) + DABS(DB)
! IF (SCALE.EQ.0.0d0) THEN
!   C = 1.0d0
!   S = 0.0d0
!   R = 0.0d0
!   Z = 0.0d0
! ELSE
!   R = SCALE*DSQRT((DA/SCALE)**2+ (DB/SCALE)**2)
!   R = DSIGN(1.0d0,ROE)*R
!   C = DA/R
!   S = DB/R
!   Z = 1.0d0
!   IF (DABS(DA).GT.DABS(DB)) Z = S
!   IF (DABS(DB).GE.DABS(DA) .AND. C.NE.0.0d0) Z = 1.0d0/C
! END IF
! DA = R
! DB = Z
! RETURN
! END SUBROUTINE
! SUBROUTINE DROT(N,DX,INCX,DY,INCY,C,S)
!> \brief \b DROT
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DROT(N,DX,INCX,DY,INCY,C,S)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION C,S
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*),DY(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!>      DROT applies a plane rotation.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_levell
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!>      jack dongarra, linpack, 3/11/78.
!>      modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
!> =====
!
! -- Reference BLAS levell routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! -- November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION C,S
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*),DY(*)
! ..
!
! =====
!
!>

```

```

! .. Local Scalars ..
DOUBLE PRECISION DTEMP
INTEGER I,IX,IY
!
! ..
! IF (N.LE.0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!     code for both increments equal to 1
!
!     DO I = 1,N
!         DTEMP = C*DX(I) + S*DY(I)
!         DY(I) = C*DY(I) - S*DX(I)
!         DX(I) = DTEMP
!     END DO
! ELSE
!
!     code for unequal increments or equal increments not equal
!     to 1
!
!     IX = 1
!     IY = 1
!     IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!     IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!     DO I = 1,N
!         DTEMP = C*DX(IX) + S*DY(IY)
!         DY(IY) = C*DY(IY) - S*DX(IX)
!         DX(IX) = DTEMP
!         IX = IX + INCX
!         IY = IY + INCY
!     END DO
! END IF
! RETURN
! END SUBROUTINE
SUBROUTINE DCOPY(N,DX,INCX,DY,INCY)
!> \brief \b DCOPY
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DCOPY(N,DX,INCX,DY,INCY)
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*),DY(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!>   DCOPY copies a vector, x, to a vector, y.
!>   uses unrolled loops for increments equal to one.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!>   jack dongarra, linpack, 3/11/78.
!>   modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
DOUBLE PRECISION DX(*),DY(*)
! ..
! =====
!
! .. Local Scalars ..
INTEGER I,IX,IY,M,MP1
! ..
! .. Intrinsic Functions ..
INTRINSIC MOD
! ..
! IF (N.LE.0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!     code for both increments equal to 1
!
!
!
!     clean-up loop

```

```

!
      M = MOD(N,7)
      IF (M.NE.0) THEN
        DO I = 1,M
          DY(I) = DX(I)
        END DO
        IF (N.LT.7) RETURN
      END IF
      MP1 = M + 1
      DO I = MP1,N,7
        DY(I) = DX(I)
        DY(I+1) = DX(I+1)
        DY(I+2) = DX(I+2)
        DY(I+3) = DX(I+3)
        DY(I+4) = DX(I+4)
        DY(I+5) = DX(I+5)
        DY(I+6) = DX(I+6)
      END DO
    ELSE
!
!       code for unequal increments or equal increments
!       not equal to 1
!
      IX = 1
      IY = 1
      IF (INCX.LT.0) IX = (-N+1)*INCX + 1
      IF (INCY.LT.0) IY = (-N+1)*INCY + 1
      DO I = 1,N
        DY(IY) = DX(IX)
        IX = IX + INCX
        IY = IY + INCY
      END DO
    END IF
    RETURN
  END SUBROUTINE

  SUBROUTINE DSWAP(N,DX,INCX,DY,INCY)
!> \brief \b DSWAP
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!   SUBROUTINE DSWAP(N,DX,INCX,DY,INCY)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,INCY,N
!
!   .. Array Arguments ..
!   DOUBLE PRECISION DX(*),DY(*)
!   ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!>   interchanges two vectors.
!>   uses unrolled loops for increments equal one.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!>   jack dongarra, linpack, 3/11/78.
!>   modified 12/3/93, array(l) declarations changed to array(*)
!> \endverbatim
!>
!> =====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
!   .. Scalar Arguments ..
!   INTEGER INCX,INCY,N
!
!   .. Array Arguments ..
!   DOUBLE PRECISION DX(*),DY(*)
!   ..
!
! =====
!
!   .. Local Scalars ..
!   DOUBLE PRECISION DTEMP
!   INTEGER I,IX,IY,M,MP1
!
!   ..
!   .. Intrinsic Functions ..
!   INTRINSIC MOD
!
!   ..
!   IF (N.LE.0) RETURN
!   IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN

```

```

!      code for both increments equal to 1
!
!
!      clean-up loop
!
      M = MOD(N,3)
      IF (M.NE.0) THEN
        DO I = 1,M
          DTEMP = DX(I)
          DX(I) = DY(I)
          DY(I) = DTEMP
        END DO
      IF (N.LT.3) RETURN
      END IF
      MP1 = M + 1
      DO I = MP1,N,3
        DTEMP = DX(I)
        DX(I) = DY(I)
        DY(I) = DTEMP
        DTEMP = DX(I+1)
        DX(I+1) = DY(I+1)
        DY(I+1) = DTEMP
        DTEMP = DX(I+2)
        DX(I+2) = DY(I+2)
        DY(I+2) = DTEMP
      END DO
    ELSE
!
!      code for unequal increments or equal increments not equal
!      to 1
!
      IX = 1
      IY = 1
      IF (INCX.LT.0) IX = (-N+1)*INCX + 1
      IF (INCY.LT.0) IY = (-N+1)*INCY + 1
      DO I = 1,N
        DTEMP = DX(IX)
        DX(IX) = DY(IY)
        DY(IY) = DTEMP
        IX = IX + INCX
        IY = IY + INCY
      END DO
      END IF
      RETURN
    END SUBROUTINE
  DOUBLE PRECISION FUNCTION DNRM2(N,X,INCX)
!> \brief \b DNRM2
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      DOUBLE PRECISION FUNCTION DNRM2(N,X,INCX)
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!      ..
!      .. Array Arguments ..
!      DOUBLE PRECISION X(*)
!      ..
!
!
!> \par Purpose:
!      =====
!> \verbatim
!>
!> DNRM2 returns the euclidean norm of a vector via the function
!> name, so that
!>
!>    DNRM2 := sqrt( x'*x )
!> \endverbatim
!
!      Authors:
!      =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
!      =====
!> \verbatim
!>
!> -- This version written on 25-October-1982.
!>    Modified on 14-October-1993 to inline the call to DLASSQ.
!>    Sven Hammarling, Nag Ltd.
!> \endverbatim
!>
!      =====
!
!      -- Reference BLAS levell routine (version 3.4.0) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!      November 2011
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!      ..
!      .. Array Arguments ..
!      DOUBLE PRECISION X(*)
!      ..
!
!

```

```

! =====
!
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! .. Local Scalars ..
DOUBLE PRECISION ABSXI,NORM,SCALE,SSQ
INTEGER IX
!
! .. Intrinsic Functions ..
INTRINSIC ABS,SQRT
!
!
IF (N.LT.1 .OR. INCX.LT.1) THEN
    NORM = ZERO
ELSE IF (N.EQ.1) THEN
    NORM = ABS(X(1))
ELSE
    SCALE = ZERO
    SSQ = ONE
!
! The following loop is equivalent to this call to the LAPACK
! auxiliary routine:
! CALL DCLASSQ( N, X, INCX, SCALE, SSQ )
!
    DO 10 IX = 1,1 + (N-1)*INCX,INCX
        IF (X(IX).NE.ZERO) THEN
            ABSXI = ABS(X(IX))
            IF (SCALE.LT.ABSXI) THEN
                SSQ = ONE + SSQ* (SCALE/ABSXI)**2
                SCALE = ABSXI
            ELSE
                SSQ = SSQ + (ABSXI/SCALE)**2
            END IF
        END IF
    10 CONTINUE
    NORM = SCALE*SQRT(SSQ)
END IF

DNRM2 = NORM
RETURN

!
! End of DNRM2.
!
END FUNCTION
DOUBLE PRECISION FUNCTION DASUM(N,DX,INCX)
!> \brief \b DASUM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! DOUBLE PRECISION FUNCTION DASUM(N,DX,INCX)
!
! .. Scalar Arguments ..
INTEGER INCX,N
!
! ..
! .. Array Arguments ..
DOUBLE PRECISION DX(*)
!
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DASUM takes the sum of the absolute values.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> jack dongarra, linpack, 3/11/78.
!> modified 3/93 to return if incx .le. 0.
!> modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
!> =====
!
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
INTEGER INCX,N
!
! ..
! .. Array Arguments ..
DOUBLE PRECISION DX(*)
!
! ..
!
! =====
!
! .. Local Scalars ..
DOUBLE PRECISION DTEMP

```

```

        INTEGER I,M,MPI,NINCX
!
! ..
! .. Intrinsic Functions ..
        INTRINSIC DABS,MOD
!
! ..
        DASUM = 0.0d0
        DTEMP = 0.0d0
        IF (N.LE.0 .OR. INCX.LE.0) RETURN
        IF (INCX.EQ.1) THEN
!
!         code for increment equal to 1
!
!
!
!         clean-up loop
!
!         M = MOD(N,6)
!         IF (M.NE.0) THEN
!             DO I = 1,M
!                 DTEMP = DTEMP + DABS(DX(I))
!             END DO
!             IF (N.LT.6) THEN
!                 DASUM = DTEMP
!                 RETURN
!             END IF
!         END IF
!         MPI = M + 1
!         DO I = MPI,N,6
!             DTEMP = DTEMP + DABS(DX(I)) + DABS(DX(I+1)) + &
!                 DABS(DX(I+2)) + DABS(DX(I+3)) + &
!                 DABS(DX(I+4)) + DABS(DX(I+5))
!         END DO
!     ELSE
!
!         code for increment not equal to 1
!
!         NINCX = N*INCX
!         DO I = 1,NINCX,INCX
!             DTEMP = DTEMP + DABS(DX(I))
!         END DO
!     END IF
!     DASUM = DTEMP
!     RETURN
! END FUNCTION
SUBROUTINE DSCAL(N,DA,DX,INCX)
!> \brief \b DSCAL
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!     SUBROUTINE DSCAL(N,DA,DX,INCX)
!
!     .. Scalar Arguments ..
!     DOUBLE PRECISION DA
!     INTEGER INCX,N
!     ..
!     .. Array Arguments ..
!     DOUBLE PRECISION DX(*)
!     ..
!
!> \par Purpose:
! =====
!> \verbatim
!>
!>   DSCAL scales a vector by a constant.
!>   uses unrolled loops for increment equal to one.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!>
!>   jack dongarra, linpack, 3/11/78.
!>   modified 3/93 to return if incx .le. 0.
!>   modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! -- November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION DA
! INTEGER INCX,N
!
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DX(*)
! ..
!
! =====
!
! .. Local Scalars ..

```

```

        INTEGER I,M,MPI,NINCX
!
! ..
! .. Intrinsic Functions ..
INTRINSIC MOD
!
! ..
! IF (N.LE.0 .OR. INCX.LE.0) RETURN
! IF (INCX.EQ.1) THEN
!
!         code for increment equal to 1
!
!
!         clean-up loop
!
!         M = MOD(N,5)
!         IF (M.NE.0) THEN
!             DO I = 1,M
!                 DX(I) = DA*DX(I)
!             END DO
!             IF (N.LT.5) RETURN
!         END IF
!         MPI = M + 1
!         DO I = MPI,N,5
!             DX(I) = DA*DX(I)
!             DX(I+1) = DA*DX(I+1)
!             DX(I+2) = DA*DX(I+2)
!             DX(I+3) = DA*DX(I+3)
!             DX(I+4) = DA*DX(I+4)
!         END DO
!     ELSE
!
!         code for increment not equal to 1
!
!         NINCX = N*INCX
!         DO I = 1,NINCX,INCX
!             DX(I) = DA*DX(I)
!         END DO
!     END IF
!     RETURN
!     END SUBROUTINE
!     INTEGER FUNCTION IDAMAX(N,DX,INCX)
!> \brief \b IDAMAX
!
!     ===== DOCUMENTATION =====
!
!     Online html documentation available at
!     http://www.netlib.org/lapack/explore-html/
!
!     Definition:
!     =====
!
!     INTEGER FUNCTION IDAMAX(N,DX,INCX)
!
!     .. Scalar Arguments ..
!     INTEGER INCX,N
!     ..
!     .. Array Arguments ..
!     DOUBLE PRECISION DX(*)
!     ..
!
!> \par Purpose:
!     =====
!> \verbatim
!>
!>     IDAMAX finds the index of element having max. absolute value.
!> \endverbatim
!
!     Authors:
!     =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup aux_blas
!
!> \par Further Details:
!     =====
!> \verbatim
!>
!>     jack dongarra, linpack, 3/11/78.
!>     modified 3/93 to return if incx .le. 0.
!>     modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!
!     =====
!
!     -- Reference BLAS level1 routine (version 3.4.0) --
!     -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!     -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!     November 2011
!
!     .. Scalar Arguments ..
!     INTEGER INCX,N
!     ..
!     .. Array Arguments ..
!     DOUBLE PRECISION DX(*)
!     ..
!
!     =====
!
!     .. Local Scalars ..
!     DOUBLE PRECISION DMAX
!     INTEGER I,IX
!
!     ..
!     .. Intrinsic Functions ..
!     INTRINSIC DABS
!     ..

```

```

IDAMAX = 0
IF (N.LT.1 .OR. INCX.LE.0) RETURN
IDAMAX = 1
IF (N.EQ.1) RETURN
IF (INCX.EQ.1) THEN
!
!   code for increment equal to 1
!
    DMAX = DABS(DX(1))
    DO I = 2,N
        IF (DABS(DX(I)).GT.DMAX) THEN
            IDAMAX = I
            DMAX = DABS(DX(I))
        END IF
    END DO
ELSE
!
!   code for increment not equal to 1
!
    IX = 1
    DMAX = DABS(DX(1))
    IX = IX + INCX
    DO I = 2,N
        IF (DABS(DX(IX)).GT.DMAX) THEN
            IDAMAX = I
            DMAX = DABS(DX(IX))
        END IF
        IX = IX + INCX
    END DO
END IF
RETURN
END FUNCTION
SUBROUTINE DROTMG (DD1,DD2,DX1,DY1,DPARAM)
!> \brief \b DROTMG
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DROTMG (DD1,DD2,DX1,DY1,DPARAM)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION DD1,DD2,DX1,DY1
!
! .. Array Arguments ..
! DOUBLE PRECISION DPARAM(5)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!>
!>   CONSTRUCT THE MODIFIED GIVENS TRANSFORMATION MATRIX H WHICH ZEROS
!>   THE SECOND COMPONENT OF THE 2-VECTOR (DSQRT(DD1)*DX1,DSQRT(DD2))*T.
!>   WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..
!>
!>   DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0
!>
!>   (DH11 DH12)      (1.D0 DH12)      (DH11 1.D0)      (1.D0 0.D0)
!>   H=(      )      (      )      (      )      (      )
!>   (DH21 DH22),      (DH21 1.D0),      (-1.D0 DH22),      (0.D0 1.D0).
!>   LOCATIONS 2-4 OF DPARAM CONTAIN DH11, DH21, DH12, AND DH22
!>   RESPECTIVELY. (VALUES OF 1.D0, -1.D0, OR 0.D0 IMPLIED BY THE
!>   VALUE OF DPARAM(1) ARE NOT STORED IN DPARAM.)
!>
!>   THE VALUES OF GAMSQ AND RGAMSQ SET IN THE DATA STATEMENT MAY BE
!>   INEXACT. THIS IS OK AS THEY ARE ONLY USED FOR TESTING THE SIZE
!>   OF DD1 AND DD2. ALL ACTUAL SCALING OF DATA IS DONE USING GAM.
!>
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in,out] DD1
!> \verbatim
!>   DD1 is DOUBLE PRECISION
!> \endverbatim
!
!> \param[in,out] DD2
!> \verbatim
!>   DD2 is DOUBLE PRECISION
!> \endverbatim
!
!> \param[in,out] DX1
!> \verbatim
!>   DX1 is DOUBLE PRECISION
!> \endverbatim
!
!> \param[in] DY1
!> \verbatim
!>   DY1 is DOUBLE PRECISION
!> \endverbatim
!
!> \param[in,out] DPARAM
!> \verbatim
!>   DPARAM is DOUBLE PRECISION array, dimension 5
!>   DPARAM(1)=DFLAG
!>   DPARAM(2)=DH11
!>   DPARAM(3)=DH21
!>   DPARAM(4)=DH12
!>   DPARAM(5)=DH22
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee

```

```

!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
! =====
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
DOUBLE PRECISION DD1,DD2,DX1,DY1
!
! ..
! .. Array Arguments ..
DOUBLE PRECISION DPARAM(5)
!
! ..
! =====
!
! .. Local Scalars ..
DOUBLE PRECISION DFLAG,DH11,DH12,DH21,DH22,DP1,DP2,DQ1,DQ2,DTEMP, &
DU,GAM,GAMSQ,ONE,RGAMSQ,TWO,ZERO
!
! ..
! .. Intrinsic Functions ..
INTRINSIC DABS
!
! ..
! .. Data statements ..
!
DATA ZERO,ONE,TWO/0.D0,1.D0,2.D0/
DATA GAM,GAMSQ,RGAMSQ/4096.D0,16777216.D0,5.9604645D-8/
!
! ..
!
IF (DD1.LT.ZERO) THEN
GO ZERO-H-D-AND-DX1..
DFLAG = -ONE
DH11 = ZERO
DH12 = ZERO
DH21 = ZERO
DH22 = ZERO
!
DD1 = ZERO
DD2 = ZERO
DX1 = ZERO
ELSE
CASE-DD1-NONNEGATIVE
DP2 = DD2*DY1
IF (DP2.EQ.ZERO) THEN
DFLAG = -TWO
DPARAM(1) = DFLAG
RETURN
END IF
REGULAR-CASE..
DP1 = DD1*DX1
DQ2 = DP2*DY1
DQ1 = DP1*DX1
!
IF (DABS(DQ1).GT.DABS(DQ2)) THEN
DH21 = -DY1/DX1
DH12 = DP2/DP1
!
DU = ONE - DH12*DH21
!
IF (DU.GT.ZERO) THEN
DFLAG = ZERO
DD1 = DD1/DU
DD2 = DD2/DU
DX1 = DX1*DU
END IF
ELSE
IF (DQ2.LT.ZERO) THEN
GO ZERO-H-D-AND-DX1..
DFLAG = -ONE
DH11 = ZERO
DH12 = ZERO
DH21 = ZERO
DH22 = ZERO
!
DD1 = ZERO
DD2 = ZERO
DX1 = ZERO
ELSE
DFLAG = ONE
DH11 = DP1/DP2
DH22 = DX1/DY1
DU = ONE + DH11*DH22
DTEMP = DD2/DU
DD2 = DD1/DU
DD1 = DTEMP
DX1 = DY1*DU
END IF
END IF
!
PROCEDURE..SCALE-CHECK
IF (DD1.NE.ZERO) THEN
DO WHILE ((DD1.LE.RGAMSQ) .OR. (DD1.GE.GAMSQ))
IF (DFLAG.EQ.ZERO) THEN
DH11 = ONE
DH22 = ONE
DFLAG = -ONE
ELSE
DH21 = -ONE
DH12 = ONE
DFLAG = -ONE
END IF
IF (DD1.LE.RGAMSQ) THEN
DD1 = DD1*GAM**2
DX1 = DX1/GAM

```

```

        DH11 = DH11/GAM
        DH12 = DH12/GAM
    ELSE
        DD1 = DD1/GAM**2
        DX1 = DX1*GAM
        DH11 = DH11*GAM
        DH12 = DH12*GAM
    END IF
ENDDO
END IF

IF (DD2.NE.ZERO) THEN
    DO WHILE ( (DABS(DD2).LE.RGAMSQ) .OR. (DABS(DD2).GE.GAMSQ) )
        IF (DFLAG.EQ.ZERO) THEN
            DH11 = ONE
            DH22 = ONE
            DFLAG = -ONE
        ELSE
            DH21 = -ONE
            DH12 = ONE
            DFLAG = -ONE
        END IF
        IF (DABS(DD2).LE.RGAMSQ) THEN
            DD2 = DD2*GAM**2
            DH21 = DH21/GAM
            DH22 = DH22/GAM
        ELSE
            DD2 = DD2/GAM**2
            DH21 = DH21*GAM
            DH22 = DH22*GAM
        END IF
    END DO
END IF

END IF

IF (DFLAG.LT.ZERO) THEN
    DPARAM(2) = DH11
    DPARAM(3) = DH21
    DPARAM(4) = DH12
    DPARAM(5) = DH22
ELSE IF (DFLAG.EQ.ZERO) THEN
    DPARAM(3) = DH21
    DPARAM(4) = DH12
ELSE
    DPARAM(2) = DH11
    DPARAM(5) = DH22
END IF

DPARAM(1) = DFLAG
RETURN
END SUBROUTINE
SUBROUTINE DROTM(N,DX,INCX,DY,INCY,DPARAM)
!> \brief \b DROTM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DROTM(N,DX,INCX,DY,INCY,DPARAM)
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION DPARAM(5),DX(*),DY(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> APPLY THE MODIFIED GIVENS TRANSFORMATION, H, TO THE 2 BY N MATRIX
!>
!> (DX**T) , WHERE **T INDICATES TRANSPOSE. THE ELEMENTS OF DX ARE IN
!> (DY**T)
!>
!> DX(LX+I*INCX), I = 0 TO N-1, WHERE LX = 1 IF INCX .GE. 0, ELSE
!> LX = (-INCX)*N, AND SIMILARLY FOR SY USING LY AND INCY.
!> WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..
!>
!> DFLAG=-1.D0      DFLAG=0.D0      DFLAG=1.D0      DFLAG=-2.D0
!>
!> (DH11 DH12)      (1.D0 DH12)      (DH11 1.D0)      (1.D0 0.D0)
!> H=(      )      (      )      (      )      (      )
!> (DH21 DH22), (DH21 1.D0), (-1.D0 DH22), (0.D0 1.D0).
!> SEE DROTMG FOR A DESCRIPTION OF DATA STORAGE IN DPARAM.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> number of elements in input vector(s)
!> \endverbatim
!>
!> \param[in,out] DX
!> \verbatim
!> DX is DOUBLE PRECISION array, dimension N
!> double precision vector with N elements
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> storage spacing between elements of DX
!>

```



```

        END DO
        ELSE IF (DFLAG.EQ.ZERO) THEN
            DH12 = DPARAM(4)
            DH21 = DPARAM(3)
            DO I = 1,N
                W = DX(KX)
                Z = DY(KY)
                DX(KX) = W + Z*DH12
                DY(KY) = W*DH21 + Z
                KX = KX + INCX
                KY = KY + INCY
            END DO
        ELSE
            DH11 = DPARAM(2)
            DH22 = DPARAM(5)
            DO I = 1,N
                W = DX(KX)
                Z = DY(KY)
                DX(KX) = W*DH11 + Z
                DY(KY) = -W + DH22*Z
                KX = KX + INCX
                KY = KY + INCY
            END DO
        END IF
    END IF
    RETURN
END SUBROUTINE
DOUBLE PRECISION FUNCTION DSDOT(N,SX,INCX,SY,INCY)
!> \brief \b DSDOT
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!     DOUBLE PRECISION FUNCTION DSDOT(N,SX,INCX,SY,INCY)
!
!     .. Scalar Arguments ..
!     INTEGER INCX,INCY,N
!     ..
!     .. Array Arguments ..
!     REAL SX(*),SY(*)
!     ..
!
!     AUTHORS
!     =====
!     Lawson, C. L., (JPL), Hanson, R. J., (SNLA),
!     Kincaid, D. R., (U. of Texas), Krogh, F. T., (JPL)
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> Compute the inner product of two vectors with extended
!> precision accumulation and result.
!>
!> Returns D.P. dot product accumulated in D.P., for S.P. SX and SY
!> DSDOT = sum for I = 0 to N-1 of SX(LX+I*INCX) * SY(LY+I*INCY),
!> where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is
!> defined in a similar way using INCY.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     number of elements in input vector(s)
!> \endverbatim
!>
!> \param[in] SX
!> \verbatim
!>     SX is REAL array, dimension(N)
!>     single precision vector with N elements
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     storage spacing between elements of SX
!> \endverbatim
!>
!> \param[in] SY
!> \verbatim
!>     SY is REAL array, dimension(N)
!>     single precision vector with N elements
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     storage spacing between elements of SY
!> \endverbatim
!>
!> \result DSDOT
!> \verbatim
!>     DSDOT is DOUBLE PRECISION
!>     DSDOT double precision dot product (zero if N.LE.0)
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!

```

```

!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!
!> \verbatim
!> \endverbatim
!
!> \par References:
! =====
!
!> \verbatim
!>
!> C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T.
!> Krogh, Basic linear algebra subprograms for Fortran
!> usage, Algorithm No. 539, Transactions on Mathematical
!> Software 5, 3 (September 1979), pp. 308-323.
!>
!> REVISION HISTORY (YYMMDD)
!>
!> 791001 DATE WRITTEN
!> 890831 Modified array declarations. (WRB)
!> 890831 REVISION DATE from Version 3.2
!> 891214 Prologue converted to Version 4.0 format. (BAB)
!> 920310 Corrected definition of LX in DESCRIPTION. (WRB)
!> 920501 Reformatted the REFERENCES section. (WRB)
!> 070118 Reformat to LAPACK style (JL)
!> \endverbatim
!>
! =====
!
! -- Reference BLAS levell routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX, INCY, N
! ..
! .. Array Arguments ..
! REAL SX(*), SY(*)
! ..
!
! Authors:
! =====
! Lawson, C. L., (JPL), Hanson, R. J., (SNLA),
! Kincaid, D. R., (U. of Texas), Krogh, F. T., (JPL)
!
! =====
!
! .. Local Scalars ..
! INTEGER I, KX, KY, NS
! ..
! .. Intrinsic Functions ..
! INTRINSIC DBLE
! ..
! DSDOT = 0.0D0
! IF (N.LE.0) RETURN
! IF (INCX.EQ.INCY .AND. INCX.GT.0) THEN
!
! Code for equal, positive, non-unit increments.
!
!     NS = N*INCX
!     DO I = 1, NS, INCX
!         DSDOT = DSDOT + DBLE(SX(I))*DBLE(SY(I))
!     END DO
! ELSE
!
! Code for unequal or nonpositive increments.
!
!     KX = 1
!     KY = 1
!     IF (INCX.LT.0) KX = 1 + (1-N)*INCX
!     IF (INCY.LT.0) KY = 1 + (1-N)*INCY
!     DO I = 1, N
!         DSDOT = DSDOT + DBLE(SX(KX))*DBLE(SY(KY))
!         KX = KX + INCX
!         KY = KY + INCY
!     END DO
! END IF
! RETURN
! END FUNCTION
!
!
! end program

```

APPENDIX A.3 – DBLAT2.f90

```

program testdblat2
implicit none

! Variables
CHARACTER :: kin
DOUBLE PRECISION AA(9), X(10), Y(10)
INTEGER LDA

CALL DBLAT2
print *, 'DBLAT2 Done.'

!https://software.intel.com/en-us/node/522288
DO 10 I=1,10
    X(I) = 1.0D0
    Y(I) = 1.0D0
10 CONTINUE

LDA = 3 ! Row major
DO 11 I= 1, 2
    DO 12 J=1,3
        AA((I-1)+(J-1)*LDA)+1 = (J-1)+1
12 CONTINUE
11 CONTINUE

CALL DGER(2,3,0.5D0,X,2,Y,1,AA,LDA)

DO 13 I= 1, 2
    DO 14 J=1,3
        print *, ( (I-1)+(J-1)*LDA)+1,AA((I-1)+(J-1)*LDA)+1)
14 CONTINUE
13 CONTINUE
read(*,'(A1)'),kin
STOP

CONTAINS
!> \brief \b LSAME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! LOGICAL FUNCTION LSAME(CA,CB)
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> LSAME returns .TRUE. if CA is the same letter as CB regardless of
!> case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] CA
!> \verbatim
!> CA is CHARACTER*1
!> \endverbatim
!>
!> \param[in] CB
!> \verbatim
!> CB is CHARACTER*1
!> CA and CB specify the single characters to be compared.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup aux_blas
!
! =====
! LOGICAL FUNCTION LSAME(CA,CB)
!
! -- Reference BLAS level1 routine (version 3.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
! =====
!
! .. Intrinsic Functions ..
! INTRINSIC ICHAR
!
! .. Local Scalars ..
! INTEGER INTA,INTB,ZCODE
! ..
!
! Test if the characters are equal
!
! LSAME = CA .EQ. CB

```

```

      IF (LSAME) RETURN
!
!   Now test for equivalence if both characters are alphabetic.
!
      ZCODE = ICHAR('Z')
!
!   Use 'Z' rather than 'A' so that ASCII can be detected on Prime
!   machines, on which ICHAR returns a value with bit 8 set.
!   ICHAR('A') on Prime machines returns 193 which is the same as
!   ICHAR('A') on an EBCDIC machine.
!
      INTA = ICHAR(CA)
      INTB = ICHAR(CB)
!
      IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
!
!         ASCII is assumed - ZCODE is the ASCII code of either lower or
!         upper case 'Z'.
!
!         IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
!         IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
!
      ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN
!
!         EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or
!         upper case 'Z'.
!
!         IF (INTA.GE.129 .AND. INTA.LE.137 .OR. &
!             INTA.GE.145 .AND. INTA.LE.153 .OR. &
!             INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
!         IF (INTB.GE.129 .AND. INTB.LE.137 .OR. &
!             INTB.GE.145 .AND. INTB.LE.153 .OR. &
!             INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
!
      ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
!
!         ASCII is assumed, on Prime machines - ZCODE is the ASCII code
!         plus 128 of either lower or upper case 'Z'.
!
!         IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
!         IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
      END IF
      LSAME = INTA .EQ. INTB
!
      RETURN
!
!   End of LSAME
!
END FUNCTION

      PROGRAM DBLAT2
!
!
!   \par Purpose:
!   =====
!
!   \verbatim
!
!   Test program for the DOUBLE PRECISION Level 2 Blas.
!
!   The program must be driven by a short data file. The first 18 records
!   of the file are read using list-directed input, the last 16 records
!   are read using the format ( A6, L2 ). An annotated example of a data
!   file can be obtained by deleting the first 3 characters from the
!   following 34 lines:
!   'dblat2.out'      NAME OF SUMMARY OUTPUT FILE
!   6                UNIT NUMBER OF SUMMARY FILE
!   'DBLAT2.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
!   -1              UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
!   F              LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
!   F              LOGICAL FLAG, T TO STOP ON FAILURES.
!   T              LOGICAL FLAG, T TO TEST ERROR EXITS.
!   16.0           THRESHOLD VALUE OF TEST RATIO
!   6              NUMBER OF VALUES OF N
!   0 1 2 3 5 9    VALUES OF N
!   4              NUMBER OF VALUES OF K
!   0 1 2 4        VALUES OF K
!   4              NUMBER OF VALUES OF INCX AND INCY
!   1 2 -1 -2      VALUES OF INCX AND INCY
!   3              NUMBER OF VALUES OF ALPHA
!   0.0 1.0 0.7    VALUES OF ALPHA
!   3              NUMBER OF VALUES OF BETA
!   0.0 1.0 0.9    VALUES OF BETA
!   DGEMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DGBMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DSYMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DSBMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DSPMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTRMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTBMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTPMV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTRSV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTBSV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DTPSV  T PUT F FOR NO TEST. SAME COLUMNS.
!   DGER   T PUT F FOR NO TEST. SAME COLUMNS.
!   DSYR   T PUT F FOR NO TEST. SAME COLUMNS.
!   DSPR   T PUT F FOR NO TEST. SAME COLUMNS.
!   DSYR2  T PUT F FOR NO TEST. SAME COLUMNS.
!   DSPR2  T PUT F FOR NO TEST. SAME COLUMNS.
!
!   Further Details
!   =====
!
!   See:
!
!   Dongarra J. J., Du Croz J. J., Hammarling S. and Hanson R. J..
!   An extended set of Fortran Basic Linear Algebra Subprograms.
!
!   Technical Memoranda Nos. 41 (revision 3) and 81, Mathematics
!   and Computer Science Division, Argonne National Laboratory,
!   9700 South Cass Avenue, Argonne, Illinois 60439, US.
!
!   Or
!

```

```

!      NAG Technical Reports TR3/87 and TR4/87, Numerical Algorithms
!      Group Ltd., NAG Central Office, 256 Banbury Road, Oxford
!      OX2 7DE, UK, and Numerical Algorithms Group Inc., 1101 31st
!      Street, Suite 100, Downers Grove, Illinois 60515-1263, USA.
!
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! 10-9-00: Change STATUS='NEW' to 'UNKNOWN' so that the testers
!          can be run multiple times without deleting generated
!          output files (susan)
! \endverbatim
!
! Authors:
! =====
!
! \author Univ. of Tennessee
! \author Univ. of California Berkeley
! \author Univ. of Colorado Denver
! \author NAG Ltd.
!
! \date April 2012
!
! \ingroup double_blas_testing
!
! =====
!
! SUBROUTINE DBLAT2
!
! -- Reference BLAS test routine (version 3.4.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! April 2012
!
! =====
!
! .. Parameters ..
INTEGER          NIN
PARAMETER        ( NIN = 5 )
INTEGER          NSUBS
PARAMETER        ( NSUBS = 16 )
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D0, ONE = 1.0D0 )
INTEGER          NMAX, INCMAX
PARAMETER        ( NMAX = 65, INCMAX = 2 )
INTEGER          NINMAX, NIDMAX, NKBMAX, NALMAX, NBEMAX
PARAMETER        ( NINMAX = 7, NIDMAX = 9, NKBMAX = 7, &
                  NALMAX = 7, NBEMAX = 7 )
!
! .. Local Scalars ..
DOUBLE PRECISION EPS, ERR, THRESH
INTEGER          I, ISNUM, J, N, NALF, NBET, NIDIM, NINC, NKB, &
                  NOUT, NTRA
LOGICAL          FATAL, LTESTT, REWI, SAME, SFATAL, TRACE, &
                  TSERR
CHARACTER*1      TRANS
CHARACTER*6      SNAMET
CHARACTER*32     SNAPS, SUMMARY
!
! .. Local Arrays ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), &
                  ALF( NALMAX ), AS( NMAX*NMAX ), BET( NBEMAX ), &
                  G( NMAX ), X( NMAX ), XS( NMAX*INCMAX ), &
                  XX( NMAX*INCMAX ), Y( NMAX ), &
                  YS( NMAX*INCMAX ), YT( NMAX ), &
                  YY( NMAX*INCMAX ), Z( 2*NMAX )
INTEGER          IDIM( NIDMAX ), INC( NINMAX ), KB( NKBMAX )
LOGICAL          LTEST( NSUBS )
CHARACTER*6      SNAME( NSUBS )
!
! .. External Functions ..
DOUBLE PRECISION DDIFF
LOGICAL          LDE
EXTERNAL          DDIFF, LDE
!
! .. External Subroutines ..
EXTERNAL          DCHK1, DCHK2, DCHK3, DCHK4, DCHK5, DCHK6,
                  $      DCHKE, DMVCH
!
! .. Intrinsic Functions ..
INTRINSIC        ABS, MAX, MIN
!
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
CHARACTER*6      SRNAMT
!
LOGICAL          MYFLAG
!
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUTC, OK, LERR
COMMON           /SRNAMC/SRNAMT
COMMON           /MYSTUFF/MYFLAG
!
! .. Data statements ..
DATA             SNAME/'DGEMV ', 'DGBMV ', 'DSYMV ', 'DSBMV ', &
                  'DSPMV ', 'DTRMV ', 'DTBMV ', 'DTPMV ', &
                  'DTRSV ', 'DTBSV ', 'DTPSV ', 'DGER ', &
                  'DSYR ', 'DSPR ', 'DSYR2 ', 'DSPR2 '/
!
! .. Executable Statements ..
!
! Read name and unit number for summary output file and open file.
!
MYFLAG = .FALSE.
READ( NIN, FMT = * ) SUMMARY
READ( NIN, FMT = * ) NOUT
! print *, SUMMARY
! print *, NOUT
!
OPEN( NOUT, FILE = SUMMARY, STATUS = 'UNKNOWN' )
NOUTC = NOUT
!
! Read name and unit number for snapshot output file and open file.
!
READ( NIN, FMT = * ) SNAPS
READ( NIN, FMT = * ) NTRA
! print *, snaps
! print *, ntra
!
TRACE = NTRA.GE.0
IF( TRACE ) THEN

```

```

        OPEN( NTRA, FILE = SNAPS, STATUS = 'UNKNOWN' )
    END IF
!   Read the flag that directs rewinding of the snapshot file.
    READ( NIN, FMT = * )REWI
    REWI = REWI.AND.TRACE
!   Read the flag that directs stopping on any failure.
    READ( NIN, FMT = * )SFATAL
!   Read the flag that indicates whether error exits are to be tested.
    READ( NIN, FMT = * )TSTERR
!   Read the threshold value of the test ratio
    READ( NIN, FMT = * )THRESH
!
!   Read and check the parameter values for the tests.
!
!   Values of N
    READ( NIN, FMT = * )NIDIM
    IF( NIDIM.LT.1.OR.NIDIM.GT.NIDMAX )THEN
        WRITE( NOUT, FMT = 9997 )'N', NIDMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( IDIM( I ), I = 1, NIDIM )
    DO 10 I = 1, NIDIM
        IF( IDIM( I ).LT.0.OR.IDIM( I ).GT.NMAX )THEN
            WRITE( NOUT, FMT = 9996 )NMAX
            GO TO 230
        END IF
    END IF
10 CONTINUE
!   print *, 'got to val k'
!   Values of K
    READ( NIN, FMT = * )NKB
    IF( NKB.LT.1.OR.NKB.GT.NKBMAX )THEN
        WRITE( NOUT, FMT = 9997 )'K', NKBMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( KB( I ), I = 1, NKB )
    DO 20 I = 1, NKB
        IF( KB( I ).LT.0 )THEN
            WRITE( NOUT, FMT = 9995 )
            GO TO 230
        END IF
    END IF
20 CONTINUE
!   Values of INCX and INCY
    READ( NIN, FMT = * )NINC
    IF( NINC.LT.1.OR.NINC.GT.NINMAX )THEN
        WRITE( NOUT, FMT = 9997 )'INCX AND INCY', NINMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( INC( I ), I = 1, NINC )
    DO 30 I = 1, NINC
        IF( INC( I ).EQ.0.OR.ABS( INC( I ) ).GT.INCMAX )THEN
            WRITE( NOUT, FMT = 9994 )INCMAX
            GO TO 230
        END IF
    END IF
30 CONTINUE
!   Values of ALPHA
    READ( NIN, FMT = * )NALF
    IF( NALF.LT.1.OR.NALF.GT.NALMAX )THEN
        WRITE( NOUT, FMT = 9997 )'ALPHA', NALMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( ALF( I ), I = 1, NALF )
!   Values of BETA
    READ( NIN, FMT = * )NBET
    IF( NBET.LT.1.OR.NBET.GT.NBEMAX )THEN
        WRITE( NOUT, FMT = 9997 )'BETA', NBEMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( BET( I ), I = 1, NBET )
!
!   Report values of parameters.
!
    WRITE( NOUT, FMT = 9993 )
    WRITE( NOUT, FMT = 9992 )( IDIM( I ), I = 1, NIDIM )
    WRITE( NOUT, FMT = 9991 )( KB( I ), I = 1, NKB )
    WRITE( NOUT, FMT = 9990 )( INC( I ), I = 1, NINC )
    WRITE( NOUT, FMT = 9989 )( ALF( I ), I = 1, NALF )
    WRITE( NOUT, FMT = 9988 )( BET( I ), I = 1, NBET )
    IF( .NOT.TSTERR )THEN
        WRITE( NOUT, FMT = * )
        WRITE( NOUT, FMT = 9980 )
    END IF
    WRITE( NOUT, FMT = * )
    WRITE( NOUT, FMT = 9999 )THRESH
    WRITE( NOUT, FMT = * )
!
!   Read names of subroutines and flags which indicate
!   whether they are to be tested.
!
    DO 40 I = 1, NSUBS
        LTEST( I ) = .FALSE.
    40 CONTINUE
50 READ( NIN, FMT = 9984, END = 80 )SNAMET, LTESTT
    DO 60 I = 1, NSUBS
        IF( SNAMET.EQ.SNAMES( I ) )%
            GO TO 70
    60 CONTINUE
    WRITE( NOUT, FMT = 9986 )SNAMET
    STOP
70 LTEST( I ) = LTESTT
    GO TO 50
!
!   80 CONTINUE
    CLOSE ( NIN )
!
!   Compute EPS (the machine precision).
!
    EPS = EPSILON(ZERO)
    WRITE( NOUT, FMT = 9998 )EPS
!
!   Check the reliability of DMVCH using exact data.
!
    N = MIN( 32, NMAX )
    DO 120 J = 1, N
        DO 110 I = 1, N

```

```

      A( I, J ) = MAX( I - J + 1, 0 )
110  CONTINUE
      X( J ) = J
      Y( J ) = ZERO
120  CONTINUE
      DO 130 J = 1, N
        YY( J ) = J*( ( J + 1)*J )/2 - ( ( J + 1)*J*( J - 1 ) )/3
130  CONTINUE
! YY holds the exact result. On exit from DMVCH YT holds
! the result computed by DMVCH.
      TRANS = 'N'
      CALL DMVCH( TRANS, N, N, ONE, A, NMAX, X, 1, ZERO, Y, 1, YT, G,&
        YY, EPS, ERR, FATAL, NOUT, .TRUE. )
      SAME = LDE( YY, YT, N )
      IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
        WRITE( NOUT, FMT = 9985 )TRANS, SAME, ERR
        STOP
      END IF
      TRANS = 'T'
! WRITE( NOUT,*) '2nd DMVCH'
      CALL DMVCH( TRANS, N, N, ONE, A, NMAX, X, -1, ZERO, Y, -1, YT, G,&
        YY, EPS, ERR, FATAL, NOUT, .TRUE. )
      SAME = LDE( YY, YT, N )
      IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
        WRITE( NOUT, FMT = 9985 )TRANS, SAME, ERR
        STOP
      END IF
!
! Test each subroutine in turn.
!
      DO 210 ISNUM = 1, NSUBS
        WRITE( NOUT, FMT = * )
        IF( .NOT.LTEST( ISNUM ) )THEN
! Subprogram is not to be tested.
          WRITE( NOUT, FMT = 9983 )SNAMES( ISNUM )
        ELSE
          SRNAMT = SNAMES( ISNUM )
! Test error exits.
          IF( TSERR )THEN
            CALL DCHKE( ISNUM, SNAMES( ISNUM ), NOUT )
            WRITE( NOUT, FMT = * )
          END IF
! Test computations.
          INFOT = 0
          OK = .TRUE.
          FATAL = .FALSE.
          GO TO ( 140, 140, 150, 150, 150, 160, 160,&
            160, 160, 160, 160, 170, 180, 180,&
            190, 190 )ISNUM
! Test DGEMV, 01, and DGBMV, 02.
! 140 FATAL = .FALSE.
! 140 CALL DCHK1( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF,&
          NBET, BET, NINC, INC, NMAX, INCMAX, A, AA, AS,&
          X, XX, XS, Y, YY, YS, YT, G )
! STOP
          GO TO 200
! Test DSYMV, 03, DSBMV, 04, and DSPMV, 05.
! 150 FATAL = .FALSE.
! 150 CALL DCHK2( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF,&
          NBET, BET, NINC, INC, NMAX, INCMAX, A, AA, AS,&
          X, XX, XS, Y, YY, YS, YT, G )
! GO TO 200
! Test DTRMV, 06, DTBMV, 07, DTPMV, 08,
! DTRSV, 09, DTBSV, 10, and DTSPV, 11.
! 160 FATAL = .FALSE.
! 160 CALL DCHK3( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NKB, KB, NINC, INC,&
          NMAX, INCMAX, A, AA, AS, Y, YY, YS, YT, G, Z )
! GO TO 200
! Test DGER, 12.
! 170 FATAL = .FALSE.
! 170 CALL DCHK4( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC,&
          NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS,&
          YT, G, Z )
! GO TO 200
! Test DSYR, 13, and DSPR, 14.
! 180 CALL DCHK5( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC,&
          NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS,&
          YT, G, Z )
! GO TO 200
! Test DSYR2, 15, and DSPR2, 16.
! 190 CALL DCHK6( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
          REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC,&
          NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS,&
          YT, G, Z )
!
! 200 IF( FATAL.AND.SFATAL )&
      GO TO 220
      END IF
210 CONTINUE
      WRITE( NOUT, FMT = 9982 )
      GO TO 240
!
220 CONTINUE
      WRITE( NOUT, FMT = 9981 )
      GO TO 240
!
230 CONTINUE
      WRITE( NOUT, FMT = 9987 )
!
240 CONTINUE
      IF( TRACE ) &
        CLOSE ( NTRA )
      CLOSE ( NOUT )
      RETURN ! replace STOP
!
9999 FORMAT( ' ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LES',&
  'S THAN', F8.2 )
9998 FORMAT( ' RELATIVE MACHINE PRECISION IS TAKEN TO BE', 1P, D9.1 )

```

```

9997 FORMAT( ' NUMBER OF VALUES OF ', A, ' IS LESS THAN 1 OR GREATER ',&
' THAN ', I2 )
9996 FORMAT( ' VALUE OF N IS LESS THAN 0 OR GREATER THAN ', I2 )
9995 FORMAT( ' VALUE OF K IS LESS THAN 0' )
9994 FORMAT( ' ABSOLUTE VALUE OF INCX OR INCY IS 0 OR GREATER THAN ',&
I2 )
9993 FORMAT( ' TESTS OF THE DOUBLE PRECISION LEVEL 2 BLAS', //' THE F',&
' FOLLOWING PARAMETER VALUES WILL BE USED:' )
9992 FORMAT( ' FOR N ', 916 )
9991 FORMAT( ' FOR K ', 716 )
9990 FORMAT( ' FOR INCX AND INCY ', 716 )
9989 FORMAT( ' FOR ALPHA ', 7F6.1 )
9988 FORMAT( ' FOR BETA ', 7F6.1 )
9987 FORMAT( ' AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM',&
/' ***** TESTS ABANDONED *****' )
9986 FORMAT( ' SUBPROGRAM NAME ', A6, ' NOT RECOGNIZED', /' ***** T',&
' ESTS ABANDONED *****' )
9985 FORMAT( ' ERROR IN DMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALU',&
' ATED WRONGLY.', /' DMVCH WAS CALLED WITH TRANS = ', A1,&
' AND RETURNED SAME = ', L1, ' AND ERR = ', F12.3, '.', /&
' THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.'&
/' ***** TESTS ABANDONED *****' )
9984 FORMAT( A6, L2 )
9983 FORMAT( IX, A6, ' WAS NOT TESTED' )
9982 FORMAT( /' END OF TESTS' )
9981 FORMAT( /' ***** FATAL ERROR - TESTS ABANDONED *****' )
9980 FORMAT( ' ERROR-EXITS WILL NOT BE TESTED' )
!
! End of DBLAT2.
!
END SUBROUTINE
SUBROUTINE DCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF, NBET,&
BET, NINC, INC, NMAX, INCMAX, A, AA, AS, X, XX,&
XS, Y, YY, YS, YT, G )
!
! Tests DGEMV and DGBMV.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
DOUBLE PRECISION ZERO, HALF
PARAMETER ( ZERO = 0.0D0, HALF = 0.5D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER INCMAX, NALF, NBET, NIDIM, NINC, NKB, NMAX,&
NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
! .. Array Arguments ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
AS( NMAX*NMAX ), BET( NBET ), G( NMAX ),&
X( NMAX ), XS( NMAX*INCMAX ),&
XX( NMAX*INCMAX ), Y( NMAX ),&
YS( NMAX*INCMAX ), YT( NMAX ),&
YY( NMAX*INCMAX )
INTEGER IDIM( NIDIM ), INC( NINC ), KB( NKB )
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, ALS, BETA, BLS, ERR, ERRMAX, TRANSL
INTEGER I, IA, IB, IC, IKU, IM, IN, INCX, INCXS, INCY,&
INCYS, IX, IY, KL, KLS, KU, KUS, LAA, LDA,&
LDAS, LX, LY, M, ML, MS, N, NARGS, NC, ND, NK,&
NL, NS
LOGICAL BANDED, FULL, NULL, RESET, SAME, TRAN
CHARACTER*1 TRANS, TRANSS
CHARACTER*3 ICH
! .. Local Arrays ..
LOGICAL ISAME( 13 )
! .. External Functions ..
LOGICAL LDE, LDERES
EXTERNAL LDE, LDERES
! .. External Subroutines ..
EXTERNAL DGBMV, DGEMV, DMAKE, DMVCH
! .. Intrinsic Functions ..
INTRINSIC ABS, MAX, MIN
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
DATA ICH/'NTC'/
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'E'
BANDED = SNAME( 3: 3 ).EQ.'B'
! Define the number of arguments.
IF( FULL )THEN
NARGS = 11
ELSE IF( BANDED )THEN
NARGS = 13
END IF
!
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
DO 120 IN = 1, NIDIM
N = IDIM( IN )
ND = N/2 + 1
!
DO 110 IM = 1, 2
IF( IM.EQ.1 )&
M = MAX( N - ND, 0 )
IF( IM.EQ.2 )&
M = MIN( N + ND, NMAX )
!
IF( BANDED )THEN
NK = NKB
ELSE
NK = 1

```

```

END IF
DO 100 IKU = 1, NK
  IF( BANDED )THEN
    KU = KB( IKU )
    KL = MAX( KU - 1, 0 )
  ELSE
    KU = N - 1
    KL = M - 1
  END IF
! Set LDA to 1 more than minimum value if room.
IF( BANDED )THEN
  LDA = KL + KU + 1
ELSE
  LDA = M
END IF
IF( LDA.LT.NMAX )&
  LDA = LDA + 1
! Skip tests if not enough room.
IF( LDA.GT.NMAX )&
  GO TO 100
LAA = LDA*N
NULL = N.LE.0.OR.M.LE.0
!
! Generate the matrix A.
!
TRANSL = ZERO
CALL DMAKE( SNAME( 2: 3 ), ' ', ' ', M, N, A, NMAX, AA,&
  LDA, KL, KU, RESET, TRANSL )
!
DO 90 IC = 1, 3
  TRANS = ICH( IC: IC )
  TRAN = TRANS.EQ.'T'.OR.TRANS.EQ.'C'
!
  IF( TRAN )THEN
    ML = N
    NL = M
  ELSE
    ML = M
    NL = N
  END IF
!
DO 80 IX = 1, NINC
  INCX = INC( IX )
  LX = ABS( INCX )*NL
!
! Generate the vector X.
!
TRANSL = HALF
CALL DMAKE( 'GE', ' ', ' ', 1, NL, X, 1, XX,&
  ABS( INCX ), 0, NL - 1, RESET, TRANSL )
IF( NL.GT.1 )THEN
  X( NL/2 ) = ZERO
  XX( 1 + ABS( INCX )*( NL/2 - 1 ) ) = ZERO
END IF
!
DO 70 IY = 1, NINC
  INCY = INC( IY )
  LY = ABS( INCY )*ML
!
DO 60 IA = 1, NALF
  ALPHA = ALF( IA )
!
DO 50 IB = 1, NBET
  BETA = BET( IB )
!
! Generate the vector Y.
!
TRANSL = ZERO
CALL DMAKE( 'GE', ' ', ' ', 1, ML, Y, 1,&
  YY, ABS( INCY ), 0, ML - 1,&
  RESET, TRANSL )
!
NC = NC + 1
!
! Save every datum before calling the
! subroutine.
!
TRANSS = TRANS
MS = M
NS = N
KLS = KL
KUS = KU
ALS = ALPHA
DO 10 I = 1, LAA
  AS( I ) = AA( I )
10 CONTINUE
LDAS = LDA
DO 20 I = 1, LX
  XS( I ) = XX( I )
20 CONTINUE
INCXS = INCX
BLS = BETA
DO 30 I = 1, LY
  YS( I ) = YY( I )
30 CONTINUE
INCYS = INCY
!
! Call the subroutine.
!
IF( FULL )THEN
  IF( TRACE )&
    WRITE( NTRA, FMT = 9994 )NC, SNAME,&
      TRANS, M, N, ALPHA, LDA, INCX, BETA,&
      INCY
  IF( REWI )&
    REWIND NTRA
  CALL DGEMV( TRANS, M, N, ALPHA, AA,&
    LDA, XX, INCX, BETA, YY,&
    INCY )
ELSE IF( BANDED )THEN
  IF( TRACE )&
    WRITE( NTRA, FMT = 9995 )NC, SNAME,&
      TRANS, M, N, KL, KU, ALPHA, LDA,&
      INCX, BETA, INCY

```

```

        IF( REWI )&
        REWIND NTRA
        CALL DGBMV( TRANS, M, N, KL, KU, ALPHA,&
        AA, LDA, XX, INCX, BETA,&
        YY, INCY )
    END IF

!
!
!
    Check if error-exit was taken incorrectly.

    IF( .NOT.OK )THEN
        WRITE( NOUT, FMT = 9993 )
        FATAL = .TRUE.
        GO TO 130
    END IF

!
!
!
    See what data changed inside subroutines.

    ISAME( 1 ) = TRANS.EQ.TRANS
    ISAME( 2 ) = MS.EQ.M
    ISAME( 3 ) = NS.EQ.N
    IF( FULL )THEN
        ISAME( 4 ) = ALS.EQ.ALPHA
        ISAME( 5 ) = LDE( AS, AA, LAA )
        ISAME( 6 ) = LDAS.EQ.LDA
        ISAME( 7 ) = LDE( XS, XX, LX )
        ISAME( 8 ) = INCXS.EQ.INCX
        ISAME( 9 ) = BLS.EQ.BETA
        IF( NULL )THEN
            ISAME( 10 ) = LDE( YS, YY, LY )
        ELSE
            ISAME( 10 ) = LDERES( 'GE', ' ', 1,&
            ML, YS, YY,&
            ABS( INCY ) )
        END IF
        ISAME( 11 ) = INCYS.EQ.INCY
    ELSE IF( BANDED )THEN
        ISAME( 4 ) = KLS.EQ.KL
        ISAME( 5 ) = KUS.EQ.KU
        ISAME( 6 ) = ALS.EQ.ALPHA
        ISAME( 7 ) = LDE( AS, AA, LAA )
        ISAME( 8 ) = LDAS.EQ.LDA
        ISAME( 9 ) = LDE( XS, XX, LX )
        ISAME( 10 ) = INCXS.EQ.INCX
        ISAME( 11 ) = BLS.EQ.BETA
        IF( NULL )THEN
            ISAME( 12 ) = LDE( YS, YY, LY )
        ELSE
            ISAME( 12 ) = LDERES( 'GE', ' ', 1,&
            ML, YS, YY,&
            ABS( INCY ) )
        END IF
        ISAME( 13 ) = INCYS.EQ.INCY
    END IF

!
!
!
    If data was incorrectly changed, report
    and return.

    SAME = .TRUE.
    DO 40 I = 1, NARGS
        SAME = SAME.AND.ISAME( I )
        IF( .NOT.ISAME( I ) )&
            WRITE( NOUT, FMT = 9998 )I
    CONTINUE
40    IF( .NOT.SAME )THEN
        FATAL = .TRUE.
        GO TO 130
    END IF

!
!
!
    IF( .NOT.NULL )THEN
        Check the result.
        WRITE(NOUT,*) "DMVCH IN DCHK1"
        CALL DMVCH( TRANS, M, N, ALPHA, A,&
        NMAX, X, INCX, BETA, Y,&
        INCY, YT, G, YY, EPS, ERR,&
        FATAL, NOUT, .TRUE. )

        ERRMAX = MAX( ERRMAX, ERR )
        If got really bad answer, report and
        return.
        IF( FATAL )&
            GO TO 130
        ELSE
            Avoid repeating tests with M.le.0 or
            N.le.0.
            GO TO 110
        END IF

!
!
!
50        CONTINUE
!
60        CONTINUE
!
70        CONTINUE
!
80        CONTINUE
!
90        CONTINUE
!
100       CONTINUE
!
110      CONTINUE
!
120     CONTINUE
!
!
!
    Report result.

    IF( ERRMAX.LT.THRESH )THEN
        WRITE( NOUT, FMT = 9999 )SNAME, NC
    ELSE
        WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
    END IF
    GO TO 140

!
130 CONTINUE

```



```

      K = KB( IK )
    ELSE
      K = N - 1
    END IF
!   Set LDA to 1 more than minimum value if room.
    IF( BANDED )THEN
      LDA = K + 1
    ELSE
      LDA = N
    END IF
    IF( LDA.LT.NMAX )&
      LDA = LDA + 1
!   Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
      GO TO 100
    IF( PACKED )THEN
      LAA = ( N*( N + 1 ) )/2
    ELSE
      LAA = LDA*N
    END IF
    NULL = N.LE.0

!   DO 90 IC = 1, 2
      UPLO = ICH( IC: IC )

!   Generate the matrix A.

      TRANSL = ZERO
      CALL DMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A, NMAX, AA,&
        LDA, K, K, RESET, TRANSL )

!   DO 80 IX = 1, NINC
      INCX = INC( IX )
      LX = ABS( INCX )*N

!   Generate the vector X.

      TRANSL = HALF
      CALL DMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX,&
        ABS( INCX ), 0, N - 1, RESET, TRANSL )
      IF( N.GT.1 )THEN
        X( N/2 ) = ZERO
        XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
      END IF

!   DO 70 IY = 1, NINC
      INCY = INC( IY )
      LY = ABS( INCY )*N

!   DO 60 IA = 1, NALF
      ALPHA = ALF( IA )

!   DO 50 IB = 1, NBET
      BETA = BET( IB )

!   Generate the vector Y.

      TRANSL = ZERO
      CALL DMAKE( 'GE', ' ', ' ', 1, N, Y, 1, YY,&
        ABS( INCY ), 0, N - 1, RESET,&
        TRANSL )

!   NC = NC + 1

!   Save every datum before calling the
!   subroutine.
!
      UPLOS = UPLO
      NS = N
      KS = K
      ALS = ALPHA
      DO 10 I = 1, LAA
10        AS( I ) = AA( I )
      CONTINUE
      LDAS = LDA
      DO 20 I = 1, LX
20        XS( I ) = XX( I )
      CONTINUE
      INCXS = INCX
      BLS = BETA
      DO 30 I = 1, LY
30        YS( I ) = YY( I )
      CONTINUE
      INCYS = INCY

!   Call the subroutine.
!
      IF( FULL )THEN
        IF( TRACE )&
          WRITE( NTRA, FMT = 9993 )NC, SNAME,&
            UPLO, N, ALPHA, LDA, INCX, BETA, INCY
        IF( REWI )&
          REWIND NTRA
        CALL DSYMV( UPLO, N, ALPHA, AA, LDA, XX,&
          INCX, BETA, YY, INCY )
      ELSE IF( BANDED )THEN
        IF( TRACE )&
          WRITE( NTRA, FMT = 9994 )NC, SNAME,&
            UPLO, N, K, ALPHA, LDA, INCX, BETA,&
            INCY
        IF( REWI )&
          REWIND NTRA
        CALL DSBMV( UPLO, N, K, ALPHA, AA, LDA,&
          XX, INCX, BETA, YY, INCY )
      ELSE IF( PACKED )THEN
        IF( TRACE )&
          WRITE( NTRA, FMT = 9995 )NC, SNAME,&
            UPLO, N, ALPHA, INCX, BETA, INCY
        IF( REWI )&
          REWIND NTRA
      write(nout,*)'NC:',NC
      write(nout,*) 'UPLO, N, ALPHA, INCX, BETA, INCY'
      write(nout,*) (A1,I4,F10.3,I4,F10.3,I4') UPLO,N,ALPHA,INCX,BETA,INCY
      write(NOUT,*)'^'^^AA'

```

```

!
! DO 160 I=1,NMAX*NMAX
!   write(NOUT,'(F18.5)')AA(I)
!160   CONTINUE
!   write(NOUT,*)'^^^XX'
!   DO 170 I=1,NMAX*INCMAX
!   write(NOUT,'(F18.5)')XX(I)
!170   CONTINUE
!   write(NOUT,*)'^^^YY'
!   DO 180 I=1,NMAX*INCMAX
!   write(NOUT,'(F18.5)')YY(I)
!180   CONTINUE
!
! STOP
!       CALL DSPMV( UPLO, N, ALPHA, AA, XX, INCX,&
!               BETA, YY, INCY )
!
!   write(NOUT,*)'^^^OUT YY'
! DO 190 I=1,NMAX*INCMAX
! write(NOUT,'(F18.5)')YY(I)
!190   CONTINUE
!       END IF
!
!       Check if error-exit was taken incorrectly.
!
!       IF( .NOT.OK )THEN
!         WRITE( NOUT, FMT = 9992 )
!         FATAL = .TRUE.
!         GO TO 120
!       END IF
!
!       See what data changed inside subroutines.
!
!       ISAME( 1 ) = UPLO.EQ.UPLOS
!       ISAME( 2 ) = NS.EQ.N
!       IF( FULL )THEN
!         ISAME( 3 ) = ALS.EQ.ALPHA
!         ISAME( 4 ) = LDE( AS, AA, LAA )
!         ISAME( 5 ) = LDAS.EQ.LDA
!         ISAME( 6 ) = LDE( XS, XX, LX )
!         ISAME( 7 ) = INCXS.EQ.INCX
!         ISAME( 8 ) = BLS.EQ.BETA
!         IF( NULL )THEN
!           ISAME( 9 ) = LDE( YS, YY, LY )
!         ELSE
!           ISAME( 9 ) = LDERES( 'GE', ' ', 1, N,&
!                               YS, YY, ABS( INCY ) )
!         END IF
!         ISAME( 10 ) = INCYS.EQ.INCY
!       ELSE IF( BANDED )THEN
!         ISAME( 3 ) = KS.EQ.K
!         ISAME( 4 ) = ALS.EQ.ALPHA
!         ISAME( 5 ) = LDE( AS, AA, LAA )
!         ISAME( 6 ) = LDAS.EQ.LDA
!         ISAME( 7 ) = LDE( XS, XX, LX )
!         ISAME( 8 ) = INCXS.EQ.INCX
!         ISAME( 9 ) = BLS.EQ.BETA
!         IF( NULL )THEN
!           ISAME( 10 ) = LDE( YS, YY, LY )
!         ELSE
!           ISAME( 10 ) = LDERES( 'GE', ' ', 1, N,&
!                               YS, YY, ABS( INCY ) )
!         END IF
!         ISAME( 11 ) = INCYS.EQ.INCY
!       ELSE IF( PACKED )THEN
!         ISAME( 3 ) = ALS.EQ.ALPHA
!         ISAME( 4 ) = LDE( AS, AA, LAA )
!         ISAME( 5 ) = LDE( XS, XX, LX )
!         ISAME( 6 ) = INCXS.EQ.INCX
!         ISAME( 7 ) = BLS.EQ.BETA
!         IF( NULL )THEN
!           ISAME( 8 ) = LDE( YS, YY, LY )
!         ELSE
!           ISAME( 8 ) = LDERES( 'GE', ' ', 1, N,&
!                               YS, YY, ABS( INCY ) )
!         END IF
!         ISAME( 9 ) = INCYS.EQ.INCY
!       END IF
!
!       If data was incorrectly changed, report and
!       return.
!
!       SAME = .TRUE.
!       DO 40 I = 1, NARGS
!       SAME = SAME.AND.ISAME( I )
!       IF( .NOT.ISAME( I ) )&
!         WRITE( NOUT, FMT = 9998 )I
!40     CONTINUE
!       IF( .NOT.SAME )THEN
!         FATAL = .TRUE.
!         GO TO 120
!       END IF
!
!       IF( .NOT.NULL )THEN
!
!         Check the result.
!
!       WRITE(NOUT,*)'DCHK2 replace below N$ with N'
!       MYFLAG = .TRUE.
!       CALL DMVCH( 'NS', N, N, ALPHA, A, NMAX, X,&
!                 INCX, BETA, Y, INCY, YT, G,&
!                 YY, EPS, ERR, FATAL, NOUT,&
!                 .TRUE. )
!       MYFLAG = .FALSE.
!       ERRMAX = MAX( ERRMAX, ERR )
!       If got really bad answer, report and
!       return.
!       IF( FATAL )&
!         GO TO 120
!     ELSE
!       Avoid repeating tests with N.le.0
!       GO TO 110
!     END IF
!
!50   CONTINUE
!

```

```

60          CONTINUE
!
70          CONTINUE
!
80          CONTINUE
!
90          CONTINUE
!
100         CONTINUE
!
110        CONTINUE
!
!       Report result.
!
!       IF( ERRMAX.LT.THRESH )THEN
!         WRITE( NOUT, FMT = 9999 )SNAME, NC
!       ELSE
!         WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
!       END IF
!       GO TO 130
!
120        CONTINUE
!       WRITE( NOUT, FMT = 9996 )SNAME
!       IF( FULL )THEN
!         WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, N, ALPHA, LDA, INCX,&
!           BETA, INCY
!       ELSE IF( BANDED )THEN
!         WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, N, K, ALPHA, LDA,&
!           INCX, BETA, INCY
!       ELSE IF( PACKED )THEN
!         WRITE( NOUT, FMT = 9995 )NC, SNAME, UPLO, N, ALPHA, INCX,&
!           BETA, INCY
!       END IF
!
130        CONTINUE
!       RETURN
!
9999        FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!          'S)' )
9998        FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!          'ANGED INCORRECTLY *****' )
9997        FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!          'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!          ' - SUSPECT *****' )
9996        FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995        FORMAT( 1X, I6, ': ', A6, '(', I3, ', ', I3, ', ', F4.1, ', ', AP',&
!          ' ', X, ', I2, ', ', F4.1, ', ', Y, ', I2, ' )' )
9994        FORMAT( 1X, I6, ': ', A6, '(', I3, ', ', I3, ', ', F4.1, ', ',&
!          ' ', A, ', I3, ', X, ', I2, ', ', F4.1, ', ', Y, ', I2, ', ',&
!          ' )' )
9993        FORMAT( 1X, I6, ': ', A6, '(', I3, ', ', I3, ', ', F4.1, ', ', A, ',&
!          I3, ', X, ', I2, ', ', F4.1, ', ', Y, ', I2, ' )' )
9992        FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
!          '*****' )
!
!       End of DCHK2.
!
!       END SUBROUTINE
!       SUBROUTINE DCHK3( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!         FATAL, NIDIM, IDIM, NKB, KB, NINC, INC, NMAX,&
!         INCMAX, A, AA, AS, X, XX, XS, XT, G, Z )
!
!       Tests DTRMV, DTBMV, DTPMV, DTRSV, DTBSV and DTPSV.
!
!       Auxiliary routine for test program for Level 2 Blas.
!
!       -- Written on 10-August-1987.
!       Richard Hanson, Sandia National Labs.
!       Jeremy Du Croz, NAG Central Office.
!
!       .. Parameters ..
!       DOUBLE PRECISION    ZERO, HALF, ONE
!       PARAMETER           ( ZERO = 0.0D0, HALF = 0.5D0, ONE = 1.0D0 )
!       .. Scalar Arguments ..
!       DOUBLE PRECISION    EPS, THRESH
!       INTEGER             INCMAX, NIDIM, NINC, NKB, NMAX, NOUT, NTRA
!       LOGICAL             FATAL, REWI, TRACE
!       CHARACTER*6         SNAME
!       .. Array Arguments ..
!       DOUBLE PRECISION    A( NMAX, NMAX ), AA( NMAX*NMAX ),&
!         AS( NMAX*NMAX ), G( NMAX ),&
!         XS( NMAX*INCMAX ), XT( NMAX ),&
!         XX( NMAX*INCMAX ), Z( NMAX )
!       INTEGER             IDIM( NIDIM ), INC( NINC ), KB( NKB )
!       .. Local Scalars ..
!       DOUBLE PRECISION    ERR, ERRMAX, TRANSL
!       INTEGER             I, ICD, ICT, ICU, IK, IN, INCX, INCXS, IX, K,&
!         KS, LAA, LDA, LDAS, LX, N, NARGS, NC, NK, NS
!       INTEGER             J
!       LOGICAL             BANDED, FULL, NULL, PACKED, RESET, SAME
!       CHARACTER*1         DIAG, DIAGS, TRANS, TRANSS, UPLO, UPLOS
!       CHARACTER*2         ICHD, ICHU
!       CHARACTER*3         ICHT
!       .. Local Arrays ..
!       LOGICAL             ISAME( 13 )
!       .. External Functions ..
!       LOGICAL             LDE, LDERES
!       EXTERNAL            LDE, LDERES
!       .. External Subroutines ..
!       EXTERNAL            DMAKE, DMVCH, DTBMV, DTBSV, DTPMV, DTPSV,&
!         $ DTRMV, DTRSV
!       .. Intrinsic Functions ..
!       INTRINSIC           ABS, MAX
!       .. Scalars in Common ..
!       INTEGER             INFOT, NOUTC
!       LOGICAL             LERR, OK
!       .. Common blocks ..
!       COMMON              /INFOC/INFOT, NOUTC, OK, LERR
!       .. Data statements ..
!       DATA               ICHU/'UL'/, ICHT/'NTC'/, ICHD/'UN'/
!       .. Executable Statements ..
!       FULL = SNAME( 3: 3 ).EQ.'R'
!       BANDED = SNAME( 3: 3 ).EQ.'B'
!       PACKED = SNAME( 3: 3 ).EQ.'P'

```

```

! Define the number of arguments.
IF( FULL )THEN
  NARGS = 8
ELSE IF( BANDED )THEN
  NARGS = 9
ELSE IF( PACKED )THEN
  NARGS = 7
END IF

!
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
! Set up zero vector for DMVCH.
DO 10 I = 1, NMAX
  Z( I ) = ZERO
10 CONTINUE

!
DO 110 IN = 1, NIDIM
  N = IDIM( IN )

!
  IF( BANDED )THEN
    NK = NKB
  ELSE
    NK = 1
  END IF
  DO 100 IK = 1, NK
    IF( BANDED )THEN
      K = KB( IK )
    ELSE
      K = N - 1
    END IF
    ! Set LDA to 1 more than minimum value if room.
    IF( BANDED )THEN
      LDA = K + 1
    ELSE
      LDA = N
    END IF
    IF( LDA.LT.NMAX )&
      LDA = LDA + 1
    ! Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
      GO TO 100
    IF( PACKED )THEN
      LAA = ( N*( N + 1 ) )/2
    ELSE
      LAA = LDA*N
    END IF
    NULL = N.LE.0

!
    DO 90 ICU = 1, 2
      UPLO = ICHU( ICU: ICU )

!
      DO 80 ICT = 1, 3
        TRANS = ICHT( ICT: ICT )

!
        DO 70 ICD = 1, 2
          DIAG = ICHD( ICD: ICD )

!
          ! Generate the matrix A.
          !
          TRANSL = ZERO
          CALL DMAKE( SNAME( 2: 3 ), UPLO, DIAG, N, N, A,&
            NMAX, AA, LDA, K, K, RESET, TRANSL )

!
          DO 60 IX = 1, NINC
            INCX = INC( IX )
            LX = ABS( INCX )*N

!
            ! Generate the vector X.
            !
            TRANSL = HALF
            CALL DMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX,&
              ABS( INCX ), 0, N - 1, RESET,&
              TRANSL )
            IF( N.GT.1 )THEN
              X( N/2 ) = ZERO
              XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
            END IF

!
            NC = NC + 1

!
            ! Save every datum before calling the subroutine.
            !
            UPLOS = UPLO
            TRANS = TRANS
            DIAGS = DIAG
            NS = N
            KS = K
            DO 20 I = 1, LAA
              AS( I ) = AA( I )
20            CONTINUE
            LDAS = LDA
            DO 30 I = 1, LX
              XS( I ) = XX( I )
30            CONTINUE
            INCXS = INCX

!
            ! Call the subroutine.
            !
            IF( SNAME( 4: 5 ).EQ.'MV' )THEN
              IF( FULL )THEN
                IF( TRACE )&
                  WRITE( NTRA, FMT = 9993 )NC, SNAME,&
                    UPLO, TRANS, DIAG, N, LDA, INCX
                IF( REWI )&
                  REWIND NTRA
                CALL DTRMV( UPLO, TRANS, DIAG, N, AA, LDA,&
                  XX, INCX )
              ELSE IF( BANDED )THEN
                IF( TRACE )&
                  WRITE( NTRA, FMT = 9994 )NC, SNAME,&
                    UPLO, TRANS, DIAG, N, K, LDA, INCX
                IF( REWI )&
                  REWIND NTRA

```

```

        CALL DTBMV( UPLO, TRANS, DIAG, N, K, AA,&
        LDA, XX, INCX )
    ELSE IF( PACKED )THEN
        IF( TRACE )&
            WRITE( NTRA, FMT = 9995 )NC, SNAME,&
            UPLO, TRANS, DIAG, N, INCX
        IF( REWI )&
            REWIND NTRA
        CALL DTPMV( UPLO, TRANS, DIAG, N, AA, XX,&
            INCX )
    END IF
    ELSE IF( SNAME( 4: 5 ).EQ.'SV' )THEN
        IF( FULL )THEN
            IF( TRACE )&
                WRITE( NTRA, FMT = 9993 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, LDA, INCX
            IF( REWI )&
                REWIND NTRA
            CALL DTRSV( UPLO, TRANS, DIAG, N, AA, LDA,&
                XX, INCX )
        ELSE IF( Banded )THEN
            IF( TRACE )&
                WRITE( NTRA, FMT = 9994 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, K, LDA, INCX
            IF( REWI )&
                REWIND NTRA
            write(nout,*)'NC:',NC
            write(nout, '(2I4)', NMAX*NMAX, LDA*N
            write(nout,*) 'UPLO, TRANS, DIAG, N, K, LDA, INCX'
            write(nout, '(A1,A1,A1,I4,I4,I4,I4)') UPLO,TRANS, DIAG, N, K, LDA, INCX
            write(NOUT,*)'^^AA'
            DO 160 I=1,NMAX*NMAX
                write(NOUT, '(F18.5)')AA(I)
            CONTINUE
!160
            write(NOUT,*)'^^XX'
            DO 170 I=1,NMAX
                write(NOUT, '(F18.5)')XX(I)
            CONTINUE
!170

        CALL DTBSV( UPLO, TRANS, DIAG, N, K, AA,&
        LDA, XX, INCX )
        write(NOUT,*)'^^^OUT XX'
        DO 190 I=1,NMAX
            write(NOUT, '(F18.5)')XX(I)
        CONTINUE
!190

    ELSE IF( PACKED )THEN
        IF( TRACE )&
            WRITE( NTRA, FMT = 9995 )NC, SNAME,&
            UPLO, TRANS, DIAG, N, INCX
        IF( REWI )&
            REWIND NTRA
        CALL DTPSV( UPLO, TRANS, DIAG, N, AA, XX,&
            INCX )
    END IF
    END IF

    Check if error-exit was taken incorrectly.

    IF( .NOT.OK )THEN
        WRITE( NOUT, FMT = 9992 )
        FATAL = .TRUE.
        GO TO 120
    END IF

    See what data changed inside subroutines.

    ISAME( 1 ) = UPLO.EQ.UPLOS
    ISAME( 2 ) = TRANS.EQ.TRANS
    ISAME( 3 ) = DIAG.EQ.DIAGS
    ISAME( 4 ) = NS.EQ.N
    IF( FULL )THEN
        ISAME( 5 ) = LDE( AS, AA, LAA )
        ISAME( 6 ) = LDAS.EQ.LDA
        IF( NULL )THEN
            ISAME( 7 ) = LDE( XS, XX, LX )
        ELSE
            ISAME( 7 ) = LDERES( 'GE', ' ', 1, N, XS,&
                XX, ABS( INCX ) )
        END IF
        ISAME( 8 ) = INCXS.EQ.INCX
    ELSE IF( Banded )THEN
        ISAME( 5 ) = KS.EQ.K
        ISAME( 6 ) = LDE( AS, AA, LAA )
        ISAME( 7 ) = LDAS.EQ.LDA
        IF( NULL )THEN
            ISAME( 8 ) = LDE( XS, XX, LX )
        ELSE
            ISAME( 8 ) = LDERES( 'GE', ' ', 1, N, XS,&
                XX, ABS( INCX ) )
        END IF
        ISAME( 9 ) = INCXS.EQ.INCX
    ELSE IF( PACKED )THEN
        ISAME( 5 ) = LDE( AS, AA, LAA )
        IF( NULL )THEN
            ISAME( 6 ) = LDE( XS, XX, LX )
        ELSE
            ISAME( 6 ) = LDERES( 'GE', ' ', 1, N, XS,&
                XX, ABS( INCX ) )
        END IF
        ISAME( 7 ) = INCXS.EQ.INCX
    END IF

    If data was incorrectly changed, report and
    return.

    SAME = .TRUE.
    DO 40 I = 1, NARGS
        SAME = SAME.AND.ISAME( I )
        IF( .NOT.ISAME( I ) )&
            WRITE( NOUT, FMT = 9998 )I
    CONTINUE

```

40

```

        IF( .NOT.SAME )THEN
            FATAL = .TRUE.
            GO TO 120
        END IF

!
        IF( .NOT.NULL )THEN
            IF( SNAME( 4: 5 ).EQ.'MV' )THEN

!
!
!
                Check the result.

                CALL DMVCH( TRANS, N, N, ONE, A, NMAX, X,&
                    INCX, ZERO, Z, INCX, XT, G,&
                    XX, EPS, ERR, FATAL, NOUT,&
                    .TRUE. )

                ELSE IF( SNAME( 4: 5 ).EQ.'SV' )THEN

!
!
!
                    Compute approximation to original vector.

                    DO 50 I = 1, N
                        Z( I ) = XX( 1 + ( I - 1 ) )*&
                            ABS( INCX ) )
                        XX( 1 + ( I - 1 ) )*&ABS( INCX ) )&
                            = X( I )
                    50 CONTINUE
                    CALL DMVCH( TRANS, N, N, ONE, A, NMAX, Z,&
                        INCX, ZERO, X, INCX, XT, G,&
                        XX, EPS, ERR, FATAL, NOUT,&
                        .FALSE. )

                    END IF
                    ERRMAX = MAX( ERRMAX, ERR )
                    If got really bad answer, report and return.
                    IF( FATAL )&
                        GO TO 120

                ELSE
                    Avoid repeating tests with N.le.0.
                    GO TO 110
                END IF

!
!
!
        60 CONTINUE
!
!
!
        70 CONTINUE
!
!
!
        80 CONTINUE
!
!
!
        90 CONTINUE
!
!
!
        100 CONTINUE
!
!
!
        110 CONTINUE
!
!
!
        Report result.
!
        IF( ERRMAX.LT.THRESH )THEN
            WRITE( NOUT, FMT = 9999 )SNAME, NC
        ELSE
            WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
        END IF
        GO TO 130

!
!
!
        120 CONTINUE
        WRITE( NOUT, FMT = 9996 )SNAME
        IF( FULL )THEN
            WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, TRANS, DIAG, N, LDA,&
                INCX
            ELSE IF( Banded )THEN
                WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, TRANS, DIAG, N, K,&
                    LDA, INCX
            ELSE IF( Packed )THEN
                WRITE( NOUT, FMT = 9995 )NC, SNAME, UPLO, TRANS, DIAG, N, INCX
            END IF

!
!
!
        130 CONTINUE
        RETURN

!
!
!
        9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
            'S)' )
        9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
            'ANGED INCORRECTLY *****' )
        9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
            'ALLS)', '/' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
            ' - SUSPECT *****' )
        9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
        9995 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ', ' ), I3, ' ', A6, ' ',&
            'X', I2, ' )' )
        9994 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ', ' ), 2( I3, ' ', ' ),&
            ' A, ', I3, ' ', X, ' ', I2, ' )' )
        9993 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ', ' ), I3, ' ', A, ' ',&
            I3, ' ', X, ' ', I2, ' )' )
        9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
            '*****' )

!
!
!
        End of DCHK3.
!
!
!
        END SUBROUTINE
        SUBROUTINE DCHK4( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
            FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX,&
            INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G,&
            Z )

!
!
!
        Tests DGER.
!
!
!
        Auxiliary routine for test program for Level 2 Blas.
!
!
!
        -- Written on 10-August-1987.
        Richard Hanson, Sandia National Labs.
        Jeremy Du Croz, NAG Central Office.
!
!
!
        .. Parameters ..
        DOUBLE PRECISION    ZERO, HALF, ONE
        PARAMETER            ( ZERO = 0.0D0, HALF = 0.5D0, ONE = 1.0D0 )
!
!
!
        .. Scalar Arguments ..
        DOUBLE PRECISION    EPS, THRESH
        INTEGER              INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
        LOGICAL              FATAL, REWI, TRACE

```

```

CHARACTER*6      SNAME
! .. Array Arguments ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
AS( NMAX*NMAX ), G( NMAX ), X( NMAX ),&
XS( NMAX*INCMAX ), XX( NMAX*INCMAX ),&
Y( NMAX ), YS( NMAX*INCMAX ), YT( NMAX ),&
YY( NMAX*INCMAX ), Z( NMAX )
INTEGER          IDIM( NIDIM ), INC( NINC )
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, ALS, ERR, ERRMAX, TRANSL
INTEGER          I, IA, IM, IN, INCX, INCXS, INCY, INCYS, IX,&
IY, J, LAA, LDA, LDAS, LX, LY, M, MS, N, NARGS,&
NC, ND, NS
LOGICAL          NULL, RESET, SAME
! .. Local Arrays ..
DOUBLE PRECISION W( 1 )
LOGICAL          ISAME( 13 )
! .. External Functions ..
LOGICAL          LDE, LDERES
EXTERNAL          LDE, LDERES
! .. External Subroutines ..
EXTERNAL          DGER, DMAKE, DMVCH
! .. Intrinsic Functions ..
INTRINSIC        ABS, MAX, MIN
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUTC, OK, LERR
! .. Executable Statements ..
! Define the number of arguments.
NARGS = 9
!
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
DO 120 IN = 1, NIDIM
    N = IDIM( IN )
    ND = N/2 + 1
!
    DO 110 IM = 1, 2
        IF( IM.EQ.1 )&
            M = MAX( N - ND, 0 )
        IF( IM.EQ.2 )&
            M = MIN( N + ND, NMAX )
!
        Set LDA to 1 more than minimum value if room.
        LDA = M
        IF( LDA.LT.NMAX )&
            LDA = LDA + 1
!
        Skip tests if not enough room.
        IF( LDA.GT.NMAX )&
            GO TO 110
        LAA = LDA*N
        NULL = N.LE.0.OR.M.LE.0
!
        DO 100 IX = 1, NINC
            INCX = INC( IX )
            LX = ABS( INCX )*M
!
            Generate the vector X.
!
            TRANSL = HALF
            CALL DMAKE( 'GE', ' ', ' ', 1, M, X, 1, XX, ABS( INCX ),&
0, M - 1, RESET, TRANSL )
            IF( M.GT.1 )THEN
                X( M/2 ) = ZERO
                XX( 1 + ABS( INCX )*( M/2 - 1 ) ) = ZERO
            END IF
!
            DO 90 IY = 1, NINC
                INCY = INC( IY )
                LY = ABS( INCY )*N
!
                Generate the vector Y.
!
                TRANSL = ZERO
                CALL DMAKE( 'GE', ' ', ' ', 1, N, Y, 1, YY,&
ABS( INCY ), 0, N - 1, RESET, TRANSL )
                IF( N.GT.1 )THEN
                    Y( N/2 ) = ZERO
                    YY( 1 + ABS( INCY )*( N/2 - 1 ) ) = ZERO
                END IF
!
                DO 80 IA = 1, NALF
                    ALPHA = ALF( IA )
!
                    Generate the matrix A.
!
                    TRANSL = ZERO
                    CALL DMAKE( SNAME( 2: 3 ), ' ', ' ', M, N, A, NMAX,&
AA, LDA, M - 1, N - 1, RESET, TRANSL )
!
                    NC = NC + 1
!
                    Save every datum before calling the subroutine.
!
                    MS = M
                    NS = N
                    ALS = ALPHA
                    DO 10 I = 1, LAA
                        AS( I ) = AA( I )
10                     CONTINUE
                    LDAS = LDA
                    DO 20 I = 1, LX
                        XS( I ) = XX( I )
20                     CONTINUE
                    INCXS = INCX
                    DO 30 I = 1, LY
                        YS( I ) = YY( I )
30                     CONTINUE
                    INCYS = INCY

```

400

```

!
! Report result.
!
IF( ERRMAX.LT.THRESH )THEN
  WRITE( NOUT, FMT = 9999 )SNAME, NC
ELSE
  WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
END IF
GO TO 150
!
130 CONTINUE
  WRITE( NOUT, FMT = 9995 )J
!
140 CONTINUE
  WRITE( NOUT, FMT = 9996 )SNAME
  WRITE( NOUT, FMT = 9994 )NC, SNAME, M, N, ALPHA, INCX, INCY, LDA
!
150 CONTINUE
  RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL', &
'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH', &
'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C', &
'ALLS)', '/ ***** BUT WITH MAXIMUM TEST RATIO', F8.2, &
' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( '      THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '(', I3, ', ', I3, ', ', F4.1, ', X', I2, &
', Y', I2, ', A', I3, ' )' )
9993 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *', &
'*****' )
!
! End of DCHK4.
!
END SUBROUTINE
SUBROUTINE DCHK5( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, &
FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX, &
INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G, &
Z )
!
! Tests DSYR and DSPR.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
DOUBLE PRECISION ZERO, HALF, ONE
PARAMETER ( ZERO = 0.0D0, HALF = 0.5D0, ONE = 1.0D0 )
!
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
!
! .. Array Arguments ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), &
AS( NMAX*NMAX ), G( NMAX ), X( NMAX ), &
XS( NMAX*INCMAX ), XX( NMAX*INCMAX ), &
Y( NMAX ), YS( NMAX*INCMAX ), YT( NMAX ), &
YY( NMAX*INCMAX ), Z( NMAX )
INTEGER IDIM( NIDIM ), INC( NINC )
!
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, ALS, ERR, ERRMAX, TRANSL
INTEGER I, IA, IC, IN, INCX, INCXS, IX, J, JA, JJ, LAA, &
LDA, LDAS, LJ, LX, N, NARGS, NC, NS
LOGICAL FULL, NULL, PACKED, RESET, SAME, UPPER
CHARACTER*1 UPLO, UPLOS
CHARACTER*2 ICH
!
! .. Local Arrays ..
DOUBLE PRECISION W( 1 )
LOGICAL ISAME( 13 )
!
! .. External Functions ..
LOGICAL LDE, LDERES
EXTERNAL LDE, LDERES
!
! .. External Subroutines ..
EXTERNAL DMAKE, DMVCH, DSPR, DSYR
!
! .. Intrinsic Functions ..
INTRINSIC ABS, MAX
!
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
INTEGER II
LOGICAL LERR, OK
!
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA ICH/'UL'/
!
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'Y'
PACKED = SNAME( 3: 3 ).EQ.'P'
!
! Define the number of arguments.
IF( FULL )THEN
  NARGS = 7
ELSE IF( PACKED )THEN
  NARGS = 6
END IF
!
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
DO 100 IN = 1, NIDIM
  N = IDIM( IN )
!
  Set LDA to 1 more than minimum value if room.
  LDA = N
  IF( LDA.LT.NMAX )&
    LDA = LDA + 1
!
  Skip tests if not enough room.
  IF( LDA.GT.NMAX )&
    GO TO 100
  IF( PACKED )THEN

```

```

      LAA = ( N*( N + 1 ) )/2
    ELSE
      LAA = LDA*N
    END IF
!
DO 90 IC = 1, 2
  UPLO = ICH( IC: IC )
  UPPER = UPLO.EQ.'U'
!
DO 80 IX = 1, NINC
  INCX = INC( IX )
  LX = ABS( INCX )*N
!
!       Generate the vector X.
!
  TRANSL = HALF
  CALL DMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX, ABS( INCX ), &
    0, N - 1, RESET, TRANSL )
  IF( N.GT.1 ) THEN
    X( N/2 ) = ZERO
    XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
  END IF
!
DO 70 IA = 1, NALF
  ALPHA = ALF( IA )
  NULL = N.LE.0.OR.ALPHA.EQ.ZERO
!
!       Generate the matrix A.
!
  TRANSL = ZERO
!       write(NOUT,*)'^^^DMAKE OUT AA BEFORE'
!       DO 380 I=1,NMAX*NMAX
!         write(NOUT,'(I4,F18.5)')I,AA(I)
!         CONTINUE
!380      CALL DMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A, NMAX,&
!         AA, LDA, N - 1, N - 1, RESET, TRANSL )
!       write(NOUT,'(A,I4)')NC:NC
!       write(NOUT,'(A2,I4,I4,L5,F18.5)')SNAME(2:3),UPLO,N,N,NMAX,LDA,N-1,N-1,RESET,TRANSL
!
!       write(NOUT,*)'^^^DMAKE OUT AA'
!       DO 280 I=1,NMAX*NMAX
!         write(NOUT,'(I4,F18.5)')I,AA(I)
!         CONTINUE
!280      STOP
!       IF( SNAME(1:4) .EQ. 'DSYR' ) THEN
!         STOP
!       ENDIF
!
!       NC = NC + 1
!
!       Save every datum before calling the subroutine.
!
      UPLOS = UPLO
      NS = N
      ALS = ALPHA
      DO 10 I = 1, LAA
        AS( I ) = AA( I )
      CONTINUE
      LDAS = LDA
      DO 20 I = 1, LX
        XS( I ) = XX( I )
      CONTINUE
      INCXS = INCX
!
!       Call the subroutine.
!
      IF( FULL ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9993 ) NC, SNAME, UPLO, N, &
            ALPHA, INCX, LDA
        IF( REWI ) &
          REWIND NTRA
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        write(nout,'(A,5I4)')NC:NC,IN,IC,IX,IA
!
        write(nout,*) 'UPLO, N, ALPHA, INCX, LDA'
        write(nout,'(A,I4,F10.3,2I4)') UPLO,N,ALPHA, INCX,LDA
        write(NOUT,*)'^^^XX'
        DO 160 I=1,NMAX*NMAX
          write(NOUT,'(F18.5)')XX(I)
!160      CONTINUE
!
        write(NOUT,*)'^^^AA'
        DO 180 I=1,NMAX*NMAX
          write(NOUT,'(F18.5)')AA(I)
!180      CONTINUE
!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        CALL DSYR( UPLO, N, ALPHA, XX, INCX, AA, LDA )
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        write(NOUT,*)'^^^OUT AA'
        DO 190 I=1,NMAX*NMAX
          write(NOUT,'(F18.5)')AA(I)
!190      CONTINUE
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      ELSE IF( PACKED ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9994 ) NC, SNAME, UPLO, N, &
            ALPHA, INCX
        IF( REWI ) &
          REWIND NTRA
        CALL DSPR( UPLO, N, ALPHA, XX, INCX, AA )
      END IF
!
!       Check if error-exit was taken incorrectly.
!
      IF( .NOT.OK ) THEN
        WRITE( NOUT, FMT = 9992 )
        FATAL = .TRUE.
        GO TO 120
      END IF
!
!       See what data changed inside subroutines.

```

```

!
ISAME( 1 ) = UPLO.EQ.UPLOS
ISAME( 2 ) = NS.EQ.N
ISAME( 3 ) = ALS.EQ.ALPHA
ISAME( 4 ) = LDE( XS, XX, LX )
ISAME( 5 ) = INCXS.EQ.INCX
IF( NULL )THEN
    ISAME( 6 ) = LDE( AS, AA, LAA )
ELSE
    ISAME( 6 ) = LDERES( SNAME( 2: 3 ), UPLO, N, N, AS,&
        AA, LDA )
END IF
IF( .NOT.PACKED )THEN
    ISAME( 7 ) = LDAS.EQ.LDA
END IF
!
! If data was incorrectly changed, report and return.
!
SAME = .TRUE.
DO 30 I = 1, NARGS
    SAME = SAME.AND.ISAME( I )
    IF( .NOT.ISAME( I ) )&
        WRITE( NOUT, FMT = 9998 ) I
30 CONTINUE
IF( .NOT.SAME )THEN
    FATAL = .TRUE.
    GO TO 120
END IF
!
IF( .NOT.NULL )THEN
!
! Check the result column by column.
!
IF( INCX.GT.0 )THEN
    DO 40 I = 1, N
        Z( I ) = X( I )
40 CONTINUE
    ELSE
        DO 50 I = 1, N
            Z( I ) = X( N - I + 1 )
50 CONTINUE
        END IF
        JA = 1
        DO 60 J = 1, N
            W( 1 ) = Z( J )
            IF( UPPER )THEN
                JJ = 1
                LJ = J
            ELSE
                JJ = J
                LJ = N - J + 1
            END IF
            write(nout, '(A,8I4,A)') 'NC:',NC,IN,IC,IX,IA,J,JJ,JA,' BEF DMVCH'
            write(NOUT,*) 'M,ALPHA,NMAX,ERR/A,X,Y,YT,G,YY BEFORE'
            write(NOUT, '(I4,F18.3,I4,F18.3)') lj,alpha, lj,err

            //a=Z(jj) tmp2d = get_2dfromlid(nmax, jj - 1, lj, z);
            write(NOUT,*) 'Z: JJ=',JJ
!
! DO 610 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,Z(II)
!     CONTINUE
!     write(NOUT,*) 'W:'
!     write(NOUT, '(F18.3)') W(1)
!     write(NOUT,*) 'A: JJ,J=',JJ,J
! DO 611 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,A(II,J)
!     CONTINUE
!     write(NOUT,*) 'YT:'
! DO 612 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,YT(II)
!     CONTINUE
!     write(NOUT,*) 'G:'
! DO 613 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,G(II)
!     CONTINUE
!     write(NOUT,*) 'AA: JA=',JA
! DO 614 II=1,NMAX*NMAX
!     write(NOUT, '(I4,F18.3)') II,AA(II)
!     CONTINUE
!     write(nout,*) 'IN DMVCH:'
!     CALL DMVCH( 'N', LJ, 1, ALPHA, Z( JJ ), LJ, W,&
!         1, ONE, A( JJ, J ), 1, YT, G,&
!         AA( JA ), EPS, ERR, FATAL, NOUT,&
!         .TRUE. )
!
!     write(NOUT,*) 'M,ALPHA,NMAX,ERR/A,X,Y,YT,G,YY'
!     write(NOUT, '(I4,F18.3,I4,F18.3)') lj,alpha, lj,err

            //a=Z(jj) tmp2d = get_2dfromlid(nmax, jj - 1, lj, z);
            write(NOUT,*) 'Z: JJ=',JJ
!
! DO 600 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,Z(II)
!     CONTINUE
!     write(NOUT,*) 'W:'
!     write(NOUT, '(F18.3)') W(1)
!     write(NOUT,*) 'A: JJ,J=',JJ,J
! DO 601 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,A(II,J)
!     CONTINUE
!     write(NOUT,*) 'YT:'
! DO 602 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,YT(II)
!     CONTINUE
!     write(NOUT,*) 'G:'
! DO 603 II=1,NMAX
!     write(NOUT, '(I4,F18.3)') II,G(II)
!     CONTINUE
!     write(NOUT,*) 'AA: JA=',JA
! DO 604 II=1,NMAX*NMAX
!     write(NOUT, '(I4,F18.3)') II,AA(II)
!     CONTINUE

```

```

        IF( FULL )THEN
            IF( UPPER )THEN
                JA = JA + LDA
            ELSE
                JA = JA + LDA + 1
            END IF
        ELSE
            JA = JA + LJ
        END IF
        ERRMAX = MAX( ERRMAX, ERR )
!       write(NOUT,'(L5,F18.5)')FATAL,ERRMAX
!       If got really bad answer, report and return.
        IF( FATAL )&
            GO TO 110
60      CONTINUE
    ELSE
!       Avoid repeating tests if N.le.0.
        IF( N.LE.0 )&
            GO TO 100
        END IF
!
70      CONTINUE
!
80      CONTINUE
!
90      CONTINUE
!
100     CONTINUE
!
!       Report result.
!
        IF( ERRMAX.LT.THRESH )THEN
            WRITE( NOUT,FMT = 9999 )SNAME, NC
        ELSE
            WRITE( NOUT,FMT = 9997 )SNAME, NC, ERRMAX
        END IF
        GO TO 130
!
110     CONTINUE
        WRITE( NOUT, FMT = 9995 )J
!
120     CONTINUE
        WRITE( NOUT, FMT = 9996 )SNAME
        IF( FULL )THEN
            WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, N, ALPHA, INCX, LDA
        ELSE IF( PACKED )THEN
            WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, N, ALPHA, INCX
        END IF
!
130     CONTINUE
        RETURN
!
9999    FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
'S') ' )
9998    FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
'ANGED INCORRECTLY *****' )
9997    FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
'ALLS)', '/' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
' - SUSPECT *****' )
9996    FORMAT( ' *****', A6, ' FAILED ON CALL NUMBER:' )
9995    FORMAT( '      THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994    FORMAT( 1X, I6, ': ', A6, '(', '', A1, ' ', I3, ' ', F4.1, ' ', X, ',&
I2, ' ', A1 )
9993    FORMAT( 1X, I6, ': ', A6, '(', '', A1, ' ', I3, ' ', F4.1, ' ', X, ',&
I2, ' ', A, ' ', I3, ' ' )
9992    FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
'*****' )
!
!       End of DCHK5.
!
        END SUBROUTINE
        SUBROUTINE DCHK6( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX,&
INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G,&
Z )
!
!       Tests DSYR2 and DSPR2.
!
!       Auxiliary routine for test program for Level 2 Blas.
!
!       -- Written on 10-August-1987.
!       Richard Hanson, Sandia National Labs.
!       Jeremy Du Croz, NAG Central Office.
!
!       .. Parameters ..
        DOUBLE PRECISION    ZERO, HALF, ONE
        PARAMETER            ( ZERO = 0.0D0, HALF = 0.5D0, ONE = 1.0D0 )
!       .. Scalar Arguments ..
        DOUBLE PRECISION    EPS, THRESH
        INTEGER              INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
        LOGICAL              FATAL, REWI, TRACE
        CHARACTER*6          SNAME
!       .. Array Arguments ..
        DOUBLE PRECISION    A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
AS( NMAX*NMAX ), G( NMAX ), X( NMAX ),&
XS( NMAX*INCMAX ), XX( NMAX*INCMAX ),&
Y( NMAX ), YS( NMAX*INCMAX ), YT( NMAX ),&
YY( NMAX*INCMAX ), Z( NMAX, 2 )
        INTEGER              IDIM( NIDIM ), INC( NINC )
!       .. Local Scalars ..
        DOUBLE PRECISION    ALPHA, ALS, ERR, ERRMAX, TRANSL
        INTEGER              I, IA, IC, IN, INCX, INCXS, INCY, INCYS, IX,&
IY, J, JA, JJ, LAA, LDA, LDAS, LJ, LX, LY, N,&
NARGS, NC, NS
        LOGICAL              FULL, NULL, PACKED, RESET, SAME, UPPER
        CHARACTER*1          UPLO, UPLOS
        CHARACTER*2          ICH
!       .. Local Arrays ..
        DOUBLE PRECISION    W( 2 )
        LOGICAL              ISAME( 13 )
!       .. External Functions ..
        LOGICAL              LDE, LDERES
        EXTERNAL              LDE, LDERES

```

```

! .. External Subroutines ..
! EXTERNAL          DMAKE, DMVCH, DSPR2, DSYR2
! .. Intrinsic Functions ..
! INTRINSIC          ABS, MAX
! .. Scalars in Common ..
! INTEGER            INFOT, NOUTC
! LOGICAL            LERR, OK
! .. Common blocks ..
! COMMON             /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
! DATA              ICH/'UL'/
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'Y'
PACKED = SNAME( 3: 3 ).EQ.'P'
! Define the number of arguments.
IF( FULL )THEN
    NARGS = 9
ELSE IF( PACKED )THEN
    NARGS = 8
END IF

!
NC = 0
RESET = .TRUE.
ERRMAX = ZERO

!
DO 140 IN = 1, NIDIM
    N = IDIM( IN )
    ! Set LDA to 1 more than minimum value if room.
    LDA = N
    IF( LDA.LT.NMAX )&
        LDA = LDA + 1
    ! Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
        GO TO 140
    IF( PACKED )THEN
        LAA = ( N*( N + 1 ) )/2
    ELSE
        LAA = LDA*N
    END IF

!
DO 130 IC = 1, 2
    UPLO = ICH( IC: IC )
    UPPER = UPLO.EQ.'U'

!
DO 120 IX = 1, NINC
    INCX = INC( IX )
    LX = ABS( INCX )*N

!
    ! Generate the vector X.
    !
    TRANSL = HALF
    CALL DMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX, ABS( INCX ),&
        0, N - 1, RESET, TRANSL )
    IF( N.GT.1 )THEN
        X( N/2 ) = ZERO
        XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
    END IF

!
DO 110 IY = 1, NINC
    INCY = INC( IY )
    LY = ABS( INCY )*N

!
    ! Generate the vector Y.
    !
    TRANSL = ZERO
    CALL DMAKE( 'GE', ' ', ' ', 1, N, Y, 1, YY,&
        ABS( INCY ), 0, N - 1, RESET, TRANSL )
    IF( N.GT.1 )THEN
        Y( N/2 ) = ZERO
        YY( 1 + ABS( INCY )*( N/2 - 1 ) ) = ZERO
    END IF

!
DO 100 IA = 1, NALF
    ALPHA = ALF( IA )
    NULL = N.LE.0.OR.ALPHA.EQ.ZERO

!
    ! Generate the matrix A.
    !
    TRANSL = ZERO
    CALL DMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A,&
        NMAX, AA, LDA, N - 1, N - 1, RESET,&
        TRANSL )

!
    write(NOUT,*)'NC:',NC
    NC = NC + 1

!
    ! Save every datum before calling the subroutine.
    !
    UPLOS = UPLO
    NS = N
    ALS = ALPHA
    DO 10 I = 1, LAA
        AS( I ) = AA( I )
    CONTINUE
    LDAS = LDA
    DO 20 I = 1, LX
        XS( I ) = XX( I )
    CONTINUE
    INCXS = INCX
    DO 30 I = 1, LY
        YS( I ) = YY( I )
    CONTINUE
    INCYS = INCY

!
    ! Call the subroutine.
    !
    IF( FULL )THEN
        IF( TRACE )&
            WRITE( NTRA, FMT = 9993 )NC, SNAME, UPLO, N,&
                ALPHA, INCX, INCY, LDA
        IF( REWI )&
            REWIND NTRA
    write(NOUT,*)'BEF DSYR2'
    CALL DSYR2( UPLO, N, ALPHA, XX, INCX, YY, INCY,&

```

```

!           AA, LDA )
!       write(NOUT,*)'OUT DSYR2'
!       ELSE IF( PACKED )THEN
!           IF( TRACE )&
!               WRITE( NTRA, FMT = 9994 )NC, SNAME, UPLO, N,&
!               ALPHA, INCX, INCY
!           IF( REWI )&
!               REWIND NTRA
!       write(NOUT,*)'BEF DSPR2'
!       CALL DSPR2( UPLO, N, ALPHA, XX, INCX, YY, INCY,&
!           AA )
!       write(NOUT,*)'OUT DSPR2'
!       END IF
!
!       Check if error-exit was taken incorrectly.
!
!       IF( .NOT.OK )THEN
!           WRITE( NOUT, FMT = 9992 )
!           FATAL = .TRUE.
!           GO TO 160
!       END IF
!
!       See what data changed inside subroutines.
!
!       ISAME( 1 ) = UPLO.EQ.UPLOS
!       ISAME( 2 ) = NS.EQ.N
!       ISAME( 3 ) = ALS.EQ.ALPHA
!       ISAME( 4 ) = LDE( XS, XX, LX )
!       ISAME( 5 ) = INCXS.EQ.INCX
!       ISAME( 6 ) = LDE( YS, YY, LY )
!       ISAME( 7 ) = INCYS.EQ.INCY
!       IF( NULL )THEN
!           ISAME( 8 ) = LDE( AS, AA, LAA )
!       ELSE
!           ISAME( 8 ) = LDERES( SNAME( 2: 3 ), UPLO, N, N,&
!               AS, AA, LDA )
!       END IF
!       IF( .NOT.PACKED )THEN
!           ISAME( 9 ) = LDAS.EQ.LDA
!       END IF
!
!       If data was incorrectly changed, report and return.
!
!       SAME = .TRUE.
!       DO 40 I = 1, NARGS
!           SAME = SAME.AND.ISAME( I )
!           IF( .NOT.ISAME( I ) )&
!               WRITE( NOUT, FMT = 9998 )I
!       CONTINUE
!       IF( .NOT.SAME )THEN
!           FATAL = .TRUE.
!           GO TO 160
!       END IF
!
!       IF( .NOT.NULL )THEN
!
!           Check the result column by column.
!
!           IF( INCX.GT.0 )THEN
!               DO 50 I = 1, N
!                   Z( I, 1 ) = X( I )
!               CONTINUE
!           ELSE
!               DO 60 I = 1, N
!                   Z( I, 1 ) = X( N - I + 1 )
!               CONTINUE
!           END IF
!           IF( INCY.GT.0 )THEN
!               DO 70 I = 1, N
!                   Z( I, 2 ) = Y( I )
!               CONTINUE
!           ELSE
!               DO 80 I = 1, N
!                   Z( I, 2 ) = Y( N - I + 1 )
!               CONTINUE
!           END IF
!           JA = 1
!           DO 90 J = 1, N
!               W( 1 ) = Z( J, 2 )
!               W( 2 ) = Z( J, 1 )
!               IF( UPPER )THEN
!                   JJ = 1
!                   LJ = J
!               ELSE
!                   JJ = J
!                   LJ = N - J + 1
!               END IF
!               CALL DMVCH( 'N', LJ, 2, ALPHA, Z( JJ, 1 ),&
!                   NMAX, W, 1, ONE, A( JJ, J ), 1,&
!                   YT, G, AA( JA ), EPS, ERR, FATAL,&
!                   NOUT, .TRUE. )
!               IF( FULL )THEN
!                   IF( UPPER )THEN
!                       JA = JA + LDA
!                   ELSE
!                       JA = JA + LDA + 1
!                   END IF
!               ELSE
!                   JA = JA + LJ
!               END IF
!               ERRMAX = MAX( ERRMAX, ERR )
!               If got really bad answer, report and return.
!               IF( FATAL )&
!                   GO TO 150
!           CONTINUE
!       ELSE
!           Avoid repeating tests with N.le.0.
!           IF( N.LE.0 )&
!               GO TO 140
!       END IF
!
!       CONTINUE
!
!       CONTINUE
!
!       CONTINUE

```

```

120      CONTINUE
!
130      CONTINUE
!
140      CONTINUE
!
      Report result.
!
      IF( ERRMAX.LT.THRESH )THEN
        WRITE( NOUT, FMT = 9999 )SNAME, NC
      ELSE
        WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
      END IF
      GO TO 170
!
150      CONTINUE
      WRITE( NOUT, FMT = 9995 )J
!
160      CONTINUE
      WRITE( NOUT, FMT = 9996 )SNAME
      IF( FULL )THEN
        WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, N, ALPHA, INCX,&
          INCY, LDA
      ELSE IF( PACKED )THEN
        WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, N, ALPHA, INCX, INCY
      END IF
!
170      CONTINUE
      RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL','&
         'S') ' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH','&
         'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C','&
         'ALLS)', '/' ,***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
         '- SUSPECT *****' )
9996 FORMAT( ' *****', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( ' ***** THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '(','',' Al, '',', I3, ',', F4.1, ' X,',&
         '12, ', Y, ', I2, ', AP)
9993 FORMAT( 1X, I6, ': ', A6, '(','',' Al, '',', I3, ',', F4.1, ' X,',&
         '12, ', Y, ', I2, ', A, ', I3, ')
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
         '*****' )
!
      End of DCHK6.
!
      END SUBROUTINE
      SUBROUTINE DCHKE( ISNUM, SRNAMT, NOUT )
!
! Tests the error exits from the Level 2 Blas.
! Requires a special version of the error-handling routine XERBLA.
! ALPHA, BETA, A, X and Y should not need to be defined.
!
! Auxiliary routine for test program for Level 2 Blas.
!
-- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central office.
!
! .. Scalar Arguments ..
INTEGER       ISNUM, NOUT
CHARACTER*6   SRNAMT
! .. Scalars in Common ..
INTEGER       INFOT, NOUTC
LOGICAL       LERR, OK
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, BETA
! .. Local Arrays ..
DOUBLE PRECISION A( 1, 1 ), X( 1 ), Y( 1 )
! .. External Subroutines ..
EXTERNAL      CHKXER, DGEMV, DGBMV, DGER, DSBMV, DSPMV, DSPR,
$             DSPR2, DSYMV, DSYR, DYSR2, DTBMV, DTBSV, DTPMV,
$             DTPSV, DTRMV, DTSRV
! .. Common blocks ..
COMMON        /INFOC/INFOT, NOUTC, OK, LERR
! .. Executable Statements ..
OK = .FALSE. by the special version of XERBLA or by CHKXER
if anything is wrong.
OK = .TRUE.
LERR is set to .TRUE. by the special version of XERBLA each time
it is called, and is then tested and re-set by CHKXER.
LERR = .FALSE.
GO TO ( 10, 20, 30, 40, 50, 60, 70, 80,&
     90, 100, 110, 120, 130, 140, 150,&
     160 )ISNUM
10 INFOT = 1
CALL DGEMV( '/', 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DGEMV( 'N', -1, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DGEMV( 'N', 0, -1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL DGEMV( 'N', 2, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL DGEMV( 'N', 0, 0, ALPHA, A, 1, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 11
CALL DGEMV( 'N', 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
20 INFOT = 1
CALL DGBMV( '/', 0, 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DGBMV( 'N', -1, 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
```

```

CALL DGBMV( 'N', 0, -1, 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DGBMV( 'N', 0, 0, -1, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DGBMV( 'N', 2, 0, 0, -1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL DGBMV( 'N', 0, 0, 1, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 10
CALL DGBMV( 'N', 0, 0, 0, 0, ALPHA, A, 1, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 13
CALL DGBMV( 'N', 0, 0, 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
30 INFOT = 1
CALL DSYMV( '/', 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSYMV( 'U', -1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DSYMV( 'U', 2, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DSYMV( 'U', 0, ALPHA, A, 1, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 10
CALL DSYMV( 'U', 0, ALPHA, A, 1, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
40 INFOT = 1
CALL DSBMV( '/', 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSBMV( 'U', -1, 0, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DSBMV( 'U', 0, -1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL DSBMV( 'U', 0, 1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL DSBMV( 'U', 0, 0, ALPHA, A, 1, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 11
CALL DSBMV( 'U', 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
50 INFOT = 1
CALL DSPMV( '/', 0, ALPHA, A, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSPMV( 'U', -1, ALPHA, A, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL DSPMV( 'U', 0, ALPHA, A, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL DSPMV( 'U', 0, ALPHA, A, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
60 INFOT = 1
CALL DTRMV( '/', 'N', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTRMV( 'U', '/', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DTRMV( 'U', 'N', '/', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTRMV( 'U', 'N', 'N', -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL DTRMV( 'U', 'N', 'N', 2, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL DTRMV( 'U', 'N', 'N', 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
70 INFOT = 1
CALL DTBMV( '/', 'N', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTBMV( 'U', '/', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DTBMV( 'U', 'N', '/', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTBMV( 'U', 'N', 'N', -1, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DTBMV( 'U', 'N', 'N', 0, -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DTBMV( 'U', 'N', 'N', 0, 1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL DTBMV( 'U', 'N', 'N', 0, 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
80 INFOT = 1
CALL DTPMV( '/', 'N', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTPMV( 'U', '/', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3

```

```

CALL DTPMV( 'U', 'N', '/', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTPMV( 'U', 'N', 'N', -1, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DTPMV( 'U', 'N', 'N', 0, A, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
90 INFOT = 1
CALL DTRSV( '/', 'N', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTRSV( 'U', '/', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DTRSV( 'U', 'N', '/', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTRSV( 'U', 'N', 'N', -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL DTRSV( 'U', 'N', 'N', 2, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL DTRSV( 'U', 'N', 'N', 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
100 INFOT = 1
CALL DTBSV( '/', 'N', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTBSV( 'U', '/', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DTBSV( 'U', 'N', '/', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTBSV( 'U', 'N', 'N', -1, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DTBSV( 'U', 'N', 'N', 0, -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DTBSV( 'U', 'N', 'N', 0, 1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL DTBSV( 'U', 'N', 'N', 0, 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
110 INFOT = 1
CALL DTPSV( '/', 'N', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DTPSV( 'U', '/', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DTPSV( 'U', 'N', '/', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DTPSV( 'U', 'N', 'N', -1, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DTPSV( 'U', 'N', 'N', 0, A, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
120 INFOT = 1
CALL DGER( -1, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DGER( 0, -1, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DGER( 0, 0, ALPHA, X, 0, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DGER( 0, 0, ALPHA, X, 1, Y, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL DGER( 2, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
130 INFOT = 1
CALL DSYR( '/', 0, ALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSYR( 'U', -1, ALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DSYR( 'U', 0, ALPHA, X, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL DSYR( 'U', 2, ALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
140 INFOT = 1
CALL DSPR( '/', 0, ALPHA, X, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSPR( 'U', -1, ALPHA, X, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DSPR( 'U', 0, ALPHA, X, 0, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 170
150 INFOT = 1
CALL DSYR2( '/', 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DSYR2( 'U', -1, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL DSYR2( 'U', 0, ALPHA, X, 0, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )

```

```

      INFOT = 7
      CALL DSYR2( 'U', 0, ALPHA, X, 1, Y, 0, A, 1 )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
      INFOT = 9
      CALL DSYR2( 'U', 2, ALPHA, X, 1, Y, 1, A, 1 )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
      GO TO 170
160  INFOT = 1
      CALL DSPR2( '/', 0, ALPHA, X, 1, Y, 1, A )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
      INFOT = 2
      CALL DSPR2( 'U', -1, ALPHA, X, 1, Y, 1, A )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
      INFOT = 5
      CALL DSPR2( 'U', 0, ALPHA, X, 0, Y, 1, A )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
      INFOT = 7
      CALL DSPR2( 'U', 0, ALPHA, X, 1, Y, 0, A )
      CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
! 170 IF( OK )THEN
!       WRITE( NOUT, FMT = 9999 )SRNAMT
!     ELSE
!       WRITE( NOUT, FMT = 9998 )SRNAMT
!     END IF
!     RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE TESTS OF ERROR-EXITS' )
9998 FORMAT( ' ***** ', A6, ' FAILED THE TESTS OF ERROR-EXITS *****', &
'***' )
!
! End of DCHKE.
!
!
      END SUBROUTINE
      SUBROUTINE DMAKE( TYPE, UPLO, DIAG, M, N, A, NMAX, AA, LDA, KL, &
        KU, RESET, TRANSL )
!
! Generates values for an M by N matrix A within the bandwidth
! defined by KL and KU.
! Stores the values in the array AA in the data structure required
! by the routine, with unwanted elements set to rogue value.
!
! TYPE is 'GE', 'GB', 'SY', 'SB', 'SP', 'TR', 'TB' OR 'TP'.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER      ( ZERO = 0.0D0, ONE = 1.0D0 )
      DOUBLE PRECISION ROGUE
      PARAMETER      ( ROGUE = -1.0D10 )
! .. Scalar Arguments ..
      DOUBLE PRECISION TRANSL
      INTEGER          KL, KU, LDA, M, N, NMAX
      LOGICAL          RESET
      CHARACTER*1      DIAG, UPLO
      CHARACTER*2      TYPE
! .. Array Arguments ..
      DOUBLE PRECISION A( NMAX, * ), AA( * )
! .. Local Scalars ..
      INTEGER          I, I1, I2, I3, IBEG, IEND, IOFF, J, KK
      LOGICAL          GEN, LOWER, SYM, TRI, UNIT, UPPER
      INTEGER          NOUTC
      COMMON           /INFOC/NOUTC
! .. External Functions ..
      DOUBLE PRECISION DBEG
      EXTERNAL         DBEG
! .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
! .. Executable Statements ..
      GEN = TYPE( 1: 1 ).EQ.'G'
      SYM = TYPE( 1: 1 ).EQ.'S'
      TRI = TYPE( 1: 1 ).EQ.'T'
      UPPER = ( SYM.OR.TRI ).AND.UPLO.EQ.'U'
      LOWER = ( SYM.OR.TRI ).AND.UPLO.EQ.'L'
      UNIT = TRI.AND.DIAG.EQ.'U'
      write(NOUTC,*)TYPE,UPLO,DIAG,GEN,SYM,TRI,UPPER,LOWER,UNIT
      write(NOUTC,'(3A2)')TYPE,UPLO,DIAG
      write(NOUTC,'(6L2)')GEN,SYM,TRI,UPPER,LOWER,UNIT
!
! Generate data in array A.
!
      DO 20 J = 1, N
        DO 10 I = 1, M
          IF( GEN.OR.( UPPER.AND.I.LE.J ).OR.( LOWER.AND.I.GE.J ) )&
            THEN
            IF( ( I.LE.J.AND.J - I.LE.KU ).OR.&
              ( I.GE.J.AND.I - J.LE.KL ) )THEN
              A( I, J ) = DBEG( RESET ) + TRANSL
            ELSE
              A( I, J ) = ZERO
            END IF
            write(noutc,'(2I4,F18.5)')I,J,A(I,J)
            IF( I.NE.J )THEN
              IF( SYM )THEN
                A( J, I ) = A( I, J )
              ELSE IF( TRI )THEN
                A( J, I ) = ZERO
              END IF
              write(noutc,'(2I4,F18.5)')I,J,A(J,I)
            END IF
          END IF
10      CONTINUE
          IF( TRI )&
            A( J, J ) = A( J, J ) + ONE
          IF( UNIT )&
            A( J, J ) = ONE
20      CONTINUE
!
! Store elements in array AS in data structure required by routine.

```

```

!
      IF( TYPE.EQ.'GE' )THEN
        DO 50 J = 1, N
          DO 30 I = 1, M
            AA( I + ( J - 1 ) * LDA ) = A( I, J )
          CONTINUE
          DO 40 I = M + 1, LDA
            AA( I + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
        CONTINUE
      ELSE IF( TYPE.EQ.'GB' )THEN
        DO 90 J = 1, N
          DO 60 I1 = 1, KU + 1 - J
            AA( I1 + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
          DO 70 I2 = I1, MIN( KL + KU + 1, KU + 1 + M - J )
            AA( I2 + ( J - 1 ) * LDA ) = A( I2 + J - KU - 1, J )
          CONTINUE
          DO 80 I3 = I2, LDA
            AA( I3 + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
        CONTINUE
      ELSE IF( TYPE.EQ.'SY'.OR.TYPE.EQ.'TR' )THEN
        DO 130 J = 1, N
          IF( UPPER )THEN
            IBEG = 1
            IF( UNIT )THEN
              IEND = J - 1
            ELSE
              IEND = J
            END IF
          ELSE
            IF( UNIT )THEN
              IBEG = J + 1
            ELSE
              IBEG = J
            END IF
            IEND = N
          END IF
          DO 100 I = 1, IBEG - 1
            AA( I + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
          DO 110 I = IBEG, IEND
            AA( I + ( J - 1 ) * LDA ) = A( I, J )
          CONTINUE
          DO 120 I = IEND + 1, LDA
            AA( I + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
        CONTINUE
      ELSE IF( TYPE.EQ.'SB'.OR.TYPE.EQ.'TB' )THEN
        DO 170 J = 1, N
          IF( UPPER )THEN
            KK = KL + 1
            IBEG = MAX( 1, KL + 2 - J )
            IF( UNIT )THEN
              IEND = KL
            ELSE
              IEND = KL + 1
            END IF
          ELSE
            KK = 1
            IF( UNIT )THEN
              IBEG = 2
            ELSE
              IBEG = 1
            END IF
            IEND = MIN( KL + 1, 1 + M - J )
          END IF
          DO 140 I = 1, IBEG - 1
            AA( I + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
          DO 150 I = IBEG, IEND
            AA( I + ( J - 1 ) * LDA ) = A( I + J - KK, J )
          CONTINUE
          DO 160 I = IEND + 1, LDA
            AA( I + ( J - 1 ) * LDA ) = ROGUE
          CONTINUE
        CONTINUE
      ELSE IF( TYPE.EQ.'SP'.OR.TYPE.EQ.'TP' )THEN
        IOFF = 0
        DO 190 J = 1, N
          IF( UPPER )THEN
            IBEG = 1
            IEND = J
          ELSE
            IBEG = J
            IEND = N
          END IF
          DO 180 I = IBEG, IEND
            IOFF = IOFF + 1
            AA( IOFF ) = A( I, J )
            IF( I.EQ.J )THEN
              IF( UNIT )&
                AA( IOFF ) = ROGUE
            END IF
          CONTINUE
        CONTINUE
      END IF
      RETURN
!
!   End of DMAKE.
!
      END SUBROUTINE
      SUBROUTINE DMVCH( TRANS, M, N, ALPHA, A, NMAX, X, INCX, BETA, Y,&
        INCY, YT, G, YY, EPS, ERR, FATAL, NOUT, MV )
!
!   Checks the results of the computational tests.
!
!   Auxiliary routine for test program for Level 2 Blas.
!
!   -- Written on 10-August-1987.
!   Richard Hanson, Sandia National Labs.
!   Jeremy Du Croz, NAG Central Office.
!

```

```

! .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER ( ZERO = 0.0D0, ONE = 1.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA, BETA, EPS, ERR
INTEGER INCX, INCY, M, N, NMAX, NOUT
LOGICAL FATAL, MV
CHARACTER*1 TRANS
CHARACTER*2 TRANS
! .. Array Arguments ..
DOUBLE PRECISION A( NMAX, * ), G( * ), X( * ), Y( * ), YT( * ), &
YY( * )
! .. Local Scalars ..
DOUBLE PRECISION ERRI
INTEGER I, INCXL, INCYL, IY, J, JX, KX, KY, ML, NL
LOGICAL TRAN
LOGICAL MYFLAG
COMMON /MYSTUFF/MYFLAG
! .. Intrinsic Functions ..
INTRINSIC ABS, MAX, SQRT
! .. Executable Statements ..
! write(NOUT, '(A,I4,A,I4,A,I4,A,I4)') 'Rank:',2,' D1:',M,' D2:',N,' NMAX:',NMAX

! DO 655 I=1,NMAX
!   write(NOUT, '(I4,F18.5)') I,A(I,1)
!655 CONTINUE
TRAN = TRANS.EQ.'T'.OR.TRANS.EQ.'C'
IF( TRAN )THEN
  ML = N
  NL = M
ELSE
  ML = M
  NL = N
END IF
IF( INCX.LT.0 )THEN
  KX = NL
  INCXL = -1
ELSE
  KX = 1
  INCXL = 1
END IF
IF( INCY.LT.0 )THEN
  KY = ML
  INCYL = -1
ELSE
  KY = 1
  INCYL = 1
END IF

! Compute expected result in YT using data in A, X and Y.
! Compute gauges in G.
!
IY = KY
WRITE(NOUT, '(A,L6)') 'DMVCH: ',TRAN
WRITE(NOUT, '(A,I4)') 'DMVCH: ',IY
DO 30 I = 1, ML
  YT( IY ) = ZERO
  G( IY ) = ZERO
  JX = KX
  IF( TRAN )THEN
    DO 10 J = 1, NL
      YT( IY ) = YT( IY ) + A( J, I ) * X( JX )
      G( IY ) = G( IY ) + ABS( A( J, I ) ) * X( JX )
!       WRITE(NOUT, '(A,2I4,4F18.5)') 'DMVCH: ',I,J,YT(IY),G(IY),A(J,I),X(JX)
      JX = JX + INCXL
    10 CONTINUE
  ELSE
    DO 20 J = 1, NL
      YT( IY ) = YT( IY ) + A( I, J ) * X( JX )
      G( IY ) = G( IY ) + ABS( A( I, J ) ) * X( JX )
!       WRITE(NOUT, '(A,2I4,4F18.5)') 'DMVCH: ',I,J,YT(IY),G(IY),A(I,J),X(JX)
      JX = JX + INCXL
    20 CONTINUE
  END IF
  YT( IY ) = ALPHA * YT( IY ) + BETA * Y( IY )
  G( IY ) = ABS( ALPHA ) * G( IY ) + ABS( BETA ) * Y( IY )
!   WRITE(NOUT, '(A,2F18.5)') 'DMVCH: ',YT(IY),G(IY)

  IY = IY + INCYL
30 CONTINUE

! Compute the error ratio for this result.
!
ERR = ZERO
!WRITE(NOUT,*) 'LEN TRANS:',LEN(TRANS),TRANS
DO 40 I = 1, ML
  ERRI = ABS( YT( I ) - YY( 1 + ( I - 1 ) * ABS( INCY ) ) ) / EPS
! write(NOUT, '(A,2I4,4F18.5)') 'DMVCH: ',I,1+(I-1)*ABS(INCY),ERRI,YT(I),YY(1+(I-1)*ABS(INCY)),G(I)
  IF( MYFLAG ) THEN
    WRITE(NOUT,9997) 'DMVCH: ',I,1+(I-1)*ABS(INCY),YT(I),YY(1+(I-1)*ABS(INCY)),ERRI
    WRITE(NOUT,9995) G(I),ERRI

    END IF
    IF( G( I ) .NE. ZERO ) &
      ERRI = ERRI / G( I )
    ERR = MAX( ERR, ERRI )

    IF( MYFLAG ) THEN
      WRITE(NOUT,9996) I,err,eps,err*sqrt(eps)
    END IF
    IF( ERR * SQRT( EPS ) .GE. ONE ) &
      GO TO 50
40 CONTINUE
! If the loop completes, all results are at least half accurate.
GO TO 70
! Report fatal error.

```

```

!
50 FATAL = .TRUE.
   WRITE( NOUT, FMT = 9999 )
   DO 60 I = 1, ML
     IF( MV )THEN
       WRITE( NOUT, FMT = 9998 )I, YT( I ),&
         YY( 1 + ( I - 1 ) * ABS( INCY ) )
     ELSE
       WRITE( NOUT, FMT = 9998 )I,&
         YY( 1 + ( I - 1 ) * ABS( INCY ) ), YT( I )
     END IF
   60 CONTINUE
!
70 CONTINUE
! STOP
! RETURN
!
9999 FORMAT( ' ***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HAL',&
  'F ACCURATE *****', /'          EXPECTED RESULT  COMPU',&
  'TED RESULT' )
9998 FORMAT( 1X, I7, 2G18.6 )
9997 FORMAT( A, 2I7, 3F10.3 )
9996 FORMAT( I7, 3F10.3 )
9995 FORMAT( 2F10.3 )
!
!   End of DMVCH.
!
!   END SUBROUTINE
!   LOGICAL FUNCTION LDE( RI, RJ, LR )
!
! Tests if two arrays are identical.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ..
! INTEGER          LR
! .. Array Arguments ..
! DOUBLE PRECISION RI( * ), RJ( * )
! .. Local Scalars ..
! INTEGER          I
! .. Executable Statements ..
! DO 10 I = 1, LR
!   IF( RI( I ) .NE. RJ( I ) )&
!     GO TO 20
! 10 CONTINUE
!   LDE = .TRUE.
!   GO TO 30
! 20 CONTINUE
!   LDE = .FALSE.
! 30 RETURN
!
!   End of LDE.
!
!   END FUNCTION
!   LOGICAL FUNCTION LDERES( TYPE, UPLO, M, N, AA, AS, LDA )
!
! Tests if selected elements in two arrays are equal.
!
! TYPE is 'GE', 'SY' or 'SP'.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ..
!
! .. Local Scalars ..
! INTEGER          LDA, M, N
! CHARACTER*1      UPLO
! CHARACTER*2      TYPE
! .. Array Arguments ..
! DOUBLE PRECISION AA( LDA, * ), AS( LDA, * )
! .. Local Scalars ..
! INTEGER          I, IBEG, IEND, J
! LOGICAL          UPPER
! .. Executable Statements ..
! print *, 'LDERES', m, n, lda, 65 * 65 / lda
! UPPER = UPLO.EQ.'U'
! IF( TYPE.EQ.'GE' )THEN
!   DO 20 J = 1, N
!     DO 10 I = M + 1, LDA
!       IF( AA( I, J ) .NE. AS( I, J ) )&
!         GO TO 70
! 10 CONTINUE
! 20 CONTINUE
! ELSE IF( TYPE.EQ.'SY' )THEN
!   DO 50 J = 1, N
!     IF( UPPER )THEN
!       IBEG = 1
!       IEND = J
!     ELSE
!       IBEG = J
!       IEND = N
!     END IF
!     DO 30 I = 1, IBEG - 1
!       IF( AA( I, J ) .NE. AS( I, J ) )&
!         GO TO 70
! 30 CONTINUE
!     DO 40 I = IEND + 1, LDA
!       IF( AA( I, J ) .NE. AS( I, J ) )&
!         GO TO 70
! 40 CONTINUE
! 50 CONTINUE
! END IF
!
!   LDERES = .TRUE.
!   GO TO 80
! 70 CONTINUE
!   LDERES = .FALSE.

```

```

80 RETURN
!
!   End of LDERES.
!
!   END FUNCTION
!   DOUBLE PRECISION FUNCTION DBEG( RESET )
!
!   Generates random numbers uniformly distributed between -0.5 and 0.5.
!
!   Auxiliary routine for test program for Level 2 Blas.
!
!   -- Written on 10-August-1987.
!   Richard Hanson, Sandia National Labs.
!   Jeremy Du Croz, NAG Central Office.
!
!   .. Scalar Arguments ..
!   LOGICAL          RESET
!   .. Local Scalars ..
!   INTEGER          I, IC, MI
!   .. Save statement ..
!   SAVE            I, IC, MI
!
!   .. Intrinsic Functions ..
!   INTRINSIC        DBLE
!   .. Executable Statements ..
!   IF( RESET )THEN
!       Initialize local variables.
!       MI = 891
!       I = 7
!       IC = 0
!       RESET = .FALSE.
!   END IF
!
!   The sequence of values of I is bounded between 1 and 999.
!   If initial I = 1,2,3,6,7 or 9, the period will be 50.
!   If initial I = 4 or 8, the period will be 25.
!   If initial I = 5, the period will be 10.
!   IC is used to break up the period by skipping 1 value of I in 6.
!
!   IC = IC + 1
10  I = I*MI
   I = I - 1000*( I/1000 )
   IF( IC.GE.5 )THEN
       IC = 0
       GO TO 10
   END IF
   DBEG = DBLE( I - 500 )/1001.0D0
   RETURN
!
!   End of DBEG.
!
!   END FUNCTION
!   DOUBLE PRECISION FUNCTION DDIFF( X, Y )
!
!   Auxiliary routine for test program for Level 2 Blas.
!
!   -- Written on 10-August-1987.
!   Richard Hanson, Sandia National Labs.
!   Jeremy Du Croz, NAG Central Office.
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION X, Y
!   .. Executable Statements ..
!   DDIFF = X - Y
!   RETURN
!
!   End of DDIFF.
!
!   END FUNCTION
!   SUBROUTINE CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
!   Tests whether XERBLA has detected an error when it should.
!
!   Auxiliary routine for test program for Level 2 Blas.
!
!   -- Written on 10-August-1987.
!   Richard Hanson, Sandia National Labs.
!   Jeremy Du Croz, NAG Central Office.
!
!   .. Scalar Arguments ..
!   INTEGER          INFOT, NOUT
!   LOGICAL          LERR, OK
!   CHARACTER*6      SRNAMT
!   .. Executable Statements ..
!   IF( .NOT.LERR )THEN
!       WRITE( NOUT, FMT = 9999 )INFOT, SRNAMT
!       OK = .FALSE.
!   END IF
!   LERR = .FALSE.
!   RETURN
!
9999 FORMAT( ' ***** ILLEGAL VALUE OF PARAMETER NUMBER ', I2, ' NOT D', &
'ETECTED BY ', A6, ' *****' )
!
!   End of CHKXER.
!
!   END SUBROUTINE
!   SUBROUTINE XERBLA( SRNAME, INFO )
!
!   This is a special version of XERBLA to be used only as part of
!   the test program for testing error exits from the Level 2 BLAS
!   routines.
!
!   XERBLA is an error handler for the Level 2 BLAS routines.
!
!   It is called by the Level 2 BLAS routines if an input parameter is
!   invalid.
!
!   Auxiliary routine for test program for Level 2 Blas.
!
!   -- Written on 10-August-1987.
!   Richard Hanson, Sandia National Labs.
!   Jeremy Du Croz, NAG Central Office.
!
!   .. Scalar Arguments ..
!   INTEGER          INFO

```

```

CHARACTER*6      SRNAME
! .. Scalars in Common ..
INTEGER          INFOT, NOUT
LOGICAL          LERR, OK
CHARACTER*6      SRNAMT
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUT, OK, LERR
COMMON           /SRNAMC/SRNAMT
! .. Executable Statements ..
LERR = .TRUE.
IF( INFO.NE.INFOT )THEN
  IF( INFOT.NE.0 )THEN
    WRITE( NOUT, FMT = 9999 )INFO, INFOT
  ELSE
    WRITE( NOUT, FMT = 9997 )INFO
  END IF
  OK = .FALSE.
END IF
IF( SRNAME.NE.SRNAMT )THEN
  WRITE( NOUT, FMT = 9998 )SRNAME, SRNAMT
  OK = .FALSE.
END IF
RETURN
!
9999 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, ' INSTEAD', &
' OF ', I2, ' *****' )
9998 FORMAT( ' ***** XERBLA WAS CALLED WITH SRNAME = ', A6, ' INSTE', &
'AD OF ', A6, ' *****' )
9997 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, &
' *****' )
END SUBROUTINE
!
!       End of XERBLA
!
!> \brief \b DGEMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER INCX,INCY,LDA,M,N
! CHARACTER TRANS
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DGEMV performs one of the matrix-vector operations
!>
!>   y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are vectors and A is an
!> m by n matrix.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!> TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
!>
!> TRANS = 'T' or 't'   y := alpha*A**T*x + beta*y.
!>
!> TRANS = 'C' or 'c'   y := alpha*A**T*x + beta*y.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix A.
!> M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix A.
!> N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION.
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!> A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!> Before entry, the leading m by n part of the array A must
!> contain the matrix of coefficients.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim

```

```

!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      max( 1, m ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n'
!>      and at least
!>      ( 1 + ( m - 1 ) * abs( INCX ) ) otherwise.
!>      Before entry, the incremented array X must contain the
!>      vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is DOUBLE PRECISION.
!>      On entry, BETA specifies the scalar beta. When BETA is
!>      supplied as zero then Y need not be set on input.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>      Y is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n'
!>      and at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ) otherwise.
!>      Before entry with BETA non-zero, the incremented array Y
!>      must contain the vector y. On exit, Y is overwritten by the
!>      updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!
!  Authors:
!  =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
!> =====
!> SUBROUTINE DGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER INCX,INCY,LDA,M,N
! CHARACTER TRANS
!
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
! =====
!
! .. Parameters ..
! DOUBLE PRECISION ONE,ZERO
! PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! .. Local Scalars ..
! DOUBLE PRECISION TEMP
! INTEGER I,INFO,IX,IY,J,JX,JY,KX,KY,LENX,LENY
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
!
! .. External Subroutines ..
! EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
! INTRINSIC MAX
! ..
!
! Test the input parameters.

```

```

!
INFO = 0
IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.&
.NOT.LSAME(TRANS,'C')) THEN
    INFO = 1
ELSE IF (M.LT.0) THEN
    INFO = 2
ELSE IF (N.LT.0) THEN
    INFO = 3
ELSE IF (LDA.LT.MAX(1,M)) THEN
    INFO = 6
ELSE IF (INCX.EQ.0) THEN
    INFO = 8
ELSE IF (INCY.EQ.0) THEN
    INFO = 11
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('DGEMV ',INFO)
    RETURN
END IF
!
! Quick return if possible.
!
IF ((M.EQ.0) .OR. (N.EQ.0) .OR.&
((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!
! Set LENX and LENY, the lengths of the vectors x and y, and set
! up the start points in X and Y.
!
IF (LSAME(TRANS,'N')) THEN
    LENX = N
    LENY = M
ELSE
    LENX = M
    LENY = N
END IF
IF (INCX.GT.0) THEN
    KX = 1
ELSE
    KX = 1 - (LENX-1)*INCX
END IF
IF (INCY.GT.0) THEN
    KY = 1
ELSE
    KY = 1 - (LENY-1)*INCY
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
! First form y := beta*y.
!
IF (BETA.NE.ONE) THEN
    IF (INCY.EQ.1) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 10 I = 1,LENY
                Y(I) = ZERO
            CONTINUE
10        ELSE
            DO 20 I = 1,LENY
                Y(I) = BETA*Y(I)
            CONTINUE
20        END IF
        ELSE
            IY = KY
            IF (BETA.EQ.ZERO) THEN
                DO 30 I = 1,LENY
                    Y(IY) = ZERO
                    IY = IY + INCY
                CONTINUE
30            ELSE
                DO 40 I = 1,LENY
                    Y(IY) = BETA*Y(IY)
                    IY = IY + INCY
                CONTINUE
40            END IF
            END IF
        END IF
    IF (ALPHA.EQ.ZERO) RETURN
    IF (LSAME(TRANS,'N')) THEN
        Form y := alpha*A*x + y.
        JX = KX
        IF (INCY.EQ.1) THEN
            DO 60 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = ALPHA*X(JX)
                    DO 50 I = 1,M
                        Y(I) = Y(I) + TEMP*A(I,J)
                    CONTINUE
50                END IF
                JX = JX + INCX
            CONTINUE
60        ELSE
            DO 80 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = ALPHA*X(JX)
                    IY = KY
                    DO 70 I = 1,M
                        Y(IY) = Y(IY) + TEMP*A(I,J)
                        IY = IY + INCY
                    CONTINUE
70                END IF
                JX = JX + INCX
            CONTINUE
80        END IF
        ELSE
            Form y := alpha*A**T*x + y.
            JY = KY
            IF (INCX.EQ.1) THEN
                DO 100 J = 1,N

```

```

        TEMP = ZERO
        DO 90 I = 1,M
            TEMP = TEMP + A(I,J)*X(I)
90        CONTINUE
        Y(JY) = Y(JY) + ALPHA*TEMP
        JY = JY + INCY
100       CONTINUE
    ELSE
        DO 120 J = 1,N
            TEMP = ZERO
            IX = KX
            DO 110 I = 1,M
                TEMP = TEMP + A(I,J)*X(IX)
                IX = IX + INCX
110            CONTINUE
            Y(JY) = Y(JY) + ALPHA*TEMP
            JY = JY + INCY
120        CONTINUE
        END IF
    END IF
!
!   RETURN
!
!   End of DGBMV .
!
!   END SUBROUTINE

!> \brief \b DGBMV
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!       http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!       SUBROUTINE DGBMV(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
!       .. Scalar Arguments ..
!       DOUBLE PRECISION ALPHA,BETA
!       INTEGER INCX,INCY,KL,KU,LDA,M,N
!       CHARACTER TRANS
!
!       .. Array Arguments ..
!       DOUBLE PRECISION A(LDA,*),X(*),Y(*)
!
!       ..
!
!> \par Purpose:
!   =====
!> \verbatim
!> DGBMV performs one of the matrix-vector operations
!>
!>   y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are vectors and A is an
!> m by n band matrix, with kl sub-diagonals and ku super-diagonals.
!> \endverbatim
!
!   Arguments:
!   =====
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
!>
!>   TRANS = 'T' or 't'   y := alpha*A**T*x + beta*y.
!>
!>   TRANS = 'C' or 'c'   y := alpha*A**T*x + beta*y.
!> \endverbatim
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix A.
!> M must be at least zero.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] KL
!> \verbatim
!> KL is INTEGER
!> On entry, KL specifies the number of sub-diagonals of the
!> matrix A. KL must satisfy 0 .le. KL.
!> \endverbatim
!> \param[in] KU
!> \verbatim
!> KU is INTEGER
!> On entry, KU specifies the number of super-diagonals of the
!> matrix A. KU must satisfy 0 .le. KU.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION.
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] A
!> \verbatim

```

```

!>      A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>      Before entry, the leading ( kl + ku + 1 ) by n part of the
!>      array A must contain the matrix of coefficients, supplied
!>      column by column, with the leading diagonal of the matrix in
!>      row ( ku + 1 ) of the array, the first super-diagonal
!>      starting at position 2 in row ku, the first sub-diagonal
!>      starting at position 1 in row ( ku + 2 ), and so on.
!>      Elements in the array A that do not correspond to elements
!>      in the band matrix (such as the top left ku by ku triangle)
!>      are not referenced.
!>      The following program segment will transfer a band matrix
!>      from conventional full matrix storage to band storage:
!>
!>      DO 20, J = 1, N
!>        K = KU + 1 - J
!>        DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
!>          A( K + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      ( kl + ku + 1 ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n'
!>      and at least
!>      ( 1 + ( m - 1 ) * abs( INCX ) ) otherwise.
!>      Before entry, the incremented array X must contain the
!>      vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is DOUBLE PRECISION.
!>      On entry, BETA specifies the scalar beta. When BETA is
!>      supplied as zero then Y need not be set on input.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>      Y is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n'
!>      and at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ) otherwise.
!>      Before entry, the incremented array Y must contain the
!>      vector y. On exit, Y is overwritten by the updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!>
!> !
!> ! Authors:
!> ! =====
!> !
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> !
!> \date November 2011
!>
!> \ingroup double_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> !
!> ! =====
!> ! SUBROUTINE DGBMV(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!> !
!> ! -- Reference BLAS level2 routine (version 3.4.0) --
!> ! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> ! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> ! November 2011
!> !
!> ! .. Scalar Arguments ..
!> ! DOUBLE PRECISION ALPHA,BETA
!> ! INTEGER INCX,INCY,KL,KU,LDA,M,N
!> ! CHARACTER TRANS
!> !
!> ! .. Array Arguments ..
!> ! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
!> !
!> ! ..

```

```

! =====
!
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! .. Local Scalars ..
DOUBLE PRECISION TEMP
INTEGER I,INFO,IX,IY,J,JX,JY,K,KUPL,KX,KY,LENX,LENY
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC MAX,MIN
!
!
! Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.&
    .NOT.LSAME(TRANS,'C')) THEN
    INFO = 1
ELSE IF (M.LT.0) THEN
    INFO = 2
ELSE IF (N.LT.0) THEN
    INFO = 3
ELSE IF (KL.LT.0) THEN
    INFO = 4
ELSE IF (KU.LT.0) THEN
    INFO = 5
ELSE IF (LDA.LT. (KL+KU+1)) THEN
    INFO = 8
ELSE IF (INCX.EQ.0) THEN
    INFO = 10
ELSE IF (INCY.EQ.0) THEN
    INFO = 13
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('DGBMV ',INFO)
    RETURN
END IF
!
! Quick return if possible.
!
IF ((M.EQ.0) .OR. (N.EQ.0) .OR.&
    ((ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN
!
! Set LENX and LENY, the lengths of the vectors x and y, and set
! up the start points in X and Y.
!
IF (LSAME(TRANS,'N')) THEN
    LENX = N
    LENY = M
ELSE
    LENX = M
    LENY = N
END IF
IF (INCX.GT.0) THEN
    KX = 1
ELSE
    KX = 1 - (LENX-1)*INCX
END IF
IF (INCY.GT.0) THEN
    KY = 1
ELSE
    KY = 1 - (LENY-1)*INCY
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the band part of A.
!
! First form y := beta*y.
!
IF (BETA.NE.ONE) THEN
    IF (INCY.EQ.1) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 10 I = 1,LENY
                Y(I) = ZERO
10            CONTINUE
        ELSE
            DO 20 I = 1,LENY
                Y(I) = BETA*Y(I)
20            CONTINUE
        END IF
    ELSE
        IY = KY
        IF (BETA.EQ.ZERO) THEN
            DO 30 I = 1,LENY
                Y(IY) = ZERO
                IY = IY + INCY
30            CONTINUE
        ELSE
            DO 40 I = 1,LENY
                Y(IY) = BETA*Y(IY)
                IY = IY + INCY
40            CONTINUE
        END IF
    END IF
END IF
IF (ALPHA.EQ.ZERO) RETURN
KUPL = KU + 1
IF (LSAME(TRANS,'N')) THEN
    Form y := alpha*A*x + y.

    JX = KX
    IF (INCY.EQ.1) THEN
        DO 60 J = 1,N
            IF (X(JX).NE.ZERO) THEN

```

```

        TEMP = ALPHA*X(JX)
        K = KUP1 - J
        DO 50 I = MAX(1,J-KU),MIN(M,J+KL)
            Y(I) = Y(I) + TEMP*A(K+I,J)
50        CONTINUE
        END IF
        JX = JX + INCX
60        CONTINUE
    ELSE
        DO 80 J = 1,N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*X(JX)
                IY = KY
                K = KUP1 - J
                DO 70 I = MAX(1,J-KU),MIN(M,J+KL)
                    Y(IY) = Y(IY) + TEMP*A(K+I,J)
                    IY = IY + INCY
70                CONTINUE
            END IF
            JX = JX + INCX
            IF (J.GT.KU) KY = KY + INCY
80        CONTINUE
        END IF
    ELSE
        Form y := alpha*A**T*x + y.
        JY = KY
        IF (INCX.EQ.1) THEN
            DO 100 J = 1,N
                TEMP = ZERO
                K = KUP1 - J
                DO 90 I = MAX(1,J-KU),MIN(M,J+KL)
                    TEMP = TEMP + A(K+I,J)*X(I)
90                CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP
                JY = JY + INCY
100            CONTINUE
        ELSE
            DO 120 J = 1,N
                TEMP = ZERO
                IX = KX
                K = KUP1 - J
                DO 110 I = MAX(1,J-KU),MIN(M,J+KL)
                    TEMP = TEMP + A(K+I,J)*X(IX)
                    IX = IX + INCX
110                CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP
                JY = JY + INCY
                IF (J.GT.KU) KX = KX + INCX
120            CONTINUE
        END IF
    END IF
!
    RETURN
!
! End of DGBMV .
!
END SUBROUTINE

!> \brief \b DSBMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DSBMV(UPLO,N,K,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER INCX,INCY,K,LDA,N
! CHARACTER UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> DSBMV performs the matrix-vector operation
!>
!>   y := alpha*A*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are n element vectors and
!> A is an n by n symmetric band matrix, with k super-diagonals.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the band matrix A is being supplied as
!> follows:
!>
!>     UPLO = 'U' or 'u'   The upper triangular part of A is
!>                          being supplied.
!>
!>     UPLO = 'L' or 'l'   The lower triangular part of A is
!>                          being supplied.
!> \endverbatim
!> \param[in] N
!> \verbatim

```

```

!>      N is INTEGER
!>      On entry, N specifies the order of the matrix A.
!>      N must be at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>      K is INTEGER
!>      On entry, K specifies the number of super-diagonals of the
!>      matrix A. K must satisfy 0 .le. K.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is DOUBLE PRECISION.
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>      A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>      Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!>      by n part of the array A must contain the upper triangular
!>      band part of the symmetric matrix, supplied column by
!>      column, with the leading diagonal of the matrix in row
!>      ( k + 1 ) of the array, the first super-diagonal starting at
!>      position 2 in row k, and so on. The top left k by k triangle
!>      of the array A is not referenced.
!>      The following program segment will transfer the upper
!>      triangular part of a symmetric band matrix from conventional
!>      full matrix storage to band storage:
!>
!>      DO 20, J = 1, N
!>        M = K + 1 - J
!>        DO 10, I = MAX( 1, J - K ), J
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!>
!>      Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!>      by n part of the array A must contain the lower triangular
!>      band part of the symmetric matrix, supplied column by
!>      column, with the leading diagonal of the matrix in row 1 of
!>      the array, the first sub-diagonal starting at position 1 in
!>      row 2, and so on. The bottom right k by k triangle of the
!>      array A is not referenced.
!>      The following program segment will transfer the lower
!>      triangular part of a symmetric band matrix from conventional
!>      full matrix storage to band storage:
!>
!>      DO 20, J = 1, N
!>        M = 1 - J
!>        DO 10, I = J, MIN( N, J + K )
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      ( k + 1 ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the
!>      vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is DOUBLE PRECISION.
!>      On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>      Y is DOUBLE PRECISION array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ).
!>      Before entry, the incremented array Y must contain the
!>      vector y. On exit, Y is overwritten by the updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup double_blas_level2
!>
!>

```



```

        ELSE
            DO 40 I = 1,N
                Y(IY) = BETA*Y(IY)
                IY = IY + INCY
40          CONTINUE
            END IF
        END IF
    END IF
    IF (ALPHA.EQ.ZERO) RETURN
    IF (LSAME(UPLO,'U')) THEN
!
!       Form y when upper triangle of A is stored.
!
        KPLUS1 = K + 1
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 60 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                L = KPLUS1 - J
                DO 50 I = MAX(1,J-K),J - 1
                    Y(I) = Y(I) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + A(L+I,J)*X(I)
50              CONTINUE
                Y(J) = Y(J) + TEMP1*A(KPLUS1,J) + ALPHA*TEMP2
60            CONTINUE
        ELSE
            JX = KX
            JY = KY
            DO 80 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                IX = KX
                IY = KY
                L = KPLUS1 - J
                DO 70 I = MAX(1,J-K),J - 1
                    Y(IY) = Y(IY) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + A(L+I,J)*X(IX)
                    IX = IX + INCX
                    IY = IY + INCY
70              CONTINUE
                Y(JY) = Y(JY) + TEMP1*A(KPLUS1,J) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
                IF (J.GT.K) THEN
                    KX = KX + INCX
                    KY = KY + INCY
                END IF
80            CONTINUE
        END IF
    ELSE
!
!       Form y when lower triangle of A is stored.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 100 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                Y(J) = Y(J) + TEMP1*A(1,J)
                L = 1 - J
                DO 90 I = J + 1,MIN(N,J+K)
                    Y(I) = Y(I) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + A(L+I,J)*X(I)
90              CONTINUE
                Y(J) = Y(J) + ALPHA*TEMP2
100           CONTINUE
        ELSE
            JX = KX
            JY = KY
            DO 120 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                Y(JY) = Y(JY) + TEMP1*A(1,J)
                L = 1 - J
                IX = JX
                IY = JY
                DO 110 I = J + 1,MIN(N,J+K)
                    IX = IX + INCX
                    IY = IY + INCY
                    Y(IY) = Y(IY) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + A(L+I,J)*X(IX)
110              CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
120           CONTINUE
            END IF
        END IF
!
        RETURN
!
!       End of DSBMV .
!
    END SUBROUTINE

```

```

!> \brief \b DSPMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DSPMV(UPLO,N,ALPHA,AP,X,INCX,BETA,Y,INCY)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER INCX,INCY,N
! CHARACTER UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION AP(*),X(*),Y(*)

```

```

!      ..
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> DSPMV performs the matrix-vector operation
!>
!>   y := alpha*A*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are n element vectors and
!> A is an n by n symmetric matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the upper or lower
!>     triangular part of the matrix A is supplied in the packed
!>     array AP as follows:
!>
!>         UPLO = 'U' or 'u'   The upper triangular part of A is
!>                             supplied in AP.
!>
!>         UPLO = 'L' or 'l'   The lower triangular part of A is
!>                             supplied in AP.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is DOUBLE PRECISION.
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] AP
!> \verbatim
!>     AP is DOUBLE PRECISION array of DIMENSION at least
!>     ( ( n*( n + 1 ) )/2 ).
!>     Before entry with UPLO = 'U' or 'u', the array AP must
!>     contain the upper triangular part of the symmetric matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>     and a( 2, 2 ) respectively, and so on.
!>     Before entry with UPLO = 'L' or 'l', the array AP must
!>     contain the lower triangular part of the symmetric matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>     and a( 3, 1 ) respectively, and so on.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>     X is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is DOUBLE PRECISION.
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then Y need not be set on input.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>     Y is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y. On exit, Y is overwritten by the updated
!>     vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
! =====

```

```

!>
!> \verbatimim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatimim
!>
!> =====
SUBROUTINE DSPMV (UPLO,N,ALPHA,AP,X, INCX,BETA,Y, INCY)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
DOUBLE PRECISION ALPHA,BETA
INTEGER INCX,INCY,N
CHARACTER UPLO
!>
!> .. Array Arguments ..
DOUBLE PRECISION AP(*),X(*),Y(*)
!>
!> ..
!> =====
!>
!> .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!>
!> ..
!> .. Local Scalars ..
DOUBLE PRECISION TEMP1,TEMP2
INTEGER I,INFO,IX,IY,J,JX,JY,K,KK,KX,KY
!>
!> ..
!> .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!>
!> ..
!> .. External Subroutines ..
EXTERNAL XERBLA
!>
!> ..
!>
!> Test the input parameters.
!>
!> INFO = 0
!> IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!>   INFO = 1
!> ELSE IF (N.LT.0) THEN
!>   INFO = 2
!> ELSE IF (INCX.EQ.0) THEN
!>   INFO = 6
!> ELSE IF (INCY.EQ.0) THEN
!>   INFO = 9
!> END IF
!> IF (INFO.NE.0) THEN
!>   CALL XERBLA('DSPMV ',INFO)
!>   RETURN
!> END IF
!>
!> Quick return if possible.
!>
!> IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!>
!> Set up the start points in X and Y.
!>
!> IF (INCX.GT.0) THEN
!>   KX = 1
!> ELSE
!>   KX = 1 - (N-1)*INCX
!> END IF
!> IF (INCY.GT.0) THEN
!>   KY = 1
!> ELSE
!>   KY = 1 - (N-1)*INCY
!> END IF
!>
!> Start the operations. In this version the elements of the array AP
!> are accessed sequentially with one pass through AP.
!>
!> First form y := beta*y.
!>
!> IF (BETA.NE.ONE) THEN
!>   IF (INCY.EQ.1) THEN
!>     IF (BETA.EQ.ZERO) THEN
!>       DO 10 I = 1,N
!>         Y(I) = ZERO
10      CONTINUE
!>     ELSE
!>       DO 20 I = 1,N
!>         Y(I) = BETA*Y(I)
20      CONTINUE
!>     END IF
!>   ELSE
!>     IY = KY
!>     IF (BETA.EQ.ZERO) THEN
!>       DO 30 I = 1,N
!>         Y(IY) = ZERO
!>         IY = IY + INCY
30      CONTINUE
!>     ELSE
!>       DO 40 I = 1,N
!>         Y(IY) = BETA*Y(IY)
!>         IY = IY + INCY
40      CONTINUE
!>     END IF
!>   END IF
!> END IF
!> IF (ALPHA.EQ.ZERO) RETURN

```

```

      KK = 1
      IF (LSAME(UPLO,'U')) THEN
!
!       Form y when AP contains the upper triangle.
!
      IF ((INCX.EQ.1).AND. (INCY.EQ.1)) THEN
        DO 60 J = 1,N
          TEMP1 = ALPHA*X(J)
          TEMP2 = ZERO
          K = KK
          DO 50 I = 1,J - 1
            Y(I) = Y(I) + TEMP1*AP(K)
            TEMP2 = TEMP2 + AP(K)*X(I)
            K = K + 1
50          CONTINUE
          Y(J) = Y(J) + TEMP1*AP(KK+J-1) + ALPHA*TEMP2
          KK = KK + J
60          CONTINUE
        ELSE
          JX = KX
          JY = KY
          DO 80 J = 1,N
            TEMP1 = ALPHA*X(JX)
            TEMP2 = ZERO
            IX = KX
            IY = KY
            DO 70 K = KK, KK + J - 2
              Y(IY) = Y(IY) + TEMP1*AP(K)
              TEMP2 = TEMP2 + AP(K)*X(IX)
              IX = IX + INCX
              IY = IY + INCY
70            CONTINUE
            Y(JY) = Y(JY) + TEMP1*AP(KK+J-1) + ALPHA*TEMP2
            JX = JX + INCX
            JY = JY + INCY
            KK = KK + J
80            CONTINUE
          END IF
        ELSE
!
!       Form y when AP contains the lower triangle.
!
      IF ((INCX.EQ.1).AND. (INCY.EQ.1)) THEN
        DO 100 J = 1,N
          TEMP1 = ALPHA*X(J)
          TEMP2 = ZERO
          Y(J) = Y(J) + TEMP1*AP(KK)
          K = KK + 1
          DO 90 I = J + 1,N
            Y(I) = Y(I) + TEMP1*AP(K)
            TEMP2 = TEMP2 + AP(K)*X(I)
            K = K + 1
90          CONTINUE
          Y(J) = Y(J) + ALPHA*TEMP2
          KK = KK + (N-J+1)
100         CONTINUE
        ELSE
          JX = KX
          JY = KY
          DO 120 J = 1,N
            TEMP1 = ALPHA*X(JX)
            TEMP2 = ZERO
            Y(JY) = Y(JY) + TEMP1*AP(KK)
            IX = JX
            IY = JY
            DO 110 K = KK + 1, KK + N - J
              IX = IX + INCX
              IY = IY + INCY
              Y(IY) = Y(IY) + TEMP1*AP(K)
              TEMP2 = TEMP2 + AP(K)*X(IX)
110            CONTINUE
            Y(JY) = Y(JY) + ALPHA*TEMP2
            JX = JX + INCX
            JY = JY + INCY
            KK = KK + (N-J+1)
120            CONTINUE
          END IF
        END IF
      RETURN
!
!       End of DSPMV .
!
      END SUBROUTINE
!> \brief \b DTRMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DTRMV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,LDA,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*)
! ..
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!> DTRMV performs one of the matrix-vector operations
!>
!>   x := A*x,   or   x := A'*T*x,
!>
!>

```

```

!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular matrix.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the operation to be performed as
!>     follows:
!>
!>         TRANS = 'N' or 'n'   x := A*x.
!>
!>         TRANS = 'T' or 't'   x := A**T*x.
!>
!>         TRANS = 'C' or 'c'   x := A**T*x.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!> \param[in] A
!> \verbatim
!>     A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular matrix and the strictly lower triangular part of
!>     A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular matrix and the strictly upper triangular part of
!>     A is not referenced.
!>     Note that when DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced either, but are assumed to be unity.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!> \param[in,out] X
!> \verbatim
!>     X is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x. On exit, X is overwritten with the
!>     transformed vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>    Jack Dongarra, Argonne National Lab.
!>    Jeremy Du Croz, Nag Central Office.
!>    Sven Hammarling, Nag Central Office.
!>    Richard Hanson, Sandia National Labs.

```

```

!> \endverbatim
!>
! =====
      SUBROUTINE DTRMV (UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
      INTEGER INCX,LDA,N
      CHARACTER DIAG,TRANS,UPLO
!
! ..
! .. Array Arguments ..
      DOUBLE PRECISION A (LDA,*),X(*)
!
! ..
! =====
!
! .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO=0.0D+0)
!
! .. Local Scalars ..
      DOUBLE PRECISION TEMP
      INTEGER I,INFO,IX,J,JX,KX
      LOGICAL NOUNIT
!
! ..
! .. External Functions ..
      LOGICAL LSAME
!
! .. External Subroutines ..
      EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
      INTRINSIC MAX
!
! ..
!
      Test the input parameters.
!
      INFO = 0
      IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
        INFO = 1
      ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.
        .NOT.LSAME(TRANS,'C')) THEN
        INFO = 2
      ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
        INFO = 3
      ELSE IF (N.LT.0) THEN
        INFO = 4
      ELSE IF (LDA.LT.MAX(1,N)) THEN
        INFO = 6
      ELSE IF (INCX.EQ.0) THEN
        INFO = 8
      END IF
      IF (INFO.NE.0) THEN
        CALL XERBLA('DTRMV ',INFO)
        RETURN
      END IF
!
      Quick return if possible.
!
      IF (N.EQ.0) RETURN
!
      NOUNIT = LSAME(DIAG,'N')
!
      Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
      IF (INCX.LE.0) THEN
        KX = 1 - (N-1)*INCX
      ELSE IF (INCX.NE.1) THEN
        KX = 1
      END IF
!
      Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
      IF (LSAME(TRANS,'N')) THEN
!
! Form x := A*x.
!
        IF (LSAME(UPLO,'U')) THEN
          IF (INCX.EQ.1) THEN
            DO 20 J = 1,N
              IF (X(J).NE.ZERO) THEN
                TEMP = X(J)
                DO 10 I = 1,J - 1
                  X(I) = X(I) + TEMP*A(I,J)
                CONTINUE
                IF (NOUNIT) X(J) = X(J) *A(J,J)
              END IF
            CONTINUE
          ELSE
            JX = KX
            DO 40 J = 1,N
              IF (X(JX).NE.ZERO) THEN
                TEMP = X(JX)
                IX = KX
                DO 30 I = 1,J - 1
                  X(IX) = X(IX) + TEMP*A(I,J)
                  IX = IX + INCX
                CONTINUE
                IF (NOUNIT) X(JX) = X(JX) *A(J,J)
              END IF
              JX = JX + INCX
            CONTINUE
          END IF
        ELSE
          IF (INCX.EQ.1) THEN
            DO 60 J = N,1,-1
              IF (X(J).NE.ZERO) THEN

```

```

        TEMP = X(J)
        DO 50 I = N,J + 1,-1
            X(I) = X(I) + TEMP*A(I,J)
50          CONTINUE
        IF (NOUNIT) X(J) = X(J)*A(J,J)
        END IF
60        CONTINUE
    ELSE
        KX = KX + (N-1)*INCX
        JX = KX
        DO 80 J = N,1,-1
            IF (X(JX).NE.ZERO) THEN
                TEMP = X(JX)
                IX = KX
                DO 70 I = N,J + 1,-1
                    X(IX) = X(IX) + TEMP*A(I,J)
                    IX = IX - INCX
70                CONTINUE
                IF (NOUNIT) X(JX) = X(JX)*A(J,J)
                END IF
                JX = JX - INCX
80            CONTINUE
        END IF
    END IF
ELSE
    Form x := A**T*x.
    IF (LSAME(UPLO,'U')) THEN
        IF (INCX.EQ.1) THEN
            DO 100 J = N,1,-1
                TEMP = X(J)
                IF (NOUNIT) TEMP = TEMP*A(J,J)
                DO 90 I = J - 1,1,-1
                    TEMP = TEMP + A(I,J)*X(I)
90                CONTINUE
                X(J) = TEMP
100            CONTINUE
        ELSE
            JX = KX + (N-1)*INCX
            DO 120 J = N,1,-1
                TEMP = X(JX)
                IX = JX
                IF (NOUNIT) TEMP = TEMP*A(J,J)
                DO 110 I = J - 1,1,-1
                    IX = IX - INCX
                    TEMP = TEMP + A(I,J)*X(IX)
110                CONTINUE
                X(JX) = TEMP
                JX = JX - INCX
120            CONTINUE
        END IF
    ELSE
        IF (INCX.EQ.1) THEN
            DO 140 J = 1,N
                TEMP = X(J)
                IF (NOUNIT) TEMP = TEMP*A(J,J)
                DO 130 I = J + 1,N
                    TEMP = TEMP + A(I,J)*X(I)
130                CONTINUE
                X(J) = TEMP
140            CONTINUE
        ELSE
            JX = KX
            DO 160 J = 1,N
                TEMP = X(JX)
                IX = JX
                IF (NOUNIT) TEMP = TEMP*A(J,J)
                DO 150 I = J + 1,N
                    IX = IX + INCX
                    TEMP = TEMP + A(I,J)*X(IX)
150                CONTINUE
                X(JX) = TEMP
                JX = JX + INCX
160            CONTINUE
        END IF
    END IF
END IF
RETURN
!
!      End of DTRMV .
!
!      END SUBROUTINE
!> \brief \b DTBMV
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE DTBMV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
!      .. Scalar Arguments ..
!      INTEGER INCX,K,LDA,N
!      CHARACTER DIAG,TRANS,UPLO
!      ..
!      .. Array Arguments ..
!      DOUBLE PRECISION A(LDA,*),X(*)
!      ..
!
!> \par Purpose:
!      =====
!>
!> \verbatim
!>
!> DTBMV performs one of the matrix-vector operations
!>
!>      x := A*x,   or   x := A**T*x,
!>

```

```

!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular band matrix, with ( k + 1 ) diagonals.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!> TRANS = 'N' or 'n' x := A*x.
!>
!> TRANS = 'T' or 't' x := A**T*x.
!>
!> TRANS = 'C' or 'c' x := A**T*x.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] K
!> \verbatim
!> K is INTEGER
!> On entry with UPLO = 'U' or 'u', K specifies the number of
!> super-diagonals of the matrix A.
!> On entry with UPLO = 'L' or 'l', K specifies the number of
!> sub-diagonals of the matrix A.
!> K must satisfy 0 .le. K.
!> \endverbatim
!> \param[in] A
!> \verbatim
!> A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!> Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!> by n part of the array A must contain the upper triangular
!> band part of the matrix of coefficients, supplied column by
!> column, with the leading diagonal of the matrix in row
!> ( k + 1 ) of the array, the first super-diagonal starting at
!> position 2 in row k, and so on. The top left k by k triangle
!> of the array A is not referenced.
!> The following program segment will transfer an upper
!> triangular band matrix from conventional full matrix storage
!> to band storage:
!>
!> DO 20, J = 1, N
!> M = K + 1 - J
!> DO 10, I = MAX( 1, J - K ), J
!> A( M + I, J ) = matrix( I, J )
!> 10 CONTINUE
!> 20 CONTINUE
!>
!> Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!> by n part of the array A must contain the lower triangular
!> band part of the matrix of coefficients, supplied column by
!> column, with the leading diagonal of the matrix in row 1 of
!> the array, the first sub-diagonal starting at position 1 in
!> row 2, and so on. The bottom right k by k triangle of the
!> array A is not referenced.
!> The following program segment will transfer a lower
!> triangular band matrix from conventional full matrix storage
!> to band storage:
!>
!> DO 20, J = 1, N
!> M = 1 - J
!> DO 10, I = J, MIN( N, J + K )
!> A( M + I, J ) = matrix( I, J )
!> 10 CONTINUE
!> 20 CONTINUE
!>
!> Note that when DIAG = 'U' or 'u' the elements of the array A
!> corresponding to the diagonal elements of the matrix are not
!> referenced, but are assumed to be unity.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. LDA must be at least
!> ( k + 1 ).
!> \endverbatim
!> \param[in,out] X
!> \verbatim

```

```

!>      X is DOUBLE PRECISION array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the n
!>      element vector x. On exit, X is overwritten with the
!>      transformed vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!
! =====
!> SUBROUTINE DTBMV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
!> INTEGER INCX,K,LDA,N
!> CHARACTER DIAG,TRANS,UPLO
!
! .. Array Arguments ..
!> DOUBLE PRECISION A(LDA,*),X(*)
!
! ..
!
! =====
!
! .. Parameters ..
!> DOUBLE PRECISION ZERO
!> PARAMETER (ZERO=0.0D+0)
!
! ..
! .. Local Scalars ..
!> DOUBLE PRECISION TEMP
!> INTEGER I,INFO,IX,J,JX,KPLUS1,KX,L
!> LOGICAL NOUNIT
!
! ..
! .. External Functions ..
!> LOGICAL LSAME
!> EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
!> EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
!> INTRINSIC MAX,MIN
!
! ..
!
! Test the input parameters.
!
!> INFO = 0
!> IF (.NOT.LSAME(UPLO,'U')) .AND. .NOT.LSAME(UPLO,'L')) THEN
!>   INFO = 1
!> ELSE IF (.NOT.LSAME(TRANS,'N')) .AND. .NOT.LSAME(TRANS,'T') .AND. &
!>   .NOT.LSAME(TRANS,'C')) THEN
!>   INFO = 2
!> ELSE IF (.NOT.LSAME(DIAG,'U')) .AND. .NOT.LSAME(DIAG,'N')) THEN
!>   INFO = 3
!> ELSE IF (N.LT.0) THEN
!>   INFO = 4
!> ELSE IF (K.LT.0) THEN
!>   INFO = 5
!> ELSE IF (LDA.LT. (K+1)) THEN
!>   INFO = 7
!> ELSE IF (INCX.EQ.0) THEN
!>   INFO = 9
!> END IF
!> IF (INFO.NE.0) THEN
!>   CALL XERBLA('DTBMV ',INFO)
!>   RETURN
!> END IF
!
!> Quick return if possible.
!
!> IF (N.EQ.0) RETURN
!
!> NOUNIT = LSAME(DIAG,'N')
!
!> Set up the start point in X if the increment is not unity. This
!> will be ( N - 1 ) * INCX too small for descending loops.
!
!> IF (INCX.LE.0) THEN
!>   KX = 1 - (N-1)*INCX

```

```

ELSE IF (INCX.NE.1) THEN
    KX = 1
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
IF (LSAME(TRANS,'N')) THEN
    Form x := A*x.
    IF (LSAME(UPLO,'U')) THEN
        KPLUS1 = K + 1
        IF (INCX.EQ.1) THEN
            DO 20 J = 1,N
                IF (X(J).NE.ZERO) THEN
                    TEMP = X(J)
                    L = KPLUS1 - J
                    DO 10 I = MAX(1,J-K),J - 1
                        X(I) = X(I) + TEMP*A(L+I,J)
10                     CONTINUE
                    IF (NOUNIT) X(J) = X(J)*A(KPLUS1,J)
                END IF
            CONTINUE
20         ELSE
            JX = KX
            DO 40 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = X(JX)
                    IX = KX
                    L = KPLUS1 - J
                    DO 30 I = MAX(1,J-K),J - 1
                        X(IX) = X(IX) + TEMP*A(L+I,J)
30                     CONTINUE
                    IF (NOUNIT) X(JX) = X(JX)*A(KPLUS1,J)
                END IF
                JX = JX + INCX
                IF (J.GT.K) KX = KX + INCX
            CONTINUE
40         END IF
        ELSE
            IF (INCX.EQ.1) THEN
                DO 60 J = N,1,-1
                    IF (X(J).NE.ZERO) THEN
                        TEMP = X(J)
                        L = 1 - J
                        DO 50 I = MIN(N,J+K),J + 1,-1
                            X(I) = X(I) + TEMP*A(L+I,J)
50                         CONTINUE
                        IF (NOUNIT) X(J) = X(J)*A(1,J)
                    END IF
                CONTINUE
60             ELSE
                KX = KX + (N-1)*INCX
                JX = KX
                DO 80 J = N,1,-1
                    IF (X(JX).NE.ZERO) THEN
                        TEMP = X(JX)
                        IX = KX
                        L = 1 - J
                        DO 70 I = MIN(N,J+K),J + 1,-1
                            X(IX) = X(IX) + TEMP*A(L+I,J)
70                         CONTINUE
                        IF (NOUNIT) X(JX) = X(JX)*A(1,J)
                    END IF
                    JX = JX - INCX
                    IF ((N-J).GE.K) KX = KX - INCX
                CONTINUE
80             END IF
            END IF
        ELSE
            Form x := A**T*x.
            IF (LSAME(UPLO,'U')) THEN
                KPLUS1 = K + 1
                IF (INCX.EQ.1) THEN
                    DO 100 J = N,1,-1
                        TEMP = X(J)
                        L = KPLUS1 - J
                        IF (NOUNIT) TEMP = TEMP*A(KPLUS1,J)
                        DO 90 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + A(L+I,J)*X(I)
90                         CONTINUE
                        X(J) = TEMP
                    CONTINUE
100                ELSE
                    KX = KX + (N-1)*INCX
                    JX = KX
                    DO 120 J = N,1,-1
                        TEMP = X(JX)
                        KX = KX - INCX
                        IX = KX
                        L = KPLUS1 - J
                        IF (NOUNIT) TEMP = TEMP*A(KPLUS1,J)
                        DO 110 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + A(L+I,J)*X(IX)
110                         CONTINUE
                        X(JX) = TEMP
                        JX = JX - INCX
                    CONTINUE
120                END IF
                ELSE
                    IF (INCX.EQ.1) THEN
                        DO 140 J = 1,N
                            TEMP = X(J)
                            L = 1 - J
                            IF (NOUNIT) TEMP = TEMP*A(1,J)
                            DO 130 I = J + 1,MIN(N,J+K)
                                TEMP = TEMP + A(L+I,J)*X(I)
130                             CONTINUE

```

```

      X(J) = TEMP
140      CONTINUE
      ELSE
      JX = KX
      DO 160 J = 1,N
      TEMP = X(JX)
      KX = KX + INCX
      IX = KX
      L = 1 - J
      IF (NOUNIT) TEMP = TEMP*A(1,J)
      DO 150 I = J + 1,MIN(N,J+K)
      TEMP = TEMP + A(L+I,J)*X(IX)
      IX = IX + INCX
150      CONTINUE
      X(JX) = TEMP
      JX = JX + INCX
160      CONTINUE
      END IF
      END IF
      END IF
      RETURN
      End of DTPMV .
      END SUBROUTINE

!> \brief \b DTPMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DTPMV (UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION AP(*),X(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> DTPMV performs one of the matrix-vector operations
!>
!>   x := A*x,   or   x := A**T*x,
!>
!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!>   UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>   UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n'   x := A*x.
!>
!>   TRANS = 'T' or 't'   x := A**T*x.
!>
!>   TRANS = 'C' or 'c'   x := A**T*x.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!>   DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>   DIAG = 'N' or 'n'   A is not assumed to be unit
!>                       triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] AP
!> \verbatim
!> AP is DOUBLE PRECISION array of DIMENSION at least
!>   ( ( n*( n + 1 ) )/2 ).
!> Before entry with UPLO = 'U' or 'u', the array AP must
!> contain the upper triangular matrix packed sequentially,
!> column by column, so that AP( 1 ) contains a( 1, 1 ),

```

```

!>      AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
!>      respectively, and so on.
!>      Before entry with UPLO = 'L' or 'l', the array AP must
!>      contain the lower triangular matrix packed sequentially,
!>      column by column, so that AP( 1 ) contains a( 1, 1 ),
!>      AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 )
!>      respectively, and so on.
!>      Note that when DIAG = 'U' or 'u', the diagonal elements of
!>      A are not referenced, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>      X is DOUBLE PRECISION array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the n
!>      element vector x. On exit, X is overwritten with the
!>      transformed vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup double_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
!> =====
!>      SUBROUTINE DTPMV( UPLO, TRANS, DIAG, N, AP, X, INCX )
!>
!>      -- Reference BLAS level2 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd. ---
!>      November 2011
!>
!>      .. Scalar Arguments ..
!>      INTEGER INCX,N
!>      CHARACTER DIAG,TRANS,UPLO
!>
!>      ..
!>      .. Array Arguments ..
!>      DOUBLE PRECISION AP(*),X(*)
!>
!>      ..
!>
!> =====
!>
!>      .. Parameters ..
!>      DOUBLE PRECISION ZERO
!>      PARAMETER ( ZERO=0.0D+0 )
!>
!>      ..
!>      .. Local Scalars ..
!>      DOUBLE PRECISION TEMP
!>      INTEGER I,INFO,IX,J,JX,K,KK,KX
!>      LOGICAL NOUNIT
!>
!>      ..
!>      .. External Functions ..
!>      LOGICAL LSAME
!>
!>      ..
!>      .. External Subroutines ..
!>      EXTERNAL XERBLA
!>
!>      ..
!>
!>      Test the input parameters.
!>
!>      INFO = 0
!>      IF ( .NOT. LSAME( UPLO, 'U' ) .AND. .NOT. LSAME( UPLO, 'L' ) ) THEN
!>          INFO = 1
!>      ELSE IF ( .NOT. LSAME( TRANS, 'N' ) .AND. .NOT. LSAME( TRANS, 'T' ) .AND. &
!>              .NOT. LSAME( TRANS, 'C' ) ) THEN
!>          INFO = 2
!>      ELSE IF ( .NOT. LSAME( DIAG, 'U' ) .AND. .NOT. LSAME( DIAG, 'N' ) ) THEN
!>          INFO = 3
!>      ELSE IF ( N.LT.0 ) THEN
!>          INFO = 4
!>      ELSE IF ( INCX.EQ.0 ) THEN
!>          INFO = 7
!>      END IF
!>      IF ( INFO.NE.0 ) THEN
!>          CALL XERBLA( 'DTPMV ', INFO )
!>          RETURN
!>      END IF
!>
!>      Quick return if possible.
!>
!>      IF ( N.EQ.0 ) RETURN
!>
!>      NOUNIT = LSAME( DIAG, 'N' )

```

```

!
! Set up the start point in X if the increment is not unity. This
! will be ( N - 1 ) * INCX too small for descending loops.
!
IF (INCX.LE.0) THEN
  KX = 1 - (N-1)*INCX
ELSE IF (INCX.NE.1) THEN
  KX = 1
END IF
!
! Start the operations. In this version the elements of AP are
! accessed sequentially with one pass through AP.
!
IF (LSAME(TRANS,'N')) THEN
!
  Form x:= A*x.
!
  IF (LSAME(UPLO,'U')) THEN
    KK = 1
    IF (INCX.EQ.1) THEN
      DO 20 J = 1,N
        IF (X(J).NE.ZERO) THEN
          TEMP = X(J)
          K = KK
          DO 10 I = 1,J - 1
            X(I) = X(I) + TEMP*AP(K)
            K = K + 1
10          CONTINUE
          IF (NOUNIT) X(J) = X(J)*AP(KK+J-1)
        END IF
        KK = KK + J
20      CONTINUE
    ELSE
      JX = KX
      DO 40 J = 1,N
        IF (X(JX).NE.ZERO) THEN
          TEMP = X(JX)
          IX = KX
          DO 30 K = KK, KK + J - 2
            X(IX) = X(IX) + TEMP*AP(K)
            IX = IX + INCX
30          CONTINUE
          IF (NOUNIT) X(JX) = X(JX)*AP(KK+J-1)
        END IF
        JX = JX + INCX
        KK = KK + J
40      CONTINUE
    END IF
  ELSE
    KK = (N*(N+1))/2
    IF (INCX.EQ.1) THEN
      DO 60 J = N,1,-1
        IF (X(J).NE.ZERO) THEN
          TEMP = X(J)
          K = KK
          DO 50 I = N, J + 1, -1
            X(I) = X(I) + TEMP*AP(K)
            K = K - 1
50          CONTINUE
          IF (NOUNIT) X(J) = X(J)*AP(KK-N+J)
        END IF
        KK = KK - (N-J+1)
60      CONTINUE
    ELSE
      KX = KX + (N-1)*INCX
      JX = KX
      DO 80 J = N,1,-1
        IF (X(JX).NE.ZERO) THEN
          TEMP = X(JX)
          IX = KX
          DO 70 K = KK, KK - (N - (J+1)), -1
            X(IX) = X(IX) + TEMP*AP(K)
            IX = IX - INCX
70          CONTINUE
          IF (NOUNIT) X(JX) = X(JX)*AP(KK-N+J)
        END IF
        JX = JX - INCX
        KK = KK - (N-J+1)
80      CONTINUE
    END IF
  END IF
ELSE
  Form x := A**T*x.
!
  IF (LSAME(UPLO,'U')) THEN
    KK = (N*(N+1))/2
    IF (INCX.EQ.1) THEN
      DO 100 J = N,1,-1
        TEMP = X(J)
        IF (NOUNIT) TEMP = TEMP*AP(KK)
        K = KK - 1
        DO 90 I = J - 1,1,-1
          TEMP = TEMP + AP(K)*X(I)
          K = K - 1
90        CONTINUE
        X(J) = TEMP
        KK = KK - J
100      CONTINUE
    ELSE
      JX = KX + (N-1)*INCX
      DO 120 J = N,1,-1
        TEMP = X(JX)
        IX = JX
        IF (NOUNIT) TEMP = TEMP*AP(KK)
        DO 110 K = KK - 1, KK - J + 1, -1
          IX = IX - INCX
          TEMP = TEMP + AP(K)*X(IX)
110        CONTINUE
        X(JX) = TEMP
        JX = JX - INCX
        KK = KK - J
120      CONTINUE
    END IF
  END IF

```

```

        ELSE
            KK = 1
            IF (INCX.EQ.1) THEN
                DO 140 J = 1,N
                    TEMP = X(J)
                    IF (NUNIT) TEMP = TEMP*AP(KK)
                    K = KK + 1
                    DO 130 I = J + 1,N
                        TEMP = TEMP + AP(K)*X(I)
                        K = K + 1
130                 CONTINUE
                    X(J) = TEMP
                    KK = KK + (N-J+1)
140                 CONTINUE
            ELSE
                JX = KX
                DO 160 J = 1,N
                    TEMP = X(JX)
                    IX = JX
                    IF (NUNIT) TEMP = TEMP*AP(KK)
                    DO 150 K = KK + 1, KK + N - J
                        IX = IX + INCX
                        TEMP = TEMP + AP(K)*X(IX)
150                 CONTINUE
                    X(JX) = TEMP
                    JX = JX + INCX
                    KK = KK + (N-J+1)
160                 CONTINUE
            END IF
        END IF
    END IF
    RETURN
!
!   End of DTPMV .
!
!   END SUBROUTINE
!> \brief \b DTBSV
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE DTBSV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,K,LDA,N
!   CHARACTER DIAG,TRANS,UPLO
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION A(LDA,*),X(*)
!   ..
!
!> \par Purpose:
!   =====
!>
!> \verbatim
!>
!> DTBSV solves one of the systems of equations
!>
!>   A*x = b,   or   A**T*x = b,
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular band matrix, with ( k + 1 )
!> diagonals.
!>
!> No test for singularity or near-singularity is included in this
!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
!   Arguments:
!   =====
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the equations to be solved as
!>     follows:
!>
!>         TRANS = 'N' or 'n'   A*x = b.
!>
!>         TRANS = 'T' or 't'   A**T*x = b.
!>
!>         TRANS = 'C' or 'c'   A**T*x = b.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>         triangular.
!> \endverbatim
!>

```

```

!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with UPLO = 'U' or 'u', K specifies the number of
!>     super-diagonals of the matrix A.
!>     On entry with UPLO = 'L' or 'l', K specifies the number of
!>     sub-diagonals of the matrix A.
!>     K must satisfy 0 .le. K.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!>     by n part of the array A must contain the upper triangular
!>     band part of the matrix of coefficients, supplied column by
!>     column, with the leading diagonal of the matrix in row
!>     ( k + 1 ) of the array, the first super-diagonal starting at
!>     position 2 in row k, and so on. The top left k by k triangle
!>     of the array A is not referenced.
!>     The following program segment will transfer an upper
!>     triangular band matrix from conventional full matrix storage
!>     to band storage:
!>
!>         DO 20, J = 1, N
!>           M = K + 1 - J
!>           DO 10, I = MAX( 1, J - K ), J
!>             A( M + I, J ) = matrix( I, J )
!>           10 CONTINUE
!>         20 CONTINUE
!>
!>     Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!>     by n part of the array A must contain the lower triangular
!>     band part of the matrix of coefficients, supplied column by
!>     column, with the leading diagonal of the matrix in row 1 of
!>     the array, the first sub-diagonal starting at position 1 in
!>     row 2, and so on. The bottom right k by k triangle of the
!>     array A is not referenced.
!>     The following program segment will transfer a lower
!>     triangular band matrix from conventional full matrix storage
!>     to band storage:
!>
!>         DO 20, J = 1, N
!>           M = 1 - J
!>           DO 10, I = J, MIN( N, J + K )
!>             A( M + I, J ) = matrix( I, J )
!>           10 CONTINUE
!>         20 CONTINUE
!>
!>     Note that when DIAG = 'U' or 'u' the elements of the array A
!>     corresponding to the diagonal elements of the matrix are not
!>     referenced, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     ( k + 1 ).
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>     X is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element right-hand side vector b. On exit, X is overwritten
!>     with the solution vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!
! =====
!
! SUBROUTINE DTBSV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
!

```

```

! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,K,LDA,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*)
! ..
!
! =====
!
! .. Parameters ..
! DOUBLE PRECISION ZERO
! PARAMETER (ZERO=0.0D+0)
! ..
! .. Local Scalars ..
! DOUBLE PRECISION TEMP
! INTEGER I,INFO,IX,J,JX,KPLUS1,KX,L
! LOGICAL NOUNIT
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX,MIN
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.
!   .NOT.LSAME(TRANS,'C')) THEN
!   INFO = 2
! ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
!   INFO = 3
! ELSE IF (N.LT.0) THEN
!   INFO = 4
! ELSE IF (K.LT.0) THEN
!   INFO = 5
! ELSE IF (LDA.LT. (K+1)) THEN
!   INFO = 7
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('DTBSV ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF (N.EQ.0) RETURN
!
! NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be ( N - 1 ) * INCX too small for descending loops.
!
! IF (INCX.LE.0) THEN
!   KX = 1 - (N-1)*INCX
! ELSE IF (INCX.NE.1) THEN
!   KX = 1
! END IF
!
! Start the operations. In this version the elements of A are
! accessed by sequentially with one pass through A.
!
! IF (LSAME(TRANS,'N')) THEN
!
!   Form x := inv( A ) * x.
!
!   IF (LSAME(UPLO,'U')) THEN
!     KPLUS1 = K + 1
!     IF (INCX.EQ.1) THEN
!       DO 20 J = N,1,-1
!         IF (X(J).NE.ZERO) THEN
!           L = KPLUS1 - J
!           IF (NOUNIT) X(J) = X(J)/A(KPLUS1,J)
!           TEMP = X(J)
!           DO 10 I = J - 1,MAX(1,J-K),-1
!             X(I) = X(I) - TEMP*A(L+I,J)
!           CONTINUE
!         END IF
!       CONTINUE
!     ELSE
!       KX = KX + (N-1)*INCX
!       JX = KX
!       DO 40 J = N,1,-1
!         KX = KX - INCX
!         IF (X(JX).NE.ZERO) THEN
!           IX = KX
!           L = KPLUS1 - J
!           IF (NOUNIT) X(JX) = X(JX)/A(KPLUS1,J)
!           TEMP = X(JX)
!           DO 30 I = J - 1,MAX(1,J-K),-1
!             X(IX) = X(IX) - TEMP*A(L+I,J)
!           CONTINUE
!         END IF
!       CONTINUE
!     ELSE
!       IF (INCX.EQ.1) THEN
!

```

```

DO 60 J = 1,N
  IF (X(J).NE.ZERO) THEN
    L = 1 - J
    IF (NOUNIT) X(J) = X(J)/A(1,J)
    TEMP = X(J)
    DO 50 I = J + 1,MIN(N,J+K)
      X(I) = X(I) - TEMP*A(L+I,J)
50    CONTINUE
  END IF
60  CONTINUE
ELSE
  JX = KX
  DO 80 J = 1,N
    KX = KX + INCX
    IF (X(JX).NE.ZERO) THEN
      IX = KX
      L = 1 - J
      IF (NOUNIT) X(JX) = X(JX)/A(1,J)
      TEMP = X(JX)
      DO 70 I = J + 1,MIN(N,J+K)
        X(IX) = X(IX) - TEMP*A(L+I,J)
70      CONTINUE
      IX = IX + INCX
    END IF
    JX = JX + INCX
80  CONTINUE
  END IF
END IF
ELSE
  Form x := inv( A**T)*x.
  IF (LSAME(UPLO,'U')) THEN
    KPLUS1 = K + 1
    IF (INCX.EQ.1) THEN
      DO 100 J = 1,N
        TEMP = X(J)
        L = KPLUS1 - J
        DO 90 I = MAX(1,J-K),J - 1
          TEMP = TEMP - A(L+I,J)*X(I)
90      CONTINUE
        IF (NOUNIT) TEMP = TEMP/A(KPLUS1,J)
        X(J) = TEMP
100     CONTINUE
      ELSE
        JX = KX
        DO 120 J = 1,N
          TEMP = X(JX)
          IX = KX
          L = KPLUS1 - J
          DO 110 I = MAX(1,J-K),J - 1
            TEMP = TEMP - A(L+I,J)*X(IX)
            IX = IX + INCX
110         CONTINUE
          IF (NOUNIT) TEMP = TEMP/A(KPLUS1,J)
          X(JX) = TEMP
          JX = JX + INCX
          IF (J.GT.K) KX = KX + INCX
120        CONTINUE
        END IF
      ELSE
        IF (INCX.EQ.1) THEN
          DO 140 J = N,1,-1
            TEMP = X(J)
            L = 1 - J
            DO 130 I = MIN(N,J+K),J + 1,-1
              TEMP = TEMP - A(L+I,J)*X(I)
130          CONTINUE
            IF (NOUNIT) TEMP = TEMP/A(1,J)
            X(J) = TEMP
140          CONTINUE
          ELSE
            KX = KX + (N-1)*INCX
            JX = KX
            DO 160 J = N,1,-1
              TEMP = X(JX)
              IX = KX
              L = 1 - J
              DO 150 I = MIN(N,J+K),J + 1,-1
                TEMP = TEMP - A(L+I,J)*X(IX)
                IX = IX - INCX
150              CONTINUE
              IF (NOUNIT) TEMP = TEMP/A(1,J)
              X(JX) = TEMP
              JX = JX - INCX
              IF ((N-J).GE.K) KX = KX - INCX
160            CONTINUE
          END IF
        END IF
      END IF
    END IF
  RETURN
  End of DTBSV .
!
!
!
!
!
!> \brief \b DTPSV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DTPSV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION AP(*),X(*)

```

```

!      ..
!
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> DTPSV solves one of the systems of equations
!>
!>  $A*x = b,$  or  $A**T*x = b,$ 
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular matrix, supplied in packed form.
!>
!> No test for singularity or near-singularity is included in this
!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the equations to be solved as
!> follows:
!>
!> TRANS = 'N' or 'n'  $A*x = b.$ 
!>
!> TRANS = 'T' or 't'  $A**T*x = b.$ 
!>
!> TRANS = 'C' or 'c'  $A**T*x = b.$ 
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] AP
!> \verbatim
!> AP is DOUBLE PRECISION array of DIMENSION at least
!> ( ( n*( n + 1 ) )/2 ).
!> Before entry with UPLO = 'U' or 'u', the array AP must
!> contain the upper triangular matrix packed sequentially,
!> column by column, so that AP( 1 ) contains a( 1, 1 ),
!> AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
!> respectively, and so on.
!> Before entry with UPLO = 'L' or 'l', the array AP must
!> contain the lower triangular matrix packed sequentially,
!> column by column, so that AP( 1 ) contains a( 1, 1 ),
!> AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 )
!> respectively, and so on.
!> Note that when DIAG = 'U' or 'u', the diagonal elements of
!> A are not referenced, but are assumed to be unity.
!> \endverbatim
!> \param[in,out] X
!> \verbatim
!> X is DOUBLE PRECISION array of dimension at least
!> ( 1 + ( n - 1 ) *abs( INCX ) ).
!> Before entry, the incremented array X must contain the n
!> element right-hand side vector b. On exit, X is overwritten
!> with the solution vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!>

```

```

!> \verbatimim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatimim
!>
! =====
! SUBROUTINE DTPSV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION AP(*),X(*)
! ..
! =====
!
! .. Parameters ..
! DOUBLE PRECISION ZERO
! PARAMETER (ZERO=0.0D+0)
! ..
! .. Local Scalars ..
! DOUBLE PRECISION TEMP
! INTEGER I,INFO,IX,J,JX,K,KK,KX
! LOGICAL NOUNIT
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND. &
!   .NOT.LSAME(TRANS,'C')) THEN
!   INFO = 2
! ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
!   INFO = 3
! ELSE IF (N.LT.0) THEN
!   INFO = 4
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 7
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('DTPSV ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF (N.EQ.0) RETURN
!
! NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
! IF (INCX.LE.0) THEN
!   KX = 1 - (N-1)*INCX
! ELSE IF (INCX.NE.1) THEN
!   KX = 1
! END IF
!
! Start the operations. In this version the elements of AP are
! accessed sequentially with one pass through AP.
!
! IF (LSAME(TRANS,'N')) THEN
!
!   Form x := inv( A ) * x.
!
!   IF (LSAME(UPLO,'U')) THEN
!     KK = (N*(N+1))/2
!     IF (INCX.EQ.1) THEN
!       DO 20 J = N,1,-1
!         IF (X(J).NE.ZERO) THEN
!           IF (NOUNIT) X(J) = X(J)/AP(KK)
!           TEMP = X(J)
!           K = KK - 1
!           DO 10 I = J - 1,1,-1
!             X(I) = X(I) - TEMP*AP(K)
!             K = K - 1
!           CONTINUE
!         END IF
!         KK = KK - J
!       CONTINUE
!     ELSE
!       JX = KX + (N-1)*INCX
!       DO 40 J = N,1,-1
!         IF (X(JX).NE.ZERO) THEN
!           IF (NOUNIT) X(JX) = X(JX)/AP(KK)
!           TEMP = X(JX)
!           IX = JX
!           DO 30 K = KK - 1, KK - J + 1, -1
!             IX = IX - INCX
!             X(IX) = X(IX) - TEMP*AP(K)
!           CONTINUE
!         END IF
!       CONTINUE
!     END IF
!   ELSE
!     IF (LSAME(UPLO,'L')) THEN
!       DO 50 J = 1,N
!         IF (X(J).NE.ZERO) THEN
!           IF (NOUNIT) X(J) = X(J)/AP(KK)
!           TEMP = X(J)
!           KX = KX + J
!           DO 60 I = J + 1,N
!             KX = KX + INCX
!             X(KX) = X(KX) - TEMP*AP(K)
!           CONTINUE
!         END IF
!       CONTINUE
!     ELSE
!       IF (LSAME(UPLO,'U')) THEN
!         DO 70 J = 1,N
!           IF (X(J).NE.ZERO) THEN
!             IF (NOUNIT) X(J) = X(J)/AP(KK)
!             TEMP = X(J)
!             KX = KX + J
!             DO 80 I = J + 1,N
!               KX = KX + INCX
!               X(KX) = X(KX) - TEMP*AP(K)
!             CONTINUE
!           END IF
!         CONTINUE
!       ELSE
!         DO 90 J = N,1,-1
!           IF (X(J).NE.ZERO) THEN
!             IF (NOUNIT) X(J) = X(J)/AP(KK)
!             TEMP = X(J)
!             KX = KX - J
!             DO 100 I = J - 1,1,-1
!               KX = KX - INCX
!               X(KX) = X(KX) - TEMP*AP(K)
!             CONTINUE
!           END IF
!         CONTINUE
!       END IF
!     END IF
!   END IF
!
!   IF (LSAME(TRANS,'T')) THEN
!     IF (NOUNIT) X(1) = X(1)/AP(1)
!     DO 110 J = 2,N
!       IF (X(J).NE.ZERO) THEN
!         IF (NOUNIT) X(J) = X(J)/AP(J)
!         TEMP = X(J)
!         KX = J
!         DO 120 I = 1,J-1
!           KX = KX - INCX
!           X(KX) = X(KX) - TEMP*AP(KX)
!         CONTINUE
!       END IF
!     CONTINUE
!   END IF
!
!   IF (LSAME(TRANS,'C')) THEN
!     IF (NOUNIT) X(1) = X(1)/AP(1)
!     DO 130 J = 2,N
!       IF (X(J).NE.ZERO) THEN
!         IF (NOUNIT) X(J) = X(J)/AP(J)
!         TEMP = X(J)
!         KX = J
!         DO 140 I = J,N
!           KX = KX + INCX
!           X(KX) = X(KX) - TEMP*AP(KX)
!         CONTINUE
!       END IF
!     CONTINUE
!   END IF
!
! END IF
!
! =====

```

```

        END IF
        JX = JX - INCX
        KK = KK - J
40      CONTINUE
      END IF
    ELSE
      KK = 1
      IF (INCX.EQ.1) THEN
        DO 60 J = 1,N
          IF (X(J).NE.ZERO) THEN
            IF (NOUNIT) X(J) = X(J)/AP(KK)
            TEMP = X(J)
            K = KK + 1
            DO 50 I = J + 1,N
              X(I) = X(I) - TEMP*AP(K)
              K = K + 1
50          CONTINUE
            END IF
            KK = KK + (N-J+1)
60        CONTINUE
      ELSE
        JX = KX
        DO 80 J = 1,N
          IF (X(JX).NE.ZERO) THEN
            IF (NOUNIT) X(JX) = X(JX)/AP(KK)
            TEMP = X(JX)
            IX = JX
            DO 70 K = KK + 1, KK + N - J
              IX = IX + INCX
              X(IX) = X(IX) - TEMP*AP(K)
70          CONTINUE
            END IF
            JX = JX + INCX
            KK = KK + (N-J+1)
80        CONTINUE
      END IF
    END IF
  ELSE
    !
    ! Form x := inv( A**T ) * x.
    !
    IF (LSAME(UPLO,'U')) THEN
      KK = 1
      IF (INCX.EQ.1) THEN
        DO 100 J = 1,N
          TEMP = X(J)
          K = KK
          DO 90 I = 1, J - 1
            TEMP = TEMP - AP(K)*X(I)
            K = K + 1
90        CONTINUE
          IF (NOUNIT) TEMP = TEMP/AP(KK+J-1)
          X(J) = TEMP
          KK = KK + J
100       CONTINUE
      ELSE
        JX = KX
        DO 120 J = 1,N
          TEMP = X(JX)
          IX = KX
          DO 110 K = KK, KK + J - 2
            TEMP = TEMP - AP(K)*X(IX)
            IX = IX + INCX
110        CONTINUE
          IF (NOUNIT) TEMP = TEMP/AP(KK+J-1)
          X(JX) = TEMP
          JX = JX + INCX
          KK = KK + J
120       CONTINUE
      END IF
    ELSE
      KK = (N*(N+1))/2
      IF (INCX.EQ.1) THEN
        DO 140 J = N, 1, -1
          TEMP = X(J)
          K = KK
          DO 130 I = N, J + 1, -1
            TEMP = TEMP - AP(K)*X(I)
            K = K - 1
130        CONTINUE
          IF (NOUNIT) TEMP = TEMP/AP(KK-N+J)
          X(J) = TEMP
          KK = KK - (N-J+1)
140       CONTINUE
      ELSE
        KX = KX + (N-1)*INCX
        JX = KX
        DO 160 J = N, 1, -1
          TEMP = X(JX)
          IX = KX
          DO 150 K = KK, KK - (N - (J+1)), -1
            TEMP = TEMP - AP(K)*X(IX)
            IX = IX - INCX
150        CONTINUE
          IF (NOUNIT) TEMP = TEMP/AP(KK-N+J)
          X(JX) = TEMP
          JX = JX - INCX
          KK = KK - (N-J+1)
160       CONTINUE
      END IF
    END IF
  END IF
  !
  ! RETURN
  !
  ! End of DTPSV .
  !
  END SUBROUTINE
!> \brief \b DGER
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!

```

```

!
! Definition:
! =====
!
! SUBROUTINE DGER(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX, INCY, LDA, M, N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
!
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DGER performs the rank 1 operation
!>
!> A := alpha*x*y**T + A,
!>
!> where alpha is a scalar, x is an m element vector, y is an n element
!> vector and A is an m by n matrix.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix A.
!> M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix A.
!> N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION.
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!> X is DOUBLE PRECISION array of dimension at least
!> ( 1 + ( m - 1 ) * abs( INCX ) ).
!> Before entry, the incremented array X must contain the m
!> element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!> Y is DOUBLE PRECISION array of dimension at least
!> ( 1 + ( n - 1 ) * abs( INCY ) ).
!> Before entry, the incremented array Y must contain the n
!> element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!> INCY is INTEGER
!> On entry, INCY specifies the increment for the elements of
!> Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!> A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!> Before entry, the leading m by n part of the array A must
!> contain the matrix of coefficients. On exit, A is
!> overwritten by the updated matrix.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. LDA must be at least
!> max( 1, m ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>
!>

```

```

!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
! =====
! SUBROUTINE DGER(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX,INCY,LDA,M,N
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
! =====
!
! .. Parameters ..
! DOUBLE PRECISION ZERO
! PARAMETER (ZERO=0.0D+0)
! ..
! .. Local Scalars ..
! DOUBLE PRECISION TEMP
! INTEGER I,INFO,IX,J,JY,KX
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (M.LT.0) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 5
! ELSE IF (INCY.EQ.0) THEN
!   INFO = 7
! ELSE IF (LDA.LT.MAX(1,M)) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('DGER ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((M.EQ.0) .OR. (N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
! IF (INCY.GT.0) THEN
!   JY = 1
! ELSE
!   JY = 1 - (N-1)*INCY
! END IF
! IF (INCX.EQ.1) THEN
!   DO 20 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*Y(JY)
!       DO 10 I = 1,M
!         A(I,J) = A(I,J) + X(I)*TEMP
!       CONTINUE
!     END IF
!     JY = JY + INCY
! 20 CONTINUE
! ELSE
!   IF (INCX.GT.0) THEN
!     KX = 1
!   ELSE
!     KX = 1 - (M-1)*INCX
!   END IF
!   DO 40 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*Y(JY)
!       IX = KX
!       DO 30 I = 1,M
!         A(I,J) = A(I,J) + X(IX)*TEMP
!       CONTINUE
!       IX = IX + INCX
!     END IF
!     JY = JY + INCY
! 40 CONTINUE
! END IF
!
! RETURN
!
! End of DGER .
!
! END SUBROUTINE
!> \brief \b DSYR
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
!

```

```

! Definition:
! =====
!
! SUBROUTINE DSYR(UPLO,N,ALPHA,X,INCX,A,LDA)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX,LDA,N
! CHARACTER UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> DSYR performs the symmetric rank 1 operation
!>
!> A := alpha*x*x**T + A,
!>
!> where alpha is a real scalar, x is an n element vector and A is an
!> n by n symmetric matrix.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array A is to be referenced as
!> follows:
!>
!> UPLO = 'U' or 'u' Only the upper triangular part of A
!> is to be referenced.
!>
!> UPLO = 'L' or 'l' Only the lower triangular part of A
!> is to be referenced.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION.
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] X
!> \verbatim
!> X is DOUBLE PRECISION array of dimension at least
!> ( 1 + ( n - 1 ) * abs( INCX ) ).
!> Before entry, the incremented array X must contain the n
!> element vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!> \param[in,out] A
!> \verbatim
!> A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!> Before entry with UPLO = 'U' or 'u', the leading n by n
!> upper triangular part of the array A must contain the upper
!> triangular part of the symmetric matrix and the strictly
!> lower triangular part of A is not referenced. On exit, the
!> upper triangular part of the array A is overwritten by the
!> upper triangular part of the updated matrix.
!> Before entry with UPLO = 'L' or 'l', the leading n by n
!> lower triangular part of the array A must contain the lower
!> triangular part of the symmetric matrix and the strictly
!> upper triangular part of A is not referenced. On exit, the
!> lower triangular part of the array A is overwritten by the
!> lower triangular part of the updated matrix.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. LDA must be at least
!> max( 1, n ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
! =====
!> \verbatim

```

```

!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!
=====
      SUBROUTINE DSYR (UPLO,N,ALPHA,X,INCX,A,LDA)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
! .. Scalar Arguments ..
      DOUBLE PRECISION ALPHA
      INTEGER INCX,LDA,N
      CHARACTER UPLO
!
! ..
! .. Array Arguments ..
      DOUBLE PRECISION A(LDA,*),X(*)
! ..
!
=====
!
! .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO=0.0D+0)
!
! .. Local Scalars ..
      DOUBLE PRECISION TEMP
      INTEGER I,INFO,IX,J,JX,KX
!
! .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
!
! .. External Subroutines ..
      EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
      INTRINSIC MAX
!
! ..
!
      Test the input parameters.
!
      INFO = 0
      IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
         INFO = 1
      ELSE IF (N.LT.0) THEN
         INFO = 2
      ELSE IF (INCX.EQ.0) THEN
         INFO = 5
      ELSE IF (LDA.LT.MAX(1,N)) THEN
         INFO = 7
      END IF
      IF (INFO.NE.0) THEN
         CALL XERBLA('DSYR ',INFO)
         RETURN
      END IF
!
! Quick return if possible.
!
      IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Set the start point in X if the increment is not unity.
!
      IF (INCX.LE.0) THEN
         KX = 1 - (N-1)*INCX
      ELSE IF (INCX.NE.1) THEN
         KX = 1
      END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
      IF (LSAME(UPLO,'U')) THEN
!
! Form A when A is stored in upper triangle.
!
         IF (INCX.EQ.1) THEN
            DO 20 J = 1,N
               IF (X(J).NE.ZERO) THEN
                  TEMP = ALPHA*X(J)
                  DO 10 I = 1,J
                     A(I,J) = A(I,J) + X(I)*TEMP
10                  CONTINUE
               END IF
               CONTINUE
10            CONTINUE
         ELSE
            JX = KX
            DO 40 J = 1,N
               IF (X(JX).NE.ZERO) THEN
                  TEMP = ALPHA*X(JX)
                  IX = KX
                  DO 30 I = 1,J
                     A(I,J) = A(I,J) + X(IX)*TEMP
30                  CONTINUE
                  IX = IX + INCX
               END IF
               JX = JX + INCX
            CONTINUE
40            CONTINUE
         END IF
      ELSE
!
! Form A when A is stored in lower triangle.
!
         IF (INCX.EQ.1) THEN
            DO 60 J = 1,N

```

```

        IF (X(J).NE.ZERO) THEN
            TEMP = ALPHA*X(J)
            DO 50 I = J,N
                A(I,J) = A(I,J) + X(I)*TEMP
50          CONTINUE
            END IF
60          CONTINUE
        ELSE
            JX = KX
            DO 80 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = ALPHA*X(JX)
                    IX = JX
                    DO 70 I = J,N
                        A(I,J) = A(I,J) + X(IX)*TEMP
70                  CONTINUE
                    IX = IX + INCX
                END IF
                JX = JX + INCX
80          CONTINUE
            END IF
        END IF
    END IF
    RETURN
!
!   End of DSYR .
!
!   END SUBROUTINE
!> \brief \b DSPR
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE DSPR (UPLO,N,ALPHA,X,INCX,AP)
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA
!   INTEGER INCX,N
!   CHARACTER UPLO
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION AP(*),X(*)
!   ..
!
!> \par Purpose:
!   =====
!>
!> \verbatim
!>
!> DSPR    performs the symmetric rank 1 operation
!>
!>    A := alpha*x*x**T + A,
!>
!> where alpha is a real scalar, x is an n element vector and A is an
!> n by n symmetric matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>    UPLO is CHARACTER*1
!>    On entry, UPLO specifies whether the upper or lower
!>    triangular part of the matrix A is supplied in the packed
!>    array AP as follows:
!>
!>        UPLO = 'U' or 'u'   The upper triangular part of A is
!>                             supplied in AP.
!>
!>        UPLO = 'L' or 'l'   The lower triangular part of A is
!>                             supplied in AP.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>    N is INTEGER
!>    On entry, N specifies the order of the matrix A.
!>    N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>    ALPHA is DOUBLE PRECISION.
!>    On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] X
!> \verbatim
!>    X is DOUBLE PRECISION array of dimension at least
!>    ( 1 + ( n - 1 ) * abs( INCX ) ).
!>    Before entry, the incremented array X must contain the n
!>    element vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>    INCX is INTEGER
!>    On entry, INCX specifies the increment for the elements of
!>    X. INCX must not be zero.
!> \endverbatim
!> \param[in,out] AP
!> \verbatim
!>    AP is DOUBLE PRECISION array of DIMENSION at least
!>    ( ( n*( n + 1 ) ) / 2 ).
!>    Before entry with  UPLO = 'U' or 'u', the array AP must
!>    contain the upper triangular part of the symmetric matrix

```

```

!>         packed sequentially, column by column, so that AP( 1 )
!>         contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>         and a( 2, 2 ) respectively, and so on. On exit, the array
!>         AP is overwritten by the upper triangular part of the
!>         updated matrix.
!>         Before entry with UPLO = 'L' or 'l', the array AP must
!>         contain the lower triangular part of the symmetric matrix
!>         packed sequentially, column by column, so that AP( 1 )
!>         contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>         and a( 3, 1 ) respectively, and so on. On exit, the array
!>         AP is overwritten by the lower triangular part of the
!>         updated matrix.
!> \endverbatim
!
!   Authors:
!   =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!   =====
!> \verbatim
!>
!>   Level 2 Blas routine.
!>
!>   -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!   =====
!   SUBROUTINE DSPR(UPLO,N,ALPHA,X,INCX,AP)
!
!   -- Reference BLAS level2 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!   November 2011
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA
!   INTEGER INCX,N
!   CHARACTER UPLO
!
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION AP(*),X(*)
!
!   ..
!
!   =====
!
!   .. Parameters ..
!   DOUBLE PRECISION ZERO
!   PARAMETER (ZERO=0.0D+0)
!
!   ..
!   .. Local Scalars ..
!   DOUBLE PRECISION TEMP
!   INTEGER I,INFO,IX,J,JX,K,KK,KX
!
!   ..
!   .. External Functions ..
!   LOGICAL LSAME
!   EXTERNAL LSAME
!
!   ..
!   .. External Subroutines ..
!   EXTERNAL XERBLA
!
!   ..
!
!   Test the input parameters.
!
!   INFO = 0
!   IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!     INFO = 1
!   ELSE IF (N.LT.0) THEN
!     INFO = 2
!   ELSE IF (INCX.EQ.0) THEN
!     INFO = 5
!   END IF
!   IF (INFO.NE.0) THEN
!     CALL XERBLA('DSPR ',INFO)
!     RETURN
!   END IF
!
!   Quick return if possible.
!
!   IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
!   Set the start point in X if the increment is not unity.
!
!   IF (INCX.LE.0) THEN
!     KX = 1 - (N-1)*INCX
!   ELSE IF (INCX.NE.1) THEN
!     KX = 1
!   END IF
!
!   Start the operations. In this version the elements of the array AP
!   are accessed sequentially with one pass through AP.
!
!   KK = 1
!   IF (LSAME(UPLO,'U')) THEN
!
!     Form A when upper triangle is stored in AP.
!
!     IF (INCX.EQ.1) THEN
!       DO 20 J = 1,N
!         IF (X(J).NE.ZERO) THEN
!           TEMP = ALPHA*X(J)

```

```

        K = KK
        DO 10 I = 1,J
            AP(K) = AP(K) + X(I)*TEMP
            K = K + 1
10         CONTINUE
        END IF
        KK = KK + J
20     CONTINUE
    ELSE
        JX = KX
        DO 40 J = 1,N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*X(JX)
                IX = KX
                DO 30 K = KK, KK + J - 1
                    AP(K) = AP(K) + X(IX)*TEMP
                    IX = IX + INCX
30             CONTINUE
                END IF
                JX = JX + INCX
                KK = KK + J
40         CONTINUE
            END IF
        ELSE
            Form A when lower triangle is stored in AP.
            IF (INCX.EQ.1) THEN
                DO 60 J = 1,N
                    IF (X(J).NE.ZERO) THEN
                        TEMP = ALPHA*X(J)
                        K = KK
                        DO 50 I = J,N
                            AP(K) = AP(K) + X(I)*TEMP
                            K = K + 1
50             CONTINUE
                        END IF
                        KK = KK + N - J + 1
60         CONTINUE
            ELSE
                JX = KX
                DO 80 J = 1,N
                    IF (X(JX).NE.ZERO) THEN
                        TEMP = ALPHA*X(JX)
                        IX = JX
                        DO 70 K = KK, KK + N - J
                            AP(K) = AP(K) + X(IX)*TEMP
                            IX = IX + INCX
70             CONTINUE
                        END IF
                        JX = JX + INCX
                        KK = KK + N - J + 1
80         CONTINUE
            END IF
        END IF
    !
    ! RETURN
    !
    ! End of DSPR .
    !
    END SUBROUTINE
!> \brief \b DSYR2
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DSYR2(UPLO,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX,INCY,LDA,N
! CHARACTER UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> DSYR2 performs the symmetric rank 2 operation
!>
!>   A := alpha*x*y**T + alpha*y*x**T + A,
!>
!> where alpha is a scalar, x and y are n element vectors and A is an n
!> by n symmetric matrix.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array A is to be referenced as
!> follows:
!>
!>   UPLO = 'U' or 'u' Only the upper triangular part of A
!> is to be referenced.
!>
!>   UPLO = 'L' or 'l' Only the lower triangular part of A
!> is to be referenced.
!> \endverbatim
!> \param[in] N

```

```

!> \verbatim
!>      N is INTEGER
!>      On entry, N specifies the order of the matrix A.
!>      N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is DOUBLE PRECISION.
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is DOUBLE PRECISION array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the n
!>      element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!>      Y is DOUBLE PRECISION array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ).
!>      Before entry, the incremented array Y must contain the n
!>      element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!>      A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>      Before entry with UPLO = 'U' or 'u', the leading n by n
!>      upper triangular part of the array A must contain the upper
!>      triangular part of the symmetric matrix and the strictly
!>      lower triangular part of A is not referenced. On exit, the
!>      upper triangular part of the array A is overwritten by the
!>      upper triangular part of the updated matrix.
!>      Before entry with UPLO = 'L' or 'l', the leading n by n
!>      lower triangular part of the array A must contain the lower
!>      triangular part of the symmetric matrix and the strictly
!>      upper triangular part of A is not referenced. On exit, the
!>      lower triangular part of the array A is overwritten by the
!>      lower triangular part of the updated matrix.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      max( 1, n ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup double_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>      Level 2 Blas routine.
!>
!>      -- Written on 22-October-1986.
!>      Jack Dongarra, Argonne National Lab.
!>      Jeremy Du Croz, Nag Central Office.
!>      Sven Hammarling, Nag Central Office.
!>      Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
!> =====
!>      SUBROUTINE DSYR2(UPLO,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!>
!>      -- Reference BLAS level2 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!>      November 2011
!>
!>      .. Scalar Arguments ..
!>      DOUBLE PRECISION ALPHA
!>      INTEGER INCX,INCY,LDA,N
!>      CHARACTER UPLO
!>
!>      ..
!>      .. Array Arguments ..
!>      DOUBLE PRECISION A(LDA,*),X(*),Y(*)
!>
!>      ..
!>
!>      =====
!>
!>      .. Parameters ..
!>      DOUBLE PRECISION ZERO

```

```

PARAMETER (ZERO=0.0D+0)
!
! ..
! .. Local Scalars ..
! DOUBLE PRECISION TEMP1,TEMP2
! INTEGER I,INFO,IX,IY,J,JX,JY,KX,KY
!
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX
!
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!     INFO = 1
! ELSE IF (N.LT.0) THEN
!     INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!     INFO = 5
! ELSE IF (INCY.EQ.0) THEN
!     INFO = 7
! ELSE IF (LDA.LT.MAX(1,N)) THEN
!     INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!     CALL XERBLA('DSYR2 ',INFO)
!     RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Set up the start points in X and Y if the increments are not both
! unity.
!
! IF ((INCX.NE.1) .OR. (INCY.NE.1)) THEN
!     IF (INCX.GT.0) THEN
!         KX = 1
!     ELSE
!         KX = 1 - (N-1)*INCX
!     END IF
!     IF (INCY.GT.0) THEN
!         KY = 1
!     ELSE
!         KY = 1 - (N-1)*INCY
!     END IF
!     JX = KX
!     JY = KY
! END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
! IF (LSAME(UPLO,'U')) THEN
!
!     Form A when A is stored in the upper triangle.
!
!     IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
!         DO 20 J = 1,N
!             IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
!                 TEMP1 = ALPHA*Y(J)
!                 TEMP2 = ALPHA*X(J)
!                 DO 10 I = 1,J
!                     A(I,J) = A(I,J) + X(I)*TEMP1 + Y(I)*TEMP2
!                 CONTINUE
!             END IF
!         CONTINUE
!     ELSE
!         DO 40 J = 1,N
!             IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
!                 TEMP1 = ALPHA*Y(JY)
!                 TEMP2 = ALPHA*X(JX)
!                 IX = KX
!                 IY = KY
!                 DO 30 I = 1,J
!                     A(I,J) = A(I,J) + X(IX)*TEMP1 + Y(IY)*TEMP2
!                     IX = IX + INCX
!                     IY = IY + INCY
!                 CONTINUE
!             END IF
!             JX = JX + INCX
!             JY = JY + INCY
!         CONTINUE
!     END IF
! ELSE
!
!     Form A when A is stored in the lower triangle.
!
!     IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
!         DO 60 J = 1,N
!             IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
!                 TEMP1 = ALPHA*Y(J)
!                 TEMP2 = ALPHA*X(J)
!                 DO 50 I = J,N
!                     A(I,J) = A(I,J) + X(I)*TEMP1 + Y(I)*TEMP2
!                 CONTINUE
!             END IF
!         CONTINUE
!     ELSE
!         DO 80 J = 1,N
!             IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
!                 TEMP1 = ALPHA*Y(JY)
!                 TEMP2 = ALPHA*X(JX)
!                 IX = JX
!                 IY = JY
!

```

```

                DO 70 I = J,N
                  A(I,J) = A(I,J) + X(IX)*TEMP1 + Y(IY)*TEMP2
                  IX = IX + INCX
                  IY = IY + INCY
70              CONTINUE
                END IF
                JX = JX + INCX
                JY = JY + INCY
80              CONTINUE
            END IF
        END IF
    !
    !       RETURN
    !
    !       End of DSYR2 .
    !
    END SUBROUTINE
!> \brief \b DSPR2
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!       SUBROUTINE DSPR2(UPLO,N,ALPHA,X,INCX,Y,INCY,AP)
!
!       .. Scalar Arguments ..
!       DOUBLE PRECISION ALPHA
!       INTEGER INCX,INCY,N
!       CHARACTER UPLO
!
!       ..
!       .. Array Arguments ..
!       DOUBLE PRECISION AP(*),X(*),Y(*)
!       ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DSPR2 performs the symmetric rank 2 operation
!>
!>   A := alpha*x*y**T + alpha*y*x**T + A,
!>
!> where alpha is a scalar, x and y are n element vectors and A is an
!> n by n symmetric matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!>   UPLO is CHARACTER*1
!>   On entry, UPLO specifies whether the upper or lower
!>   triangular part of the matrix A is supplied in the packed
!>   array AP as follows:
!>
!>       UPLO = 'U' or 'u'   The upper triangular part of A is
!>                           supplied in AP.
!>
!>       UPLO = 'L' or 'l'   The lower triangular part of A is
!>                           supplied in AP.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>   N is INTEGER
!>   On entry, N specifies the order of the matrix A.
!>   N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>   ALPHA is DOUBLE PRECISION.
!>   On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] X
!> \verbatim
!>   X is DOUBLE PRECISION array of dimension at least
!>   ( 1 + ( n - 1 ) *abs( INCX ) ).
!>   Before entry, the incremented array X must contain the n
!>   element vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>   INCX is INTEGER
!>   On entry, INCX specifies the increment for the elements of
!>   X. INCX must not be zero.
!> \endverbatim
!> \param[in] Y
!> \verbatim
!>   Y is DOUBLE PRECISION array of dimension at least
!>   ( 1 + ( n - 1 ) *abs( INCY ) ).
!>   Before entry, the incremented array Y must contain the n
!>   element vector y.
!> \endverbatim
!> \param[in] INCY
!> \verbatim
!>   INCY is INTEGER
!>   On entry, INCY specifies the increment for the elements of
!>   Y. INCY must not be zero.
!> \endverbatim
!> \param[in,out] AP
!> \verbatim

```

```

!>      AP is DOUBLE PRECISION array of DIMENSION at least
!>      ( ( n*( n + 1 ) )/2 ).
!>      Before entry with  UPLO = 'U' or 'u', the array AP must
!>      contain the upper triangular part of the symmetric matrix
!>      packed sequentially, column by column, so that AP( 1 )
!>      contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>      and a( 2, 2 ) respectively, and so on. On exit, the array
!>      AP is overwritten by the upper triangular part of the
!>      updated matrix.
!>      Before entry with  UPLO = 'L' or 'l', the array AP must
!>      contain the lower triangular part of the symmetric matrix
!>      packed sequentially, column by column, so that AP( 1 )
!>      contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>      and a( 3, 1 ) respectively, and so on. On exit, the array
!>      AP is overwritten by the lower triangular part of the
!>      updated matrix.
!> \endverbatim
!
!   Authors:
!   =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!   =====
!> \verbatim
!>
!>   Level 2 Blas routine.
!>
!>   -- Written on 22-October-1986.
!>      Jack Dongarra, Argonne National Lab.
!>      Jeremy Du Croz, Nag Central Office.
!>      Sven Hammarling, Nag Central Office.
!>      Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!   =====
!   SUBROUTINE DSPR2(UPLO,N,ALPHA,X,INCX,Y,INCY,AP)
!
!   -- Reference BLAS level2 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA
!   INTEGER INCX,INCY,N
!   CHARACTER UPLO
!
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION AP(*),X(*),Y(*)
!
!   ..
!
!   =====
!
!   .. Parameters ..
!   DOUBLE PRECISION ZERO
!   PARAMETER (ZERO=0.0D+0)
!
!   ..
!   .. Local Scalars ..
!   DOUBLE PRECISION TEMP1,TEMP2
!   INTEGER I,INFO,IX,IY,J,JX,JY,K,KK,KX,KY
!
!   ..
!   .. External Functions ..
!   LOGICAL LSAME
!   EXTERNAL LSAME
!
!   ..
!   .. External Subroutines ..
!   EXTERNAL XERBLA
!
!   ..
!
!   Test the input parameters.
!
!   INFO = 0
!   IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!       INFO = 1
!   ELSE IF (N.LT.0) THEN
!       INFO = 2
!   ELSE IF (INCX.EQ.0) THEN
!       INFO = 5
!   ELSE IF (INCY.EQ.0) THEN
!       INFO = 7
!   END IF
!   IF (INFO.NE.0) THEN
!       CALL XERBLA('DSPR2 ',INFO)
!       RETURN
!   END IF
!
!   Quick return if possible.
!
!   IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
!   Set up the start points in X and Y if the increments are not both
!   unity.
!
!   IF ((INCX.NE.1) .OR. (INCY.NE.1)) THEN
!       IF (INCX.GT.0) THEN
!           KX = 1
!       ELSE
!           KX = 1 - (N-1)*INCX
!       END IF
!       IF (INCY.GT.0) THEN
!           KY = 1
!       ELSE
!           KY = 1 - (N-1)*INCY
!       END IF
!   END IF

```

```

        JX = KX
        JY = KY
    END IF
!
!   Start the operations. In this version the elements of the array AP
!   are accessed sequentially with one pass through AP.
!
    KK = 1
    IF (LSAME(UPLO,'U')) THEN
!
!       Form A when upper triangle is stored in AP.
!
        IF ((INCX.EQ.1).AND.(INCY.EQ.1)) THEN
            DO 20 J = 1,N
                IF ((X(J).NE.ZERO).OR.(Y(J).NE.ZERO)) THEN
                    TEMP1 = ALPHA*Y(J)
                    TEMP2 = ALPHA*X(J)
                    K = KK
                    DO 10 I = 1,J
                        AP(K) = AP(K) + X(I)*TEMP1 + Y(I)*TEMP2
                        K = K + 1
                    10 CONTINUE
                END IF
                KK = KK + J
            20 CONTINUE
        ELSE
            DO 40 J = 1,N
                IF ((X(JX).NE.ZERO).OR.(Y(JY).NE.ZERO)) THEN
                    TEMP1 = ALPHA*Y(JY)
                    TEMP2 = ALPHA*X(JX)
                    IX = KX
                    IY = KY
                    DO 30 K = KK, KK + J - 1
                        AP(K) = AP(K) + X(IX)*TEMP1 + Y(IY)*TEMP2
                        IX = IX + INCX
                        IY = IY + INCY
                    30 CONTINUE
                END IF
                JX = JX + INCX
                JY = JY + INCY
                KK = KK + J
            40 CONTINUE
        END IF
    ELSE
!
!       Form A when lower triangle is stored in AP.
!
        IF ((INCX.EQ.1).AND.(INCY.EQ.1)) THEN
            DO 60 J = 1,N
                IF ((X(J).NE.ZERO).OR.(Y(J).NE.ZERO)) THEN
                    TEMP1 = ALPHA*Y(J)
                    TEMP2 = ALPHA*X(J)
                    K = KK
                    DO 50 I = J,N
                        AP(K) = AP(K) + X(I)*TEMP1 + Y(I)*TEMP2
                        K = K + 1
                    50 CONTINUE
                END IF
                KK = KK + N - J + 1
            60 CONTINUE
        ELSE
            DO 80 J = 1,N
                IF ((X(JX).NE.ZERO).OR.(Y(JY).NE.ZERO)) THEN
                    TEMP1 = ALPHA*Y(JY)
                    TEMP2 = ALPHA*X(JX)
                    IX = JX
                    IY = JY
                    DO 70 K = KK, KK + N - J
                        AP(K) = AP(K) + X(IX)*TEMP1 + Y(IY)*TEMP2
                        IX = IX + INCX
                        IY = IY + INCY
                    70 CONTINUE
                END IF
                JX = JX + INCX
                JY = JY + INCY
                KK = KK + N - J + 1
            80 CONTINUE
        END IF
    END IF
!
!   RETURN
!
!   End of DSPR2 .
!
END SUBROUTINE

```

```

!> \brief \b DSYMV
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE DSYMV(UPLO,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA,BETA
!   INTEGER INCX,INCY,LDA,N
!   CHARACTER UPLO
!
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION A(LDA,*),X(*),Y(*)
!   ..
!
!> \par Purpose:
!   =====
!>
!> \verbatim

```

```

!>
!> DSYMV performs the matrix-vector operation
!>
!>   y := alpha*A*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are n element vectors and
!> A is an n by n symmetric matrix.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the upper or lower
!>     triangular part of the array A is to be referenced as
!>     follows:
!>
!>         UPLO = 'U' or 'u'    Only the upper triangular part of A
!>         is to be referenced.
!>
!>         UPLO = 'L' or 'l'    Only the lower triangular part of A
!>         is to be referenced.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is DOUBLE PRECISION.
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] A
!> \verbatim
!>     A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular part of the symmetric matrix and the strictly
!>     lower triangular part of A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular part of the symmetric matrix and the strictly
!>     upper triangular part of A is not referenced.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!> \param[in] X
!> \verbatim
!>     X is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 )*abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!> \param[in] BETA
!> \verbatim
!>     BETA is DOUBLE PRECISION.
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then Y need not be set on input.
!> \endverbatim
!> \param[in,out] Y
!> \verbatim
!>     Y is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 )*abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y. On exit, Y is overwritten by the updated
!>     vector y.
!> \endverbatim
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level2
!
!> \par Further Details:
!> =====
!> \endverbatim

```

```

!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!=====
      SUBROUTINE DSYMV(UPLO,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
      DOUBLE PRECISION ALPHA,BETA
      INTEGER INCX,INCY,LDA,N
      CHARACTER UPLO
!
! ..
! .. Array Arguments ..
      DOUBLE PRECISION A(LDA,*),X(*),Y(*)
! ..
!=====
!
! .. Parameters ..
      DOUBLE PRECISION ONE,ZERO
      PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! .. Local Scalars ..
      DOUBLE PRECISION TEMP1,TEMP2
      INTEGER I,INFO,IX,IY,J,JX,JY,KX,KY
!
! ..
! .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
      EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
      INTRINSIC MAX
!
! ..
!
! Test the input parameters.
!
      INFO = 0
      IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
          INFO = 1
      ELSE IF (N.LT.0) THEN
          INFO = 2
      ELSE IF (LDA.LT.MAX(1,N)) THEN
          INFO = 5
      ELSE IF (INCX.EQ.0) THEN
          INFO = 7
      ELSE IF (INCY.EQ.0) THEN
          INFO = 10
      END IF
      IF (INFO.NE.0) THEN
          CALL XERBLA('DSYMV ',INFO)
          RETURN
      END IF
!
! Quick return if possible.
!
      IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!
! Set up the start points in X and Y.
!
      IF (INCX.GT.0) THEN
          KX = 1
      ELSE
          KX = 1 - (N-1)*INCX
      END IF
      IF (INCY.GT.0) THEN
          KY = 1
      ELSE
          KY = 1 - (N-1)*INCY
      END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
! First form y := beta*y.
!
      IF (BETA.NE.ONE) THEN
          IF (INCY.EQ.1) THEN
              IF (BETA.EQ.ZERO) THEN
                  DO 10 I = 1,N
                      Y(I) = ZERO
                  CONTINUE
10              ELSE
                  DO 20 I = 1,N
                      Y(I) = BETA*Y(I)
                  CONTINUE
20              END IF
              ELSE
                  IY = KY
                  IF (BETA.EQ.ZERO) THEN
                      DO 30 I = 1,N
                          Y(IY) = ZERO
                          IY = IY + INCY
                      CONTINUE
30                  ELSE
                      DO 40 I = 1,N
                          Y(IY) = BETA*Y(IY)
                          IY = IY + INCY
                      CONTINUE
40

```

```

        END IF
    END IF
    END IF
    IF (ALPHA.EQ.ZERO) RETURN
    IF (LSAME(UPLO,'U')) THEN
!
!       Form y when A is stored in upper triangle.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 60 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                DO 50 I = 1,J - 1
                    Y(I) = Y(I) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + A(I,J)*X(I)
20                CONTINUE
                Y(J) = Y(J) + TEMP1*A(J,J) + ALPHA*TEMP2
30            CONTINUE
40        ELSE
            JX = KX
            JY = KY
            DO 80 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                IX = KX
                IY = KY
                DO 70 I = 1,J - 1
                    Y(IY) = Y(IY) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + A(I,J)*X(IX)
                    IX = IX + INCX
                    IY = IY + INCY
50                CONTINUE
                Y(JY) = Y(JY) + TEMP1*A(J,J) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
60            CONTINUE
70        END IF
80    ELSE
!
!       Form y when A is stored in lower triangle.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 100 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                Y(J) = Y(J) + TEMP1*A(J,J)
                DO 90 I = J + 1,N
                    Y(I) = Y(I) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + A(I,J)*X(I)
90                CONTINUE
                Y(J) = Y(J) + ALPHA*TEMP2
100            CONTINUE
110        ELSE
            JX = KX
            JY = KY
            DO 120 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                Y(JY) = Y(JY) + TEMP1*A(J,J)
                IX = JX
                IY = JY
                DO 110 I = J + 1,N
                    IX = IX + INCX
                    IY = IY + INCY
                    Y(IY) = Y(IY) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + A(I,J)*X(IX)
110                CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
120            CONTINUE
130        END IF
140    END IF
!
!       RETURN
!
!       End of DSYMV .
!
    END SUBROUTINE
!
!> \brief \b DTRSV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DTRSV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,LDA,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),X(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DTRSV solves one of the systems of equations
!>
!>  $Ax = b,$  or  $A^T x = b,$ 
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular matrix.
!>
!> No test for singularity or near-singularity is included in this

```

```

!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the equations to be solved as
!>     follows:
!>
!>         TRANS = 'N' or 'n'   A*x = b.
!>
!>         TRANS = 'T' or 't'   A**T*x = b.
!>
!>         TRANS = 'C' or 'c'   A**T*x = b.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is DOUBLE PRECISION array of DIMENSION ( LDA, n ).
!>     Before entry with  UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular matrix and the strictly lower triangular part of
!>     A is not referenced.
!>     Before entry with  UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular matrix and the strictly upper triangular part of
!>     A is not referenced.
!>     Note that when  DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>     X is DOUBLE PRECISION array of dimension at least
!>     ( 1 + ( n - 1 )*abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element right-hand side vector b. On exit, X is overwritten
!>     with the solution vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
! =====
!     SUBROUTINE DTRSV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! -- Reference BLAS levell routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---

```

```

! November 2011
!
! .. Scalar Arguments ..
INTEGER INCX,LDA,N
CHARACTER DIAG,TRANS,UPLO
!
! .. Array Arguments ..
DOUBLE PRECISION A(LDA,*),X(*)
!
!
=====
!
! .. Parameters ..
DOUBLE PRECISION ZERO
PARAMETER (ZERO=0.0D+0)
!
! .. Local Scalars ..
DOUBLE PRECISION TEMP
INTEGER I,INFO,IX,J,JX,KX
LOGICAL NOUNIT
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC MAX
!
!
Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
    INFO = 1
ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.
        .NOT.LSAME(TRANS,'C')) THEN
    INFO = 2
ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
    INFO = 3
ELSE IF (N.LT.0) THEN
    INFO = 4
ELSE IF (LDA.LT.MAX(1,N)) THEN
    INFO = 6
ELSE IF (INCX.EQ.0) THEN
    INFO = 8
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('DTRSV ',INFO)
    RETURN
END IF

Quick return if possible.
!
IF (N.EQ.0) RETURN
!
NOUNIT = LSAME(DIAG,'N')
!
Set up the start point in X if the increment is not unity. This
will be (N-1)*INCX too small for descending loops.
!
IF (INCX.LE.0) THEN
    KX = 1 - (N-1)*INCX
ELSE IF (INCX.NE.1) THEN
    KX = 1
END IF
!
Start the operations. In this version the elements of A are
accessed sequentially with one pass through A.
!
IF (LSAME(TRANS,'N')) THEN
!
    Form x := inv( A )*x.
!
    IF (LSAME(UPLO,'U')) THEN
        IF (INCX.EQ.1) THEN
            DO 20 J = N,1,-1
                IF (X(J).NE.ZERO) THEN
                    IF (NOUNIT) X(J) = X(J)/A(J,J)
                    TEMP = X(J)
                    DO 10 I = J-1,1,-1
                        X(I) = X(I) - TEMP*A(I,J)
10                     CONTINUE
                END IF
20                 CONTINUE
            ELSE
                JX = KX + (N-1)*INCX
                DO 40 J = N,1,-1
                    IF (X(JX).NE.ZERO) THEN
                        IF (NOUNIT) X(JX) = X(JX)/A(J,J)
                        TEMP = X(JX)
                        IX = JX
                        DO 30 I = J-1,1,-1
                            IX = IX - INCX
                            X(IX) = X(IX) - TEMP*A(I,J)
30                         CONTINUE
                        END IF
                        JX = JX - INCX
40                     CONTINUE
                END IF
            ELSE
                IF (INCX.EQ.1) THEN
                    DO 60 J = 1,N
                        IF (X(J).NE.ZERO) THEN
                            IF (NOUNIT) X(J) = X(J)/A(J,J)
                            TEMP = X(J)
                            DO 50 I = J+1,N
                                X(I) = X(I) - TEMP*A(I,J)
50                             CONTINUE
                            END IF
                        END IF
60                     CONTINUE
                ELSE

```

```

      JX = KX
      DO 80 J = 1,N
        IF (X(JX).NE.ZERO) THEN
          IF (NOUNIT) X(JX) = X(JX)/A(J,J)
          TEMP = X(JX)
          IX = JX
          DO 70 I = J + 1,N
            IX = IX + INCX
            X(IX) = X(IX) - TEMP*A(I,J)
70          CONTINUE
          END IF
          JX = JX + INCX
80        CONTINUE
      END IF
    ELSE IF
      ELSE
!      Form x := inv( A**T ) * x.
!
!      IF (LSAME(UPLO,'U')) THEN
!        IF (INCX.EQ.1) THEN
!          DO 100 J = 1,N
!            TEMP = X(J)
!            DO 90 I = 1,J - 1
!              TEMP = TEMP - A(I,J)*X(I)
90            CONTINUE
!            IF (NOUNIT) TEMP = TEMP/A(J,J)
!            X(J) = TEMP
100          CONTINUE
!        ELSE
!          JX = KX
!          DO 120 J = 1,N
!            TEMP = X(JX)
!            IX = KX
!            DO 110 I = 1,J - 1
!              TEMP = TEMP - A(I,J)*X(IX)
110            CONTINUE
!            IF (NOUNIT) TEMP = TEMP/A(J,J)
!            X(JX) = TEMP
!            JX = JX + INCX
120          CONTINUE
!        END IF
!      ELSE
!        IF (INCX.EQ.1) THEN
!          DO 140 J = N,1,-1
!            TEMP = X(J)
!            DO 130 I = N,J + 1,-1
!              TEMP = TEMP - A(I,J)*X(I)
130            CONTINUE
!            IF (NOUNIT) TEMP = TEMP/A(J,J)
!            X(J) = TEMP
140          CONTINUE
!        ELSE
!          KX = KX + (N-1)*INCX
!          JX = KX
!          DO 160 J = N,1,-1
!            TEMP = X(JX)
!            IX = KX
!            DO 150 I = N,J + 1,-1
!              TEMP = TEMP - A(I,J)*X(IX)
150            CONTINUE
!            IF (NOUNIT) TEMP = TEMP/A(J,J)
!            X(JX) = TEMP
!            JX = JX - INCX
160          CONTINUE
!        END IF
!      END IF
!    END IF
!  RETURN
!
!  End of DTRSV .
!
!  END SUBROUTINE
end program

```

APPENDIX A.4 – DBLAT3.f90

```

program testdblat3
implicit none

! Variables
CHARACTER :: kin

DOUBLE PRECISION AA(12), BB(8), CC(8)
INTEGER LDA, LDB, LDC, I, J

CALL DBLAT3
print *, 'DBLAT3 Done.'

!https://software.intel.com/en-us/node/522288
LDA = 4
LDB = 4
LDC = 4

DO 11 I= 1, 3
  DO 12 J=1,3
    AA(((I-1)+(J-1)*LDA)+1) = 1.0
  12 CONTINUE
  11 CONTINUE

DO 13 I= 1, 3
  DO 14 J=1,2
    CC( ((I-1) + (J-1)*LDC)+1 ) = 1.0
    BB( ((I-1) + (J-1)*LDB)+1 ) = 2.0
  14 CONTINUE
  13 CONTINUE

CALL DSYMM('L', 'U', 3, 2, 0.5D0, AA, LDA, BB, LDB, 2.0D0, CC, LDC)

DO 15 I= 1, 3
  DO 16 J=1,2
    print *, ( (I-1)+(J-1)*LDC)+1, CC(((I-1)+(J-1)*LDC)+1)
  16 CONTINUE
  15 CONTINUE
read(*, '(A1)'), kin
STOP

CONTAINS
!> \brief \b LSAME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! LOGICAL FUNCTION LSAME(CA,CB)
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> LSAME returns .TRUE. if CA is the same letter as CB regardless of
!> case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] CA
!> \verbatim
!> CA is CHARACTER*1
!> \endverbatim
!>
!> \param[in] CB
!> \verbatim
!> CB is CHARACTER*1
!> CA and CB specify the single characters to be compared.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup aux_blas
!
! =====
! LOGICAL FUNCTION LSAME(CA,CB)
!
! -- Reference BLAS level1 routine (version 3.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
! =====
!
! .. Intrinsic Functions ..
! INTRINSIC ICHAR
!
! ..
! .. Local Scalars ..

```

```

      INTEGER INTA,INTB,ZCODE
      ..
      !
      ! Test if the characters are equal
      !
      LSAME = CA .EQ. CB
      IF (LSAME) RETURN
      !
      ! Now test for equivalence if both characters are alphabetic.
      !
      ZCODE = ICHAR('Z')
      !
      ! Use 'Z' rather than 'A' so that ASCII can be detected on Prime
      ! machines, on which ICHAR returns a value with bit 8 set.
      ! ICHAR('A') on Prime machines returns 193 which is the same as
      ! ICHAR('A') on an EBCDIC machine.
      !
      INTA = ICHAR(CA)
      INTB = ICHAR(CB)
      !
      IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
      !
      !   ASCII is assumed - ZCODE is the ASCII code of either lower or
      !   upper case 'Z'.
      !
      IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
      IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
      !
      ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN
      !
      !   EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or
      !   upper case 'Z'.
      !
      IF (INTA.GE.129 .AND. INTA.LE.137 .OR.&
          INTA.GE.145 .AND. INTA.LE.153 .OR.&
          INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
      IF (INTB.GE.129 .AND. INTB.LE.137 .OR.&
          INTB.GE.145 .AND. INTB.LE.153 .OR.&
          INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
      !
      ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
      !
      !   ASCII is assumed, on Prime machines - ZCODE is the ASCII code
      !   plus 128 of either lower or upper case 'Z'.
      !
      IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
      IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
      END IF
      LSAME = INTA .EQ. INTB
      !
      ! RETURN
      !
      ! End of LSAME
      !
      END FUNCTION

! \brief \b DBLAT3
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! PROGRAM DBLAT3
!
! \par Purpose:
! =====
!
! \verbatim
!
! Test program for the DOUBLE PRECISION Level 3 Blas.
!
! The program must be driven by a short data file. The first 14 records
! of the file are read using list-directed input, the last 6 records
! are read using the format ( A6, L2 ). An annotated example of a data
! file can be obtained by deleting the first 3 characters from the
! following 20 lines:
! 'dlat3.out'      NAME OF SUMMARY OUTPUT FILE
! 6               UNIT NUMBER OF SUMMARY FILE
! 'DBLAT3.SNAP'   NAME OF SNAPSHOT OUTPUT FILE
! -1             UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
! F              LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
! F              LOGICAL FLAG, T TO STOP ON FAILURES.
! T              LOGICAL FLAG, T TO TEST ERROR EXITS.
! 16.0           THRESHOLD VALUE OF TEST RATIO
! 6              NUMBER OF VALUES OF N
! 0 1 2 3 5 9    VALUES OF N
! 3              NUMBER OF VALUES OF ALPHA
! 0.0 1.0 0.7    VALUES OF ALPHA
! 3              NUMBER OF VALUES OF BETA
! 0.0 1.0 1.3    VALUES OF BETA
! DGEMM T PUT F FOR NO TEST. SAME COLUMNS.
! DSYMM T PUT F FOR NO TEST. SAME COLUMNS.
! DTRMM T PUT F FOR NO TEST. SAME COLUMNS.
! DTRSM T PUT F FOR NO TEST. SAME COLUMNS.
! DSYRK T PUT F FOR NO TEST. SAME COLUMNS.
! DSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
!
! Further Details
! =====
!
! See:
!
! Dongarra J. J., Du Croz J. J., Duff I. S. and Hammarling S.
! A Set of Level 3 Basic Linear Algebra Subprograms.
!
! Technical Memorandum No.88 (Revision 1), Mathematics and
! Computer Science Division, Argonne National Laboratory, 9700
! South Cass Avenue, Argonne, Illinois 60439, US.
!

```

```

! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! 10-9-00: Change STATUS='NEW' to 'UNKNOWN' so that the testers
!         can be run multiple times without deleting generated
!         output files (susan)
!
! \endverbatim
!
! Authors:
! =====
!
! \author Univ. of Tennessee
! \author Univ. of California Berkeley
! \author Univ. of Colorado Denver
! \author NAG Ltd.
!
! \date April 2012
!
! \ingroup double_blas_testing
!
! =====
SUBROUTINE DBLAT3
!
! -- Reference BLAS test routine (version 3.4.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! April 2012
!
! =====
!
! .. Parameters ..
INTEGER          NIN
PARAMETER        ( NIN = 5 )
INTEGER          NSUBS
PARAMETER        ( NSUBS = 6 )
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D0, ONE = 1.0D0 )
INTEGER          NMAX
PARAMETER        ( NMAX = 65 )
INTEGER          NIDMAX, NALMAX, NBEMAX
PARAMETER        ( NIDMAX = 9, NALMAX = 7, NBEMAX = 7 )
!
! .. Local Scalars ..
DOUBLE PRECISION EPS, ERR, THRESH
INTEGER          I, ISNUM, J, N, NALF, NBET, NIDIM, NOUT, NTRA
LOGICAL          FATAL, LTESTT, REWI, SAME, SFATAL, TRACE, &
                 TSERR
CHARACTER*1      TRANSA, TRANSB
CHARACTER*6      SNAME
CHARACTER*32     SNAPS, SUMMRY
!
! .. Local Arrays ..
DOUBLE PRECISION AA( NMAX*NMAX ), AB( NMAX, 2*NMAX ), &
                 ALF( NALMAX ), AS( NMAX*NMAX ), &
                 BB( NMAX*NMAX ), BET( NBEMAX ), &
                 BS( NMAX*NMAX ), C( NMAX, NMAX ), &
                 CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ), &
                 G( NMAX ), W( 2*NMAX ), &
                 T2( 2,4 )
INTEGER          IDIM( NIDMAX )
LOGICAL          LTEST( NSUBS )
CHARACTER*6      SNAME$( NSUBS )
!
! .. External Functions ..
DOUBLE PRECISION DDIFF
LOGICAL          LDE
EXTERNAL         DDIFF, LDE
!
! .. External Subroutines ..
EXTERNAL        DCHK1, DCHK2, DCHK3, DCHK4, DCHK5, DCHKE, DMMCH
!
! .. Intrinsic Functions ..
INTRINSIC       MAX, MIN
!
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
CHARACTER*6      SRNAMT
!
! .. Common blocks ..
COMMON          /INFOC/INFOT, NOUTC, OK, LERR
COMMON          /SRNAMC/SRNAMT
!
! .. Data statements ..
DATA           SNAME$/ 'DGEMM ', 'DSYMM ', 'DTRMM ', 'DTRSM ', &
                'DSYRK ', 'DSYR2K' /
!
! .. Executable Statements ..
!
! Read name and unit number for summary output file and open file.
!
READ( NIN, FMT = * ) SUMMRY
READ( NIN, FMT = * ) NOUT
OPEN( NOUT, FILE = SUMMRY, STATUS = 'UNKNOWN' )
NOUTC = NOUT
!
! Read name and unit number for snapshot output file and open file.
!
READ( NIN, FMT = * ) SNAPS
READ( NIN, FMT = * ) NTRA
TRACE = NTRA.GE.0
IF( TRACE ) THEN
    OPEN( NTRA, FILE = SNAPS, STATUS = 'UNKNOWN' )
END IF
!
! Read the flag that directs rewinding of the snapshot file.
READ( NIN, FMT = * ) REWI
REWI = REWI.AND.TRACE
!
! Read the flag that directs stopping on any failure.
READ( NIN, FMT = * ) SFATAL
!
! Read the flag that indicates whether error exits are to be tested.
READ( NIN, FMT = * ) TSERR
!
! Read the threshold value of the test ratio
READ( NIN, FMT = * ) THRESH
!
! Read and check the parameter values for the tests.
!
!
! Values of N
READ( NIN, FMT = * ) NIDIM
IF( NIDIM.LT.1.OR.NIDIM.GT.NIDMAX ) THEN
    WRITE( NOUT, FMT = 9997 ) 'N', NIDMAX

```

```

        GO TO 220
    END IF
    READ( NIN, FMT = * ) ( IDIM( I ), I = 1, NIDIM )
    DO 10 I = 1, NIDIM
        IF( IDIM( I ).LT.0.OR.IDIM( I ).GT.NMAX )THEN
            WRITE( NOUT, FMT = 9996 )NMAX
            GO TO 220
        END IF
10 CONTINUE
! Values of ALPHA
    READ( NIN, FMT = * )NALF
    IF( NALF.LT.1.OR.NALF.GT.NALMAX )THEN
        WRITE( NOUT, FMT = 9997 )'ALPHA', NALMAX
        GO TO 220
    END IF
! Values of BETA
    READ( NIN, FMT = * )NBET
    IF( NBET.LT.1.OR.NBET.GT.NBEMAX )THEN
        WRITE( NOUT, FMT = 9997 )'BETA', NBEMAX
        GO TO 220
    END IF
    READ( NIN, FMT = * ) ( BET( I ), I = 1, NBET )
!
! Report values of parameters.
!
    WRITE( NOUT, FMT = 9995 )
    WRITE( NOUT, FMT = 9994 ) ( IDIM( I ), I = 1, NIDIM )
    WRITE( NOUT, FMT = 9993 ) ( ALF( I ), I = 1, NALF )
    WRITE( NOUT, FMT = 9992 ) ( BET( I ), I = 1, NBET )
    IF( .NOT.TSTERR )THEN
        WRITE( NOUT, FMT = * )
        WRITE( NOUT, FMT = 9984 )
    END IF
    WRITE( NOUT, FMT = * )
    WRITE( NOUT, FMT = 9999 )THRESH
    WRITE( NOUT, FMT = * )
!
! Read names of subroutines and flags which indicate
! whether they are to be tested.
!
    DO 20 I = 1, NSUBS
        LTEST( I ) = .FALSE.
20 CONTINUE
30 READ( NIN, FMT = 9988, END = 60 )SNAMET, LTESTT
    DO 40 I = 1, NSUBS
        IF( SNAMET.EQ.SNAMES( I ) )&
            GO TO 50
40 CONTINUE
    WRITE( NOUT, FMT = 9990 )SNAMET
    STOP
50 LTEST( I ) = LTESTT
    GO TO 30
!
60 CONTINUE
    CLOSE ( NIN )
!
! Compute EPS (the machine precision).
!
    EPS = EPSILON(ZERO)
    WRITE( NOUT, FMT = 9998 )EPS
!
! Check the reliability of DMMCH using exact data.
!
    N = MIN( 32, NMAX )
    DO 100 J = 1, N
        DO 90 I = 1, N
            AB( I, J ) = MAX( I - J + 1, 0 )
90 CONTINUE
            AB( J, NMAX + 1 ) = J
            AB( 1, NMAX + J ) = J
            C( J, 1 ) = ZERO
100 CONTINUE
            DO 110 J = 1, N
                CC( J ) = J*( ( J + 1 ) * J ) / 2 - ( ( J + 1 ) * J * ( J - 1 ) ) / 3
110 CONTINUE
! CC holds the exact result. On exit from DMMCH CT holds
! the result computed by DMMCH.
    TRANSA = 'N'
    TRANSB = 'N'
    CALL DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
        AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
        NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
    SAME = LDE( CC, CT, N )
    IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
        WRITE( NOUT, FMT = 9989 )TRANSA, TRANSB, SAME, ERR
        STOP
    END IF
    TRANSB = 'T'
    CALL DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
        AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
        NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
    SAME = LDE( CC, CT, N )
    IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
        WRITE( NOUT, FMT = 9989 )TRANSA, TRANSB, SAME, ERR
        STOP
    END IF
    DO 120 J = 1, N
        AB( J, NMAX + 1 ) = N - J + 1
        AB( 1, NMAX + J ) = N - J + 1
120 CONTINUE
            DO 130 J = 1, N
                CC( N - J + 1 ) = J*( ( J + 1 ) * J ) / 2 - &
                    ( ( J + 1 ) * J * ( J - 1 ) ) / 3
130 CONTINUE
    TRANSA = 'T'
    TRANSB = 'N'
    CALL DMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
        AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
        NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
    SAME = LDE( CC, CT, N )
    IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
        WRITE( NOUT, FMT = 9989 )TRANSA, TRANSB, SAME, ERR
        STOP

```

```

END IF
TRANSB = 'T'
CALL DMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX,&
  AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC,&
  NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
SAME = LDE( CC, CT, N )
IF( .NOT.SAME.OR.ERR.NE.ZERO )THEN
  WRITE( NOUT, FMT = 9989 )TRANSA, TRANSB, SAME, ERR
  STOP
END IF

!
! Test each subroutine in turn.
!
DO 200 ISNUM = 1, NSUBS
  WRITE( NOUT, FMT = * )
  IF( .NOT.LTEST( ISNUM ) )THEN
    Subprogram is not to be tested.
    WRITE( NOUT, FMT = 9987 )SNAMES( ISNUM )
  ELSE
    SRNAMT = SNAMES( ISNUM )
    Test error exits.
    IF( TSERR )THEN
      CALL DCHKE( ISNUM, SNAMES( ISNUM ), NOUT )
      WRITE( NOUT, FMT = * )
    END IF
    Test computations.
    INFOT = 0
    OK = .TRUE.
    FATAL = .FALSE.
    GO TO ( 140, 150, 160, 170, 180 )ISNUM
    Test DGEMM, 01.
140    CALL DCHK1( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
      REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET,&
      NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C,&
      CC, CS, CT, G )
    GO TO 190
    Test DSYMM, 02.
150    CALL DCHK2( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
      REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET,&
      NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C,&
      CC, CS, CT, G )
    GO TO 190
    Test DTRMM, 03, DTRSM, 04.
160    CALL DCHK3( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
      REWI, FATAL, NIDIM, IDIM, NALF, ALF, NMAX, AB,&
      AA, AS, AB( 1, NMAX + 1 ), BB, BS, CT, G, C )
    GO TO 190
    Test DSYRK, 05.
170    CALL DCHK4( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
      REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET,&
      NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C,&
      CC, CS, CT, G )
    GO TO 190
    Test DSYR2K, 06.
180    CALL DCHK5( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE,&
      REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET,&
      NMAX, AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )
    GO TO 190
190    IF( FATAL.AND.SPATAL )&
      GO TO 210
    END IF
200 CONTINUE
    WRITE( NOUT, FMT = 9986 )
    GO TO 230
!
210 CONTINUE
    WRITE( NOUT, FMT = 9985 )
    GO TO 230
!
220 CONTINUE
    WRITE( NOUT, FMT = 9991 )
!
230 CONTINUE
    IF( TRACE )&
      CLOSE ( NTRA )
    CLOSE ( NOUT )
    RETURN ! replace STOP
!
9999 FORMAT( ' ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LES',&
  'S THAN', F8.2 )
9998 FORMAT( ' RELATIVE MACHINE PRECISION IS TAKEN TO BE', 1P, D9.1 )
9997 FORMAT( ' NUMBER OF VALUES OF ', A, ' IS LESS THAN 1 OR GREATER ',&
  'THAN ', I2 )
9996 FORMAT( ' VALUE OF N IS LESS THAN 0 OR GREATER THAN ', I2 )
9995 FORMAT( ' TESTS OF THE DOUBLE PRECISION LEVEL 3 BLAS', '/' THE F',&
  'OLLOWING PARAMETER VALUES WILL BE USED:' )
9994 FORMAT( ' FOR N ', 9I6 )
9993 FORMAT( ' FOR ALPHA ', 7F6.1 )
9992 FORMAT( ' FOR BETA ', 7F6.1 )
9991 FORMAT( ' AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM',&
  '/' ***** TESTS ABANDONED ***** )
9990 FORMAT( ' SUBPROGRAM NAME ', A6, ' NOT RECOGNIZED', '/' ***** T',&
  'ESTS ABANDONED ***** )
9989 FORMAT( ' ERROR IN DMCH - IN-LINE DOT PRODUCTS ARE BEING EVALU',&
  'ATED WRONGLY.', '/' DMCH WAS CALLED WITH TRANSA = ', A1,&
  ' AND TRANSB = ', A1, '/' AND RETURNED SAME = ', L1, ' AND ',&
  'ERR = ', F12.3, ' ', '/' THIS MAY BE DUE TO FAULTS IN THE ',&
  'ARITHMETIC OR THE COMPILER.', '/' ***** TESTS ABANDONED ',&
  '***** )
9988 FORMAT( A6, I2 )
9987 FORMAT( 1X, A6, ' WAS NOT TESTED' )
9986 FORMAT( '/' END OF TESTS' )
9985 FORMAT( '/' ***** FATAL ERROR - TESTS ABANDONED ***** )
9984 FORMAT( ' ERROR-EXITS WILL NOT BE TESTED' )
!
! End of DBLAT3.
!
END SUBROUTINE

SUBROUTINE DCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
  FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
  A, AA, AS, B, BB, BS, C, CC, CS, CT, G )

```

```

! Tests DGEMM.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Parameters ..
DOUBLE PRECISION ZERO
PARAMETER ( ZERO = 0.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER NALF, NBET, NIDIM, NMAX, NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
! .. Array Arguments ..
DOUBLE PRECISION A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), &
AS( NMAX*NMAX ), B( NMAX, NMAX ), &
BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ), &
C( NMAX, NMAX ), CC( NMAX*NMAX ), &
CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
INTEGER IDIM( NIDIM )
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, ALS, BETA, BLS, ERR, ERRMAX
INTEGER I, IA, IB, ICA, ICB, IK, IM, IN, K, KS, LAA, &
LBB, LCC, LDA, LDAS, LDB, LDBS, LDC, LDCS, M, &
MA, MB, MS, N, NA, NARGS, NB, NC, NS
LOGICAL NULL, RESET, SAME, TRANA, TRANB
CHARACTER*1 TRANAS, TRANBS, TRANSA, TRANSB
CHARACTER*3 ICH
! .. Local Arrays ..
LOGICAL ISAME( 13 )
! .. External Functions ..
LOGICAL LDE, LDERES
EXTERNAL LDE, LDERES
! .. External Subroutines ..
EXTERNAL DGEMM, DMAKE, DMMCH
! .. Intrinsic Functions ..
INTRINSIC MAX
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
DATA ICH/'NTC'/
! .. Executable Statements ..
!
NARGS = 13
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
DO 110 IM = 1, NIDIM
M = IDIM( IM )
!
DO 100 IN = 1, NIDIM
N = IDIM( IN )
! Set LDC to 1 more than minimum value if room.
LDC = M
IF( LDC.LT.NMAX ) &
LDC = LDC + 1
! Skip tests if not enough room.
IF( LDC.GT.NMAX ) &
GO TO 100
LCC = LDC*N
NULL = N.LE.0.OR.M.LE.0
!
DO 90 IK = 1, NIDIM
K = IDIM( IK )
!
DO 80 ICA = 1, 3
TRANSA = ICH( ICA: ICA )
TRANA = TRANSA.EQ.'T'.OR.TRANSA.EQ.'C'
!
IF( TRANA )THEN
MA = K
NA = M
ELSE
MA = M
NA = K
END IF
! Set LDA to 1 more than minimum value if room.
LDA = MA
IF( LDA.LT.NMAX ) &
LDA = LDA + 1
! Skip tests if not enough room.
IF( LDA.GT.NMAX ) &
GO TO 80
LAA = LDA*NA
!
! Generate the matrix A.
!
CALL DMAKE( 'GE', ' ', ' ', MA, NA, A, NMAX, AA, LDA, &
RESET, ZERO )
!
DO 70 ICB = 1, 3
TRANSB = ICH( ICB: ICB )
TRANSB = TRANSB.EQ.'T'.OR.TRANSB.EQ.'C'
!
IF( TRANSB )THEN
MB = N
NB = K
ELSE
MB = K
NB = N
END IF
! Set LDB to 1 more than minimum value if room.
LDB = MB
IF( LDB.LT.NMAX ) &
LDB = LDB + 1

```

```

!
! Skip tests if not enough room.
! IF( LDB.GT.NMAX )&
!   GO TO 70
!   LBB = LDB*NB
!
! Generate the matrix B.
!
! CALL DMAKE( 'GE', ' ', ' ', MB, NB, B, NMAX, BB,&
!   LDB, RESET, ZERO )
!
! DO 60 IA = 1, NALF
!   ALPHA = ALF( IA )
!
!   DO 50 IB = 1, NBET
!     BETA = BET( IB )
!
!     Generate the matrix C.
!
!     CALL DMAKE( 'GE', ' ', ' ', M, N, C, NMAX,&
!       CC, LDC, RESET, ZERO )
!
!     NC = NC + 1
!
!     Save every datum before calling the
!     subroutine.
!
!     TRANAS = TRANSA
!     TRANBS = TRANSB
!     MS = M
!     NS = N
!     KS = K
!     ALS = ALPHA
!     DO 10 I = 1, LAA
!       AS( I ) = AA( I )
!     CONTINUE
!     LDAS = LDA
!     DO 20 I = 1, LBB
!       BS( I ) = BB( I )
!     CONTINUE
!     LDBS = LDB
!     BLS = BETA
!     DO 30 I = 1, LCC
!       CS( I ) = CC( I )
!     CONTINUE
!     LDCS = LDC
!
!     Call the subroutine.
!
!     IF( TRACE )&
!       WRITE( NTRA, FMT = 9995 )NC, SNAME,&
!         TRANSA, TRANSB, M, N, K, ALPHA, LDA, LDB,&
!         BETA, LDC
!     IF( REWI )&
!       REWIND NTRA
!     CALL DGEMM( TRANSA, TRANSB, M, N, K, ALPHA,&
!       AA, LDA, BB, LDB, BETA, CC, LDC )
!
!     Check if error-exit was taken incorrectly.
!
!     IF( .NOT.OK )THEN
!       WRITE( NOUT, FMT = 9994 )
!       FATAL = .TRUE.
!       GO TO 120
!     END IF
!
!     See what data changed inside subroutines.
!
!     ISAME( 1 ) = TRANSA.EQ.TRANAS
!     ISAME( 2 ) = TRANSB.EQ.TRANBS
!     ISAME( 3 ) = MS.EQ.M
!     ISAME( 4 ) = NS.EQ.N
!     ISAME( 5 ) = KS.EQ.K
!     ISAME( 6 ) = ALS.EQ.ALPHA
!     ISAME( 7 ) = LDE( AS, AA, LAA )
!     ISAME( 8 ) = LDAS.EQ.LDA
!     ISAME( 9 ) = LDE( BS, BB, LBB )
!     ISAME( 10 ) = LDBS.EQ.LDB
!     ISAME( 11 ) = BLS.EQ.BETA
!     IF( NULL )THEN
!       ISAME( 12 ) = LDE( CS, CC, LCC )
!     ELSE
!       ISAME( 12 ) = LDERES( 'GE', ' ', M, N, CS,&
!         CC, LDC )
!     END IF
!     ISAME( 13 ) = LDCS.EQ.LDC
!
!     If data was incorrectly changed, report
!     and return.
!
!     SAME = .TRUE.
!     DO 40 I = 1, NARGS
!       SAME = SAME.AND.ISAME( I )
!       IF( .NOT.ISAME( I ) )&
!         WRITE( NOUT, FMT = 9998 )I
!     CONTINUE
!     IF( .NOT.SAME )THEN
!       FATAL = .TRUE.
!       GO TO 120
!     END IF
!
!     IF( .NOT.NULL )THEN
!
!       Check the result.
!
!       CALL DMMCH( TRANSA, TRANSB, M, N, K,&
!         ALPHA, A, NMAX, B, NMAX, BETA,&
!         C, NMAX, CT, G, CC, LDC, EPS,&
!         ERR, FATAL, NOUT, .TRUE. )
!       ERRMAX = MAX( ERRMAX, ERR )
!       If got really bad answer, report and
!       return.
!       IF( FATAL )&
!         GO TO 120
!     END IF

```

```

50          CONTINUE
!
!          CONTINUE
!
70          CONTINUE
!
!          CONTINUE
!
90          CONTINUE
!
!          CONTINUE
!
100         CONTINUE
!
!          CONTINUE
!
!          Report result.
!
!          IF( ERRMAX.LT.THRESH )THEN
!            WRITE( NOUT, FMT = 9999 )SNAME, NC
!          ELSE
!            WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
!          END IF
!          GO TO 130
!
!
120        CONTINUE
!          WRITE( NOUT, FMT = 9996 )SNAME
!          WRITE( NOUT, FMT = 9995 )NC, SNAME, TRANSA, TRANSB, M, N, K,&
!            ALPHA, LDA, LDB, BETA, LDC
!
!
130        CONTINUE
!          RETURN
!
!
9999       FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!            'S)' )
9998       FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!            'ANGED INCORRECTLY *****' )
9997       FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!            'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!            ' - SUSPECT' )
9996       FORMAT( ' *****', A6, ' FAILED ON CALL NUMBER:' )
9995       FORMAT( '1X, I6, ', ' ', A6, '(', '', A1, ' ', ' ', ' ', A1, ' ', ' ', ' ',&
!            '3 I3, ', ' ', ' ', F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ', ' ', F4.1, ' ', ' ',&
!            'C', ' ', I3, ' ', ' ')
9994       FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
!            '*****' )
!
!          End of DCHK1.
!
!          END SUBROUTINE
!          SUBROUTINE DCHK2( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!            FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
!            A, AA, AS, B, BB, BS, C, CC, CS, CT, G )
!
!          Tests DSYMM.
!
!
!          Auxiliary routine for test program for Level 3 Blas.
!
!          -- Written on 8-February-1989.
!          Jack Dongarra, Argonne National Laboratory.
!          Iain Duff, AERE Harwell.
!          Jeremy Du Croz, Numerical Algorithms Group Ltd.
!          Sven Hammarling, Numerical Algorithms Group Ltd.
!
!          .. Parameters ..
!          DOUBLE PRECISION      ZERO
!          PARAMETER              ( ZERO = 0.0D0 )
!          .. Scalar Arguments ..
!          DOUBLE PRECISION      EPS, THRESH
!          INTEGER                NALF, NBET, NIDIM, NMAX, NOUT, NTRA
!          LOGICAL                FATAL, REWI, TRACE
!          CHARACTER*6            SNAME
!          .. Array Arguments ..
!          DOUBLE PRECISION      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
!            AS( NMAX*NMAX ), B( NMAX, NMAX ),&
!            BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),&
!            C( NMAX, NMAX ), CC( NMAX*NMAX ),&
!            CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
!          INTEGER                IDIM( NIDIM )
!          .. Local Scalars ..
!          DOUBLE PRECISION      ALPHA, ALS, BETA, BLS, ERR, ERRMAX
!          INTEGER                I, IA, IB, ICS, ICU, IM, IN, IAA, LBB, LCC,&
!            LDA, LDAS, LDB, LDBS, LDC, LDCS, M, MS, N, NA,&
!            NARGS, NC, NS
!          LOGICAL                LEFT, NULL, RESET, SAME
!          CHARACTER*1            SIDE, SIDES, UPLO, UPLOS
!          CHARACTER*2            ICHS, ICHU
!          .. Local Arrays ..
!          LOGICAL                ISAME( 13 )
!          .. External Functions ..
!          LOGICAL                LDE, LDERES
!          EXTERNAL                LDE, LDERES
!          .. External Subroutines ..
!          EXTERNAL                DMAKE, DMMCH, DSYMM
!          .. Intrinsic Functions ..
!          INTRINSIC                MAX
!          .. Scalars in Common ..
!          INTEGER                INFOT, NOUTC
!          LOGICAL                LERR, OK
!          .. Common blocks ..
!          COMMON                /INFOC/INFOT, NOUTC, OK, LERR
!          .. Data statements ..
!          DATA                ICHS/'LR'/, ICHU/'UL'/
!          .. Executable Statements ..
!
!          NARGS = 12
!          NC = 0
!          RESET = .TRUE.
!          ERRMAX = ZERO
!
!          DO 100 IM = 1, NIDIM
!            M = IDIM( IM )
!
!
!          DO 90 IN = 1, NIDIM
!            N = IDIM( IN )

```

```

!      Set LDC to 1 more than minimum value if room.
LDC = M
IF( LDC.LT.NMAX )&
    LDC = LDC + 1
!      Skip tests if not enough room.
IF( LDC.GT.NMAX )&
    GO TO 90
LCC = LDC*N
NULL = N.LE.0.OR.M.LE.0
!
!      Set LDB to 1 more than minimum value if room.
LDB = M
IF( LDB.LT.NMAX )&
    LDB = LDB + 1
!      Skip tests if not enough room.
IF( LDB.GT.NMAX )&
    GO TO 90
LBB = LDB*N
!
!      Generate the matrix B.
CALL DMAKE( 'GE', ' ', ' ', M, N, B, NMAX, BB, LDB, RESET,&
    ZERO )
!
DO 80 ICS = 1, 2
    SIDE = ICHS( ICS: ICS )
    LEFT = SIDE.EQ.'L'
!
    IF( LEFT )THEN
        NA = M
    ELSE
        NA = N
    END IF
!      Set LDA to 1 more than minimum value if room.
LDA = NA
IF( LDA.LT.NMAX )&
    LDA = LDA + 1
!      Skip tests if not enough room.
IF( LDA.GT.NMAX )&
    GO TO 80
LAA = LDA*NA
!
DO 70 ICU = 1, 2
    UPLO = ICHU( ICU: ICU )
!
!      Generate the symmetric matrix A.
CALL DMAKE( 'SY', UPLO, ' ', NA, NA, A, NMAX, AA, LDA,&
    RESET, ZERO )
!
DO 60 IA = 1, NALF
    ALPHA = ALF( IA )
!
DO 50 IB = 1, NBET
    BETA = BET( IB )
!
!      Generate the matrix C.
CALL DMAKE( 'GE', ' ', ' ', M, N, C, NMAX, CC,&
    LDC, RESET, ZERO )
!
NC = NC + 1
!
!      Save every datum before calling the
!      subroutine.
!
SIDES = SIDE
UPLOS = UPLO
MS = M
NS = N
ALS = ALPHA
DO 10 I = 1, LAA
    AS( I ) = AA( I )
10 CONTINUE
LDAS = LDA
DO 20 I = 1, LBB
    BS( I ) = BB( I )
20 CONTINUE
LDBS = LDB
BLS = BETA
DO 30 I = 1, LCC
    CS( I ) = CC( I )
30 CONTINUE
LDCS = LDC
!
!      Call the subroutine.
!
IF( TRACE )&
    WRITE( NTRA, FMT = 9995 )NC, SNAME, SIDE,&
        UPLO, M, N, ALPHA, LDA, LDB, BETA, LDC
IF( REWI )&
    REWIND NTRA
CALL DSymm( SIDE, UPLO, M, N, ALPHA, AA, LDA,&
    BB, LDB, BETA, CC, LDC )
!
!      Check if error-exit was taken incorrectly.
!
IF( .NOT.OK )THEN
    WRITE( NOUT, FMT = 9994 )
    FATAL = .TRUE.
    GO TO 110
END IF
!
!      See what data changed inside subroutines.
!
ISAME( 1 ) = SIDES.EQ.SIDE
ISAME( 2 ) = UPLOS.EQ.UPLO
ISAME( 3 ) = MS.EQ.M
ISAME( 4 ) = NS.EQ.N
ISAME( 5 ) = ALS.EQ.ALPHA
ISAME( 6 ) = LDE( AS, AA, LAA )
ISAME( 7 ) = LDAS.EQ.LDA
ISAME( 8 ) = LDE( BS, BB, LBB )
ISAME( 9 ) = LDBS.EQ.LDB

```

```

ISAME( 10 ) = BLS.EQ.BETA
IF( NULL )THEN
  ISAME( 11 ) = LDE( CS, CC, LCC )
ELSE
  ISAME( 11 ) = LDERES( 'GE', ' ', M, N, CS,&
    CC, LDC )
END IF
ISAME( 12 ) = LDCS.EQ.LDC

!
! If data was incorrectly changed, report and
! return.
!

SAME = .TRUE.
DO 40 I = 1, NARGS
  SAME = SAME.AND.ISAME( I )
  IF( .NOT.ISAME( I ) )&
    WRITE( NOUT, FMT = 9998 ) I
40 CONTINUE
IF( .NOT.SAME )THEN
  FATAL = .TRUE.
  GO TO 110
END IF

IF( .NOT.NULL )THEN
  Check the result.

  IF( LEFT )THEN
    CALL DMMCH( 'N', 'N', M, N, M, ALPHA, A,&
      NMAX, B, NMAX, BETA, C, NMAX,&
      CT, G, CC, LDC, EPS, ERR,&
      FATAL, NOUT, .TRUE. )
  ELSE
    CALL DMMCH( 'N', 'N', M, N, N, ALPHA, B,&
      NMAX, A, NMAX, BETA, C, NMAX,&
      CT, G, CC, LDC, EPS, ERR,&
      FATAL, NOUT, .TRUE. )
  END IF
  ERRMAX = MAX( ERRMAX, ERR )
  If got really bad answer, report and
  return.
  IF( FATAL )&
    GO TO 110
END IF

!
! 50 CONTINUE
!
! 60 CONTINUE
!
! 70 CONTINUE
!
! 80 CONTINUE
!
! 90 CONTINUE
!
! 100 CONTINUE
!
! Report result.
!
IF( ERRMAX.LT.THRESH )THEN
  WRITE( NOUT, FMT = 9999 ) SNAME, NC
ELSE
  WRITE( NOUT, FMT = 9997 ) SNAME, NC, ERRMAX
END IF
GO TO 120

!
! 110 CONTINUE
!
WRITE( NOUT, FMT = 9996 ) SNAME
WRITE( NOUT, FMT = 9995 ) NC, SNAME, SIDE, UPLO, M, N, ALPHA, LDA,&
  LDB, BETA, LDC
!
! 120 CONTINUE
!
RETURN

!
! 9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
! 'S)' )
! 9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
! 'ANGED INCORRECTLY *****' )
! 9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
! 'ALLS)', '/', ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
! ' - SUSPECT *****' )
! 9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
! 9995 FORMAT( 1X, I6, ': ', A6, '(', 2( ' ', A1, ' ', ' '), 2( I3, ' ', ' '),&
! F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ', ' ', F4.1, ' ', C, ' ', I3, ' ', ' '),&
! ' ' )
! 9994 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
! '*****' )
!
! End of DCHK2.
!
END SUBROUTINE
SUBROUTINE DCHK3( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
  FATAL, NIDIM, IDIM, NALF, ALF, NMAX, A, AA, AS,&
  B, BB, BS, CT, G, C )

!
! Tests DTRMM and DTRSM.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER ( ZERO = 0.0D0, ONE = 1.0D0 )
!
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER NALF, NIDIM, NMAX, NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
!
! .. Array Arguments ..

```

```

DOUBLE PRECISION  A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
                   AS( NMAX*NMAX ), B( NMAX, NMAX ),&
                   BB( NMAX*NMAX ), BS( NMAX*NMAX ),&
                   C( NMAX, NMAX ), CT( NMAX ), G( NMAX )
INTEGER           IDIM( NIDIM )
!
! .. Local Scalars ..
DOUBLE PRECISION  ALPHA, ALS, ERR, ERRMAX
INTEGER           I, IA, ICD, ICS, ICT, ICU, IM, IN, J, LAA, LBB,&
                   LDA, LDAS, LDB, LDBS, M, MS, N, NA, NARGS, NC,&
                   NS
LOGICAL           LEFT, NULL, RESET, SAME
CHARACTER*1       DIAG, DIAS, SIDE, SIDES, TRANAS, TRANSA, UPLO,&
CHARACTER*2       ICHD, ICHS, ICHU
CHARACTER*3       ICHT
!
! .. Local Arrays ..
LOGICAL           ISAME( 13 )
!
! .. External Functions ..
LOGICAL           LDE, LDERES
EXTERNAL          LDE, LDERES
!
! .. External Subroutines ..
EXTERNAL          DMAKE, DMMCH, DTRMM, DTRSM
!
! .. Intrinsic Functions ..
INTRINSIC         MAX
!
! .. Scalars in Common ..
INTEGER           INFOT, NOUTC, ZZ, ZZ1
LOGICAL           LERR, OK
!
! .. Common blocks ..
COMMON            /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA             ICHU/'UL'/, ICHT/'NTC'/, ICHD/'UN'/, ICHS/'LR'/
!
! .. Executable Statements ..
!
NARGS = 11
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
! Set up zero matrix for DMMCH.
DO 20 J = 1, NMAX
    DO 10 I = 1, NMAX
        C( I, J ) = ZERO
10    CONTINUE
20    CONTINUE
!
DO 140 IM = 1, NIDIM
    M = IDIM( IM )
!
DO 130 IN = 1, NIDIM
    N = IDIM( IN )
!
! Set LDB to 1 more than minimum value if room.
LDB = M
IF( LDB.LT.NMAX )&
    LDB = LDB + 1
!
! Skip tests if not enough room.
IF( LDB.GT.NMAX )&
    GO TO 130
LBB = LDB*N
NULL = M.LE.0.OR.N.LE.0
!
DO 120 ICS = 1, 2
    SIDE = ICHS( ICS: ICS )
    LEFT = SIDE.EQ.'L'
    IF( LEFT ) THEN
        NA = M
    ELSE
        NA = N
    END IF
!
! Set LDA to 1 more than minimum value if room.
LDA = NA
IF( LDA.LT.NMAX )&
    LDA = LDA + 1
!
! Skip tests if not enough room.
IF( LDA.GT.NMAX )&
    GO TO 130
LAA = LDA*NA
!
DO 110 ICU = 1, 2
    UPLO = ICHU( ICU: ICU )
!
DO 100 ICT = 1, 3
    TRANSA = ICHT( ICT: ICT )
!
DO 90 ICD = 1, 2
    DIAG = ICHD( ICD: ICD )
!
DO 80 IA = 1, NALF
    ALPHA = ALF( IA )
!
! Generate the matrix A.
CALL DMAKE( 'TR', UPLO, DIAG, NA, NA, A,&
            NMAX, AA, LDA, RESET, ZERO )
!
IF ( (ICD.EQ.2).AND.(IA.EQ.2) ) THEN
    WRITE( NOUT, *) 'a='
    DO 711 ZZ = 1, NMAX
        DO 721 ZZ1 = 1, NMAX
            WRITE( NOUT, *) ZZ, ', ', ZZ1, ': ', A(ZZ,ZZ1)
721    CONTINUE
711    CONTINUE
!
    WRITE( NOUT, *) 'aa='
    DO 731 ZZ = 1, NMAX*NMAX
        WRITE( NOUT, *) ZZ, ': ', AA(ZZ)
731    CONTINUE
!
ENDIF
!
IF ( .TRUE. ) THEN !SNAME(1:5).EQ.'DTRSM'.AND. NC .GT. 502) THEN
    WRITE( NOUT, *) 'a='
    DO 713 ZZ = 1, NMAX
        DO 723 ZZ1 = 1, NMAX
            WRITE( NOUT, *) ZZ, ', ', ZZ1, ': ', A(ZZ,ZZ1)
723    CONTINUE
!

```

```

! 713 CONTINUE
!
! WRITE( NOUT, *)'aa='
! DO 733 ZZ = 1, NMAX*NMAX
!
! WRITE( NOUT, *)ZZ,': ',AA(ZZ)
!
! 733 CONTINUE
!
! ENDIF
! Generate the matrix B.
!
! CALL DMAKE( 'GE', ' ', ' ', M, N, B, NMAX,&
! BB, LDB, RESET, ZERO )
! IF (.TRUE.) THEN !SNAME(1:5).EQ.'DTRSM' .AND. NC .GT. 502) THEN
! WRITE( NOUT, *)'b='
! DO 712 ZZ = 1, NMAX
! DO 722 ZZ1 = 1, NMAX
! WRITE( NOUT, *)ZZ,': ',ZZ1,': ',B(ZZ,ZZ1)
! 722 CONTINUE
! 712 CONTINUE
!
! WRITE( NOUT, *)'bb='
! DO 732 ZZ = 1, NMAX*NMAX
!
! WRITE( NOUT, *)ZZ,': ',BB(ZZ)
!
! 732 CONTINUE
!
! STOP
! ENDIF
!!
!STOP
!write(nout,*)'nc=',nc
NC = NC + 1
!
! Save every datum before calling the
! subroutine.
!
! SIDES = SIDE
! UPLOS = UPLO
! TRANAS = TRANSA
! DIAGS = DIAG
! MS = M
! NS = N
! ALS = ALPHA
! DO 30 I = 1, LAA
! AS( I ) = AA( I )
! 30 CONTINUE
! LDAS = LDA
! DO 40 I = 1, LBB
! BS( I ) = BB( I )
! 40 CONTINUE
! LDBS = LDB
!
! Call the subroutine.
!
! IF( SNAME( 4: 5 ).EQ.'MM' )THEN
! IF( TRACE )&
! WRITE( NTRA, FMT = 9995 )NC, SNAME,&
! SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA,&
! LDA, LDB
! IF( REWI )&
! REWIND NTRA
! CALL DTRMM( SIDE, UPLO, TRANSA, DIAG, M,&
! N, ALPHA, AA, LDA, BB, LDB )
! ELSE IF( SNAME( 4: 5 ).EQ.'SM' )THEN
! IF( TRACE )&
! WRITE( NTRA, FMT = 9995 )NC, SNAME,&
! SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA,&
! LDA, LDB
! IF( REWI )&
! REWIND NTRA
! CALL DTRSM( SIDE, UPLO, TRANSA, DIAG, M,&
! N, ALPHA, AA, LDA, BB, LDB )
! END IF
!
! Check if error-exit was taken incorrectly.
!
! IF( .NOT.OK )THEN
! WRITE( NOUT, FMT = 9994 )
! FATAL = .TRUE.
! GO TO 150
! END IF
!
! See what data changed inside subroutines.
!
! ISAME( 1 ) = SIDES.EQ.SIDE
! ISAME( 2 ) = UPLOS.EQ.UPLO
! ISAME( 3 ) = TRANAS.EQ.TRANSA
! ISAME( 4 ) = DIAGS.EQ.DIAG
! ISAME( 5 ) = MS.EQ.M
! ISAME( 6 ) = NS.EQ.N
! ISAME( 7 ) = ALS.EQ.ALPHA
! ISAME( 8 ) = LDE( AS, AA, LAA )
! ISAME( 9 ) = LDAS.EQ.LDA
! IF( NULL )THEN
! ISAME( 10 ) = LDE( BS, BB, LBB )
! ELSE
! ISAME( 10 ) = LDERES( 'GE', ' ', M, N, BS,&
! BB, LDB )
! END IF
! ISAME( 11 ) = LDBS.EQ.LDB
!
! If data was incorrectly changed, report and
! return.
!
! SAME = .TRUE.
! DO 50 I = 1, NARGS
! SAME = SAME.AND.ISAME( I )
! IF( .NOT.ISAME( I ) )&
! WRITE( NOUT, FMT = 9998 )I
! 50 CONTINUE
! IF( .NOT.SAME )THEN
! FATAL = .TRUE.
! GO TO 150
! END IF

```



```

9995 FORMAT( 1X, I6, ': ', A6, '(', 4( ' ', A1, ' ', ' '), 2( I3, ' ', ' '),&
      F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ' )
9994 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
      '*****' )
!
! End of DCHK3.
!
END SUBROUTINE
SUBROUTINE DCHK4( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
      FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
      A, AA, AS, B, BB, BS, C, CC, CS, CT, G )
!
! Tests DSYRK.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Parameters ..
DOUBLE PRECISION      ZERO
PARAMETER              ( ZERO = 0.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION      EPS, THRESH
INTEGER                NALF, NBET, NIDIM, NMAX, NOUT, NTRA
LOGICAL                FATAL, REWI, TRACE
CHARACTER*6           SNAME
! .. Array Arguments ..
DOUBLE PRECISION      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
      AS( NMAX*NMAX ), B( NMAX, NMAX ),&
      BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),&
      C( NMAX, NMAX ), CC( NMAX*NMAX ),&
      CS( NMAX*NMAX ), CT( NMAX ), G( NMAX )
INTEGER                IDIM( NIDIM )
! .. Local Scalars ..
DOUBLE PRECISION      ALPHA, ALS, BETA, BETS, ERR, ERRMAX
INTEGER                I, IA, IB, ICT, ICU, IK, IN, J, JC, JJ, K, KS,&
      LAA, LCC, LDA, LDAS, LDC, LDCS, LJ, MA, N, NA,&
      NARGS, NC, NS
LOGICAL                NULL, RESET, SAME, TRAN, UPPER
CHARACTER*1            TRANS, TRANSS, UPLO, UPLOS
CHARACTER*2            ICHU
CHARACTER*3            ICHT
! .. Local Arrays ..
LOGICAL                ISAME( 13 )
! .. External Functions ..
LOGICAL                LDE, LDERES
EXTERNAL               LDE, LDERES
! .. External Subroutines ..
EXTERNAL               DMAKE, DMMCH, DSYRK
! .. Intrinsic Functions ..
INTRINSIC              MAX
! .. Scalars in Common ..
INTEGER                INFOT, NOUTC
LOGICAL                LERR, OK
! .. Common blocks ..
COMMON                /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
DATA                  ICHT/'NTC'/, ICHU/'UL'/
! .. Executable Statements ..
!
NARGS = 10
NC = 0
RESET = .TRUE.
ERRMAX = ZERO
!
DO 100 IN = 1, NIDIM
      N = IDIM( IN )
      Set LDC to 1 more than minimum value if room.
      LDC = N
      IF( LDC.LT.NMAX )&
        LDC = LDC + 1
      Skip tests if not enough room.
      IF( LDC.GT.NMAX )&
        GO TO 100
      LCC = LDC*N
      NULL = N.LE.0
!
      DO 90 IK = 1, NIDIM
            K = IDIM( IK )
!
            DO 80 ICT = 1, 3
                  TRANS = ICHT( ICT: ICT )
                  TRAN = TRANS.EQ.'T'.OR.TRANS.EQ.'C'
                  IF( TRAN )THEN
                        MA = K
                        NA = N
                  ELSE
                        MA = N
                        NA = K
                  END IF
                  Set LDA to 1 more than minimum value if room.
                  LDA = MA
                  IF( LDA.LT.NMAX )&
                    LDA = LDA + 1
                  Skip tests if not enough room.
                  IF( LDA.GT.NMAX )&
                    GO TO 80
                  LAA = LDA*NA
!
                  Generate the matrix A.
!
                  CALL DMAKE( 'GE', ' ', ' ', MA, NA, A, NMAX, AA, LDA,&
                        RESET, ZERO )
!
                  DO 70 ICU = 1, 2
                        UPLO = ICHU( ICU: ICU )
                        UPPER = UPLO.EQ.'U'
!
                        DO 60 IA = 1, NALF
                              ALPHA = ALF( IA )

```

```

!
DO 50 IB = 1, NBET
  BETA = BET( IB )
!
!
!   Generate the matrix C.
!
  CALL DMAKE( 'SY', UPLO, ' ', N, N, C, NMAX, CC, &
    LDC, RESET, ZERO )
!
  NC = NC + 1
!
!   Save every datum before calling the subroutine.
!
  UPLOS = UPLO
  TRANSS = TRANS
  NS = N
  KS = K
  ALS = ALPHA
  DO 10 I = 1, LAA
    AS( I ) = AA( I )
10  CONTINUE
  LDAS = LDA
  BETS = BETA
  DO 20 I = 1, LCC
    CS( I ) = CC( I )
20  CONTINUE
  LDCS = LDC
!
!   Call the subroutine.
!
  IF( TRACE )&
    WRITE( NTRA, FMT = 9994 )NC, SNAME, UPLO,&
      TRANS, N, K, ALPHA, LDA, BETA, LDC
  IF( REWI )&
    REWIND NTRA
  CALL DSYRK( UPLO, TRANS, N, K, ALPHA, AA, LDA,&
    BETA, CC, LDC )
!
!   Check if error-exit was taken incorrectly.
!
  IF( .NOT.OK )THEN
    WRITE( NOUT, FMT = 9993 )
    FATAL = .TRUE.
    GO TO 120
  END IF
!
!   See what data changed inside subroutines.
!
  ISAME( 1 ) = UPLOS.EQ.UPLO
  ISAME( 2 ) = TRANSS.EQ.TRANS
  ISAME( 3 ) = NS.EQ.N
  ISAME( 4 ) = KS.EQ.K
  ISAME( 5 ) = ALS.EQ.ALPHA
  ISAME( 6 ) = LDE( AS, AA, LAA )
  ISAME( 7 ) = LDAS.EQ.LDA
  ISAME( 8 ) = BETS.EQ.BETA
  IF( NULL )THEN
    ISAME( 9 ) = LDE( CS, CC, LCC )
  ELSE
    ISAME( 9 ) = LDERES( 'SY', UPLO, N, N, CS,&
      CC, LDC )
  END IF
  ISAME( 10 ) = LDCS.EQ.LDC
!
!   If data was incorrectly changed, report and
!   return.
!
  SAME = .TRUE.
  DO 30 I = 1, NARGS
    SAME = SAME.AND.ISAME( I )
    IF( .NOT.ISAME( I ) )&
      WRITE( NOUT, FMT = 9998 )I
30  CONTINUE
  IF( .NOT.SAME )THEN
    FATAL = .TRUE.
    GO TO 120
  END IF
!
  IF( .NOT.NULL )THEN
    Check the result column by column.
    JC = 1
    DO 40 J = 1, N
      IF( UPPER )THEN
        JJ = 1
        LJ = J
      ELSE
        JJ = J
        LJ = N - J + 1
      END IF
      IF( TRAN )THEN
        CALL DMMCH( 'T', 'N', LJ, 1, K, ALPHA,&
          A( 1, JJ ), NMAX,&
          A( 1, J ), NMAX, BETA,&
          C( JJ, J ), NMAX, CT, G,&
          CC( JC ), LDC, EPS, ERR,&
          FATAL, NOUT, .TRUE. )
      ELSE
        CALL DMMCH( 'N', 'T', LJ, 1, K, ALPHA,&
          A( JJ, 1 ), NMAX,&
          A( J, 1 ), NMAX, BETA,&
          C( JJ, J ), NMAX, CT, G,&
          CC( JC ), LDC, EPS, ERR,&
          FATAL, NOUT, .TRUE. )
      END IF
      IF( UPPER )THEN
        JC = JC + LDC
      ELSE
        JC = JC + LDC + 1
      END IF
      ERRMAX = MAX( ERRMAX, ERR )
      If got really bad answer, report and
      return.
    END DO
  END IF
!
!
!

```



```

RESET = .TRUE.
ERRMAX = ZERO

!
DO 130 IN = 1, NIDIM
  N = IDIM( IN )
  Set LDC to 1 more than minimum value if room.
  LDC = N
  IF( LDC.LT.NMAX )&
    LDC = LDC + 1
!
  Skip tests if not enough room.
  IF( LDC.GT.NMAX )&
    GO TO 130
  LCC = LDC*N
  NULL = N.LE.0
!
DO 120 IK = 1, NIDIM
  K = IDIM( IK )
!
  DO 110 ICT = 1, 3
    TRANS = ICHT( ICT: ICT )
    TRAN = TRANS.EQ.'T'.OR.TRANS.EQ.'C'
    IF( TRAN )THEN
      MA = K
      NA = N
    ELSE
      MA = N
      NA = K
    END IF
!
    Set LDA to 1 more than minimum value if room.
    LDA = MA
    IF( LDA.LT.NMAX )&
      LDA = LDA + 1
!
    Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
      GO TO 110
    LAA = LDA*NA
!
!
    Generate the matrix A.
!
    IF( TRAN )THEN
      CALL DMAKE( 'GE', ' ', ' ', MA, NA, AB, 2*NMAX, AA,&
        LDA, RESET, ZERO )
    ELSE
      CALL DMAKE( 'GE', ' ', ' ', MA, NA, AB, NMAX, AA, LDA,&
        RESET, ZERO )
    END IF
!
!
    Generate the matrix B.
!
    LDB = LDA
    LBB = LAA
    IF( TRAN )THEN
      CALL DMAKE( 'GE', ' ', ' ', MA, NA, AB( K + 1 ),&
        2*NMAX, BB, LDB, RESET, ZERO )
    ELSE
      CALL DMAKE( 'GE', ' ', ' ', MA, NA, AB( K*NMAX + 1 ),&
        NMAX, BB, LDB, RESET, ZERO )
    END IF
!
DO 100 ICU = 1, 2
  UPLO = ICHU( ICU: ICU )
  UPPER = UPLO.EQ.'U'
!
DO 90 IA = 1, NALF
  ALPHA = ALF( IA )
!
DO 80 IB = 1, NBET
  BETA = BET( IB )
!
!
  Generate the matrix C.
!
  CALL DMAKE( 'SY', UPLO, ' ', N, N, C, NMAX, CC,&
    LDC, RESET, ZERO )
!
  NC = NC + 1
!
!
  Save every datum before calling the subroutine.
!
  UPLOS = UPLO
  TRANSS = TRANS
  NS = N
  KS = K
  ALS = ALPHA
  DO 10 I = 1, LAA
    AS( I ) = AA( I )
10  CONTINUE
  LDAS = LDA
  DO 20 I = 1, LBB
    BS( I ) = BB( I )
20  CONTINUE
  LDBS = LDB
  BETS = BETA
  DO 30 I = 1, LCC
    CS( I ) = CC( I )
30  CONTINUE
  LDCS = LDC
!
!
  Call the subroutine.
!
  IF( TRACE )&
    WRITE( NTRA, FMT = 9994 )NC, SNAME, UPLO,&
      TRANS, N, K, ALPHA, LDA, LDB, BETA, LDC
  IF( REWI )&
    REWIND NTRA
  CALL DSYR2K( UPLO, TRANS, N, K, ALPHA, AA, LDA,&
    BB, LDB, BETA, CC, LDC )
!
!
  Check if error-exit was taken incorrectly.
!
  IF( .NOT.OK )THEN
    WRITE( NOUT, FMT = 9993 )
    FATAL = .TRUE.
    GO TO 150
  END IF

```

```

!
!
!
See what data changed inside subroutines.

ISAME( 1 ) = UPLOS.EQ.UPLO
ISAME( 2 ) = TRANS.EQ.TRANS
ISAME( 3 ) = NS.EQ.N
ISAME( 4 ) = KS.EQ.K
ISAME( 5 ) = ALS.EQ.ALPHA
ISAME( 6 ) = LDE( AS, AA, LAA )
ISAME( 7 ) = LDAS.EQ.LDA
ISAME( 8 ) = LDE( BS, BB, LBB )
ISAME( 9 ) = LDBS.EQ.LDB
ISAME( 10 ) = BETS.EQ.BETA
IF( NULL )THEN
    ISAME( 11 ) = LDE( CS, CC, LCC )
ELSE
    ISAME( 11 ) = LDERES( 'SY', UPLO, N, N, CS,&
        CC, LDC )
END IF
ISAME( 12 ) = LDCS.EQ.LDC

!
!
!
If data was incorrectly changed, report and
return.

!
!
!
SAME = .TRUE.
DO 40 I = 1, NARGS
    SAME = SAME.AND.ISAME( I )
    IF( .NOT.ISAME( I ) )&
        WRITE( NOUT, FMT = 9998 ) I
CONTINUE
IF( .NOT.SAME )THEN
    FATAL = .TRUE.
    GO TO 150
END IF

!
!
!
IF( .NOT.NULL )THEN

    Check the result column by column.

    JJAB = 1
    JC = 1
    DO 70 J = 1, N
        IF( UPPER )THEN
            JJ = 1
            LJ = J
        ELSE
            JJ = J
            LJ = N - J + 1
        END IF
        IF( TRAN )THEN
            DO 50 I = 1, K
                W( I ) = AB( ( J - 1 ) * 2 * NMAX + K +&
                    I )
                W( K + I ) = AB( ( J - 1 ) * 2 * NMAX +&
                    I )
            CONTINUE
            CALL DMMCH( 'T', 'N', LJ, 1, 2 * K,&
                ALPHA, AB( JJAB ), 2 * NMAX,&
                W, 2 * NMAX, BETA,&
                C( JJ, J ), NMAX, CT, G,&
                CC( JC ), LDC, EPS, ERR,&
                FATAL, NOUT, .TRUE. )
        ELSE
            DO 60 I = 1, K
                W( I ) = AB( ( K + I - 1 ) * NMAX +&
                    J )
                W( K + I ) = AB( ( I - 1 ) * NMAX +&
                    J )
            CONTINUE
            CALL DMMCH( 'N', 'N', LJ, 1, 2 * K,&
                ALPHA, AB( JJ ), NMAX, W,&
                2 * NMAX, BETA, C( JJ, J ),&
                NMAX, CT, G, CC( JC ), LDC,&
                EPS, ERR, FATAL, NOUT,&
                .TRUE. )
        END IF
        IF( UPPER )THEN
            JC = JC + LDC
        ELSE
            JC = JC + LDC + 1
            IF( TRAN )&
                JJAB = JJAB + 2 * NMAX
        END IF
        ERRMAX = MAX( ERRMAX, ERR )
        If got really bad answer, report and
        return.
        IF( FATAL )&
            GO TO 140
    70 CONTINUE
    END IF

!
!
!
80 CONTINUE
!
!
!
90 CONTINUE
!
!
!
100 CONTINUE
!
!
!
110 CONTINUE
!
!
!
120 CONTINUE
!
!
!
130 CONTINUE
!
!
!
Report result.
!
IF( ERRMAX.LT.THRESH )THEN
    WRITE( NOUT, FMT = 9999 ) SNAME, NC
ELSE
    WRITE( NOUT, FMT = 9997 ) SNAME, NC, ERRMAX
END IF
GO TO 160

!
!
!
140 CONTINUE
IF( N.GT.1 )&

```

```

        WRITE( NOUT, FMT = 9995 )J
!
150 CONTINUE
    WRITE( NOUT, FMT = 9996 )SNAME
    WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, TRANS, N, K, ALPHA,&
        LDA, LDB, BETA, LDC
!
160 CONTINUE
    RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
    'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
    'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
    'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
    ' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( '          THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( ' 1X, I6, ': ', A6, '( ', 2( ' ', A1, ' ', ' '), 2( I3, ' ', ' '),&
    F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ', F4.1, ' ', C, ' ', I3, ' ' ),&
    ' ' )
9993 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
    '*****' )
!
!     End of DCHK5.
!
END SUBROUTINE
SUBROUTINE DCHKE( ISNUM, SRNAMT, NOUT )
!
! Tests the error exits from the Level 3 Blas.
! Requires a special version of the error-handling routine XERBLA.
! A, B and C should not need to be defined.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! 3-19-92:  Initialize ALPHA and BETA (eca)
! 3-19-92:  Fix argument I2 in calls to SSYMM with INFOT = 9 (eca)
!
! .. Scalar Arguments ..
INTEGER          ISNUM, NOUT
CHARACTER*6      SRNAMT
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
! .. Parameters ..
DOUBLE PRECISION ONE, TWO
PARAMETER        ( ONE = 1.0D0, TWO = 2.0D0 )
! .. Local Scalars ..
DOUBLE PRECISION ALPHA, BETA
! .. Local Arrays ..
DOUBLE PRECISION A( 2, 1 ), B( 2, 1 ), C( 2, 1 )
! .. External Subroutines ..
EXTERNAL         CHKXER, DGEMM, DSYMM, DSYR2K, DSYRK, DTRMM,&
    DTRSM
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUTC, OK, LERR
! .. Executable Statements ..
! OK is set to .FALSE. by the special version of XERBLA or by CHKXER
! if anything is wrong.
! OK = .TRUE.
! LERR is set to .TRUE. by the special version of XERBLA each time
! it is called, and is then tested and re-set by CHKXER.
! LERR = .FALSE.
!
! Initialize ALPHA and BETA.
!
ALPHA = ONE
BETA = TWO
!
GO TO ( 10, 20, 30, 40, 50, 60 )ISNUM
10 INFOT = 1
CALL DGEMM( '/', 'N', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 1
CALL DGEMM( '/', 'T', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DGEMM( 'N', '/', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL DGEMM( 'T', '/', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DGEMM( 'N', 'N', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DGEMM( 'N', 'T', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DGEMM( 'T', 'N', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL DGEMM( 'T', 'T', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DGEMM( 'N', 'N', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DGEMM( 'N', 'T', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DGEMM( 'T', 'N', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL DGEMM( 'T', 'T', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5

```



```

        WRITE( NOUT, FMT = 9999 )SRNAMT
    ELSE
        WRITE( NOUT, FMT = 9998 )SRNAMT
    END IF
    RETURN
!
! 9999 FORMAT( ' ', A6, ' PASSED THE TESTS OF ERROR-EXITS' )
! 9998 FORMAT( ' ***** ', A6, ' FAILED THE TESTS OF ERROR-EXITS *****', &
!           '***' )
!
! End of DCHKE.
!
END SUBROUTINE
SUBROUTINE DMAKE( TYPE, UPLO, DIAG, M, N, A, NMAX, AA, LDA, RESET,&
    TRANSL )
!
! Generates values for an M by N matrix A.
! Stores the values in the array AA in the data structure required
! by the routine, with unwanted elements set to rogue value.
!
! TYPE is 'GE', 'SY' or 'TR'.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Parameters ..
DOUBLE PRECISION    ZERO, ONE
PARAMETER            ( ZERO = 0.0D0, ONE = 1.0D0 )
DOUBLE PRECISION    ROGUE
PARAMETER            ( ROGUE = -1.0D10 )
! .. Scalar Arguments ..
DOUBLE PRECISION    TRANSL
INTEGER              LDA, M, N, NMAX
LOGICAL              RESET
CHARACTER*1          DIAG, UPLO
CHARACTER*2          TYPE
! .. Array Arguments ..
DOUBLE PRECISION    A( NMAX, * ), AA( * )
! .. Local Scalars ..
INTEGER              I, IBEG, IEND, J
LOGICAL              GEN, LOWER, SYM, TRI, UNIT, UPPER
! .. External Functions ..
DOUBLE PRECISION    DBEG
EXTERNAL             DBEG
! .. Executable Statements ..
GEN = TYPE.EQ.'GE'
SYM = TYPE.EQ.'SY'
TRI = TYPE.EQ.'TR'
UPPER = ( SYM.OR.TRI ).AND.UPLO.EQ.'U'
LOWER = ( SYM.OR.TRI ).AND.UPLO.EQ.'L'
UNIT = TRI.AND.DIAG.EQ.'U'
!
! Generate data in array A.
!
DO 20 J = 1, N
    DO 10 I = 1, M
        IF( GEN.OR.( UPPER.AND.I.LE.J ).OR.( LOWER.AND.I.GE.J ) )&
            THEN
            A( I, J ) = DBEG( RESET ) + TRANSL
            IF( I.NE.J )THEN
                Set some elements to zero
                IF( N.GT.3.AND.J.EQ.N/2 )&
                    A( I, J ) = ZERO
                IF( SYM )THEN
                    A( J, I ) = A( I, J )
                ELSE IF( TRI )THEN
                    A( J, I ) = ZERO
                END IF
            END IF
        END IF
    10 CONTINUE
    IF( TRI )&
        A( J, J ) = A( J, J ) + ONE
    IF( UNIT )&
        A( J, J ) = ONE
    20 CONTINUE
!
! Store elements in array AS in data structure required by routine.
!
IF( TYPE.EQ.'GE' )THEN
    DO 50 J = 1, N
        DO 30 I = 1, M
            AA( I + ( J - 1 )*LDA ) = A( I, J )
        30 CONTINUE
        DO 40 I = M + 1, LDA
            AA( I + ( J - 1 )*LDA ) = ROGUE
        40 CONTINUE
    50 CONTINUE
ELSE IF( TYPE.EQ.'SY'.OR.TYPE.EQ.'TR' )THEN
    DO 60 J = 1, N
        IF( UPPER )THEN
            IBEG = 1
            IF( UNIT )THEN
                IEND = J - 1
            ELSE
                IEND = J
            END IF
        ELSE
            IF( UNIT )THEN
                IBEG = J + 1
            ELSE
                IBEG = J
            END IF
            IEND = N
        END IF
        DO 60 I = 1, IBEG - 1
            AA( I + ( J - 1 )*LDA ) = ROGUE
        60 CONTINUE
        DO 70 I = IBEG, IEND

```

```

      AA( I + ( J - 1 ) * LDA ) = A( I, J )
70      CONTINUE
      DO 80 I = IEND + 1, LDA
        AA( I + ( J - 1 ) * LDA ) = ROGUE
80      CONTINUE
90      CONTINUE
      END IF
      RETURN
!
!      End of DMAKE.
!
      END SUBROUTINE
      SUBROUTINE DMMCH( TRANSA, TRANSB, M, N, KK, ALPHA, A, LDA, B, LDB, &
        BETA, C, LDC, CT, G, CC, LDCC, EPS, ERR, FATAL, &
        NOUT, MV )
!
!      Checks the results of the computational tests.
!
!      Auxiliary routine for test program for Level 3 Blas.
!
!      -- Written on 8-February-1989.
!      Jack Dongarra, Argonne National Laboratory.
!      Iain Duff, AERE Harwell.
!      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!      Sven Hammarling, Numerical Algorithms Group Ltd.
!
!      .. Parameters ..
      DOUBLE PRECISION    ZERO, ONE
      PARAMETER            ( ZERO = 0.0D0, ONE = 1.0D0 )
!      .. Scalar Arguments ..
      DOUBLE PRECISION    ALPHA, BETA, EPS, ERR
      INTEGER              KK, LDA, LDB, LDC, LDCC, M, N, NOUT
      LOGICAL              FATAL, MV
      CHARACTER*1          TRANSA, TRANSB
!      .. Array Arguments ..
      DOUBLE PRECISION    A( LDA, * ), B( LDB, * ), C( LDC, * ), &
        CC( LDCC, * ), CT( * ), G( * )
!      .. Local Scalars ..
      DOUBLE PRECISION    ERRI
      INTEGER              I, J, K
      LOGICAL              TRANA, TRANB
!      .. Intrinsic Functions ..
      INTRINSIC            ABS, MAX, SQRT
!      .. Executable Statements ..
      TRANA = TRANSA.EQ.'T'.OR.TRANSA.EQ.'C'
      TRANB = TRANSB.EQ.'T'.OR.TRANSB.EQ.'C'
!
!      Compute expected result, one column at a time, in CT using data
!      in A, B and C.
!      Compute gauges in G.
!
      DO 120 J = 1, N
!
!         DO 10 I = 1, M
!         CT( I ) = ZERO
!         G( I ) = ZERO
10      CONTINUE
      IF( .NOT.TRANA.AND..NOT.TRANB ) THEN
        DO 30 K = 1, KK
          DO 20 I = 1, M
            CT( I ) = CT( I ) + A( I, K ) * B( K, J )
            G( I ) = G( I ) + ABS( A( I, K ) ) * ABS( B( K, J ) )
20          CONTINUE
30        CONTINUE
      ELSE IF( TRANA.AND..NOT.TRANB ) THEN
        DO 50 K = 1, KK
          DO 40 I = 1, M
            CT( I ) = CT( I ) + A( K, I ) * B( K, J )
            G( I ) = G( I ) + ABS( A( K, I ) ) * ABS( B( K, J ) )
40          CONTINUE
50        CONTINUE
      ELSE IF( .NOT.TRANA.AND.TRANB ) THEN
        DO 70 K = 1, KK
          DO 60 I = 1, M
            CT( I ) = CT( I ) + A( I, K ) * B( J, K )
            G( I ) = G( I ) + ABS( A( I, K ) ) * ABS( B( J, K ) )
60          CONTINUE
70        CONTINUE
      ELSE IF( TRANA.AND.TRANB ) THEN
        DO 90 K = 1, KK
          DO 80 I = 1, M
            CT( I ) = CT( I ) + A( K, I ) * B( J, K )
            G( I ) = G( I ) + ABS( A( K, I ) ) * ABS( B( J, K ) )
80          CONTINUE
90        CONTINUE
      END IF
      DO 100 I = 1, M
        CT( I ) = ALPHA * CT( I ) + BETA * C( I, J )
        G( I ) = ABS( ALPHA ) * G( I ) + ABS( BETA ) * ABS( C( I, J ) )
100      CONTINUE
!
!      Compute the error ratio for this result.
!
      ERR = ZERO
      DO 110 I = 1, M
        ERRI = ABS( CT( I ) - CC( I, J ) ) / EPS
        IF( G( I ) .NE. ZERO ) &
          ERRI = ERRI / G( I )
        ERR = MAX( ERR, ERRI )
        IF( ERR * SQRT( EPS ) .GE. ONE ) &
          GO TO 130
110      CONTINUE
!
120      CONTINUE
!
!      If the loop completes, all results are at least half accurate.
      GO TO 150
!
!      Report fatal error.
!
130      FATAL = .TRUE.
      WRITE( NOUT, FMT = 9999 )
      DO 140 I = 1, M
        IF( MV ) THEN

```

```

        WRITE( NOUT, FMT = 9998 ) I, CT( I ), CC( I, J )
    ELSE
        WRITE( NOUT, FMT = 9998 ) I, CC( I, J ), CT( I )
    END IF
140 CONTINUE
    IF( N.GT.1 )&
        WRITE( NOUT, FMT = 9997 ) J
!
! 150 CONTINUE
    RETURN
!
9999 FORMAT( ' ***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HAL',&
    'F ACCURATE *****', '/'          EXPECTED RESULT  COMPU',&
    'TED RESULT' )
9998 FORMAT( 1X, I7, 2G18.6 )
9997 FORMAT( '      THESE ARE THE RESULTS FOR COLUMN ', I3 )
!
!      End of DMMCH.
!
!      END SUBROUTINE
      LOGICAL FUNCTION LDE( RI, RJ, LR )
!
!      Tests if two arrays are identical.
!
!      Auxiliary routine for test program for Level 3 Blas.
!
!      -- Written on 8-February-1989.
!      Jack Dongarra, Argonne National Laboratory.
!      Iain Duff, AERE Harwell.
!      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!      Sven Hammarling, Numerical Algorithms Group Ltd.
!
!      .. Scalar Arguments ..
      INTEGER          LR
!      .. Array Arguments ..
      DOUBLE PRECISION RI( * ), RJ( * )
!      .. Local Scalars ..
      INTEGER          I
!      .. Executable Statements ..
      DO 10 I = 1, LR
          IF( RI( I ).NE.RJ( I ) )&
              GO TO 20
      10 CONTINUE
      LDE = .TRUE.
      GO TO 30
      20 CONTINUE
      LDE = .FALSE.
      30 RETURN
!
!      End of LDE.
!
!      END FUNCTION
      LOGICAL FUNCTION LDERES( TYPE, UPLO, M, N, AA, AS, LDA )
!
!      Tests if selected elements in two arrays are equal.
!
!      TYPE is 'GE' or 'SY'.
!
!      Auxiliary routine for test program for Level 3 Blas.
!
!      -- Written on 8-February-1989.
!      Jack Dongarra, Argonne National Laboratory.
!      Iain Duff, AERE Harwell.
!      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!      Sven Hammarling, Numerical Algorithms Group Ltd.
!
!      .. Scalar Arguments ..
      INTEGER          LDA, M, N
      CHARACTER*1      UPLO
      CHARACTER*2      TYPE
!      .. Array Arguments ..
      DOUBLE PRECISION AA( LDA, * ), AS( LDA, * )
!      .. Local Scalars ..
      INTEGER          I, IBEG, IEND, J
      LOGICAL          UPPER
!      .. Executable Statements ..
      UPPER = UPLO.EQ.'U'
      IF( TYPE.EQ.'GE' )THEN
          DO 20 J = 1, N
              DO 10 I = M + 1, LDA
                  IF( AA( I, J ).NE.AS( I, J ) )&
                      GO TO 70
              10 CONTINUE
              20 CONTINUE
          ELSE IF( TYPE.EQ.'SY' )THEN
              DO 50 J = 1, N
                  IF( UPPER )THEN
                      IBEG = 1
                      IEND = J
                  ELSE
                      IBEG = J
                      IEND = N
                  END IF
                  DO 30 I = 1, IBEG - 1
                      IF( AA( I, J ).NE.AS( I, J ) )&
                          GO TO 70
                  30 CONTINUE
                  DO 40 I = IEND + 1, LDA
                      IF( AA( I, J ).NE.AS( I, J ) )&
                          GO TO 70
                  40 CONTINUE
                  50 CONTINUE
              END IF
          LDERES = .TRUE.
          GO TO 80
          70 CONTINUE
          LDERES = .FALSE.
          80 RETURN
!
!      End of LDERES.
!
!      END FUNCTION
      DOUBLE PRECISION FUNCTION DBEG( RESET )

```

```

!
! Generates random numbers uniformly distributed between -0.5 and 0.5.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
LOGICAL RESET
! .. Local Scalars ..
INTEGER I, IC, MI
! .. Save statement ..
SAVE I, IC, MI
! .. Executable Statements ..
IF( RESET )THEN
! Initialize local variables.
MI = 891
I = 7
IC = 0
RESET = .FALSE.
END IF
!
! The sequence of values of I is bounded between 1 and 999.
! If initial I = 1,2,3,6,7 or 9, the period will be 50.
! If initial I = 4 or 8, the period will be 25.
! If initial I = 5, the period will be 10.
! IC is used to break up the period by skipping 1 value of I in 6.
!
! IC = IC + 1
10 I = I*MI
I = I - 1000*( I/1000 )
IF( IC.GE.5 )THEN
IC = 0
GO TO 10
END IF
DBEG = ( I - 500 )/1001.0D0
RETURN
!
! End of DBEG.
!
! END FUNCTION
DOUBLE PRECISION FUNCTION DDIFF( X, Y )
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
DOUBLE PRECISION X, Y
! .. Executable Statements ..
DDIFF = X - Y
RETURN
!
! End of DDIFF.
!
! END FUNCTION
SUBROUTINE CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
! Tests whether XERBLA has detected an error when it should.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
INTEGER INFOT, NOUT
LOGICAL LERR, OK
CHARACTER*6 SRNAMT
! .. Executable Statements ..
IF( .NOT.LERR )THEN
WRITE( NOUT, FMT = 9999 )INFOT, SRNAMT
OK = .FALSE.
END IF
LERR = .FALSE.
RETURN
!
9999 FORMAT( ' ***** ILLEGAL VALUE OF PARAMETER NUMBER ', I2, ' NOT D', &
'ETECTED BY ', A6, ' *****' )
!
! End of CHKXER.
!
! END SUBROUTINE
SUBROUTINE XERBLA( SRNAME, INFO )
!
! This is a special version of XERBLA to be used only as part of
! the test program for testing error exits from the Level 3 BLAS
! routines.
!
! XERBLA is an error handler for the Level 3 BLAS routines.
!
! It is called by the Level 3 BLAS routines if an input parameter is
! invalid.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..

```

```

        INTEGER          INFO
        CHARACTER*6      SRNAME
!      .. Scalars in Common ..
        INTEGER          INFOT, NOUT
        LOGICAL          LERR, OK
        CHARACTER*6      SRNAMT
!      .. Common blocks ..
        COMMON           /INFOC/INFOT, NOUT, OK, LERR
        COMMON           /SRNAMC/SRNAMT
!      .. Executable Statements ..
        LERR = .TRUE.
        IF( INFO.NE.INFOT )THEN
            IF( INFOT.NE.0 )THEN
                WRITE( NOUT, FMT = 9999 )INFO, INFOT
            ELSE
                WRITE( NOUT, FMT = 9997 )INFO
            END IF
            OK = .FALSE.
        END IF
        IF( SRNAME.NE.SRNAMT )THEN
            WRITE( NOUT, FMT = 9998 )SRNAME, SRNAMT
            OK = .FALSE.
        END IF
        RETURN
!
9999 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, ' INSTEAD', &
' OF ', I2, ' *****' )
9998 FORMAT( ' ***** XERBLA WAS CALLED WITH SRNAME = ', A6, ' INSTE', &
'AD OF ', A6, ' *****' )
9997 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, &
' *****' )
!
!      End of XERBLA
!
!      END SUBROUTINE
!!> \brief \b DGEMM
!!!!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
!      .. Scalar Arguments ..
!      DOUBLE PRECISION ALPHA,BETA
!      INTEGER K,LDA,LDB,LDC,M,N
!      CHARACTER TRANSA,TRANSB
!
!      .. Array Arguments ..
!      DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
!
!      ..
!
!
!> \par Purpose:
!      =====
!>
!> \verbatim
!>
!> DGEMM performs one of the matrix-matrix operations
!>
!>   C := alpha*op( A )*op( B ) + beta*C,
!>
!> where op( X ) is one of
!>
!>   op( X ) = X   or   op( X ) = X**T,
!>
!> alpha and beta are scalars, and A, B and C are matrices, with op( A )
!> an m by k matrix, op( B ) a k by n matrix and C an m by n matrix.
!> \endverbatim
!
!      Arguments:
!      =====
!> \param[in] TRANSA
!> \verbatim
!>   TRANSA is CHARACTER*1
!>   On entry, TRANSA specifies the form of op( A ) to be used in
!>   the matrix multiplication as follows:
!>
!>       TRANSA = 'N' or 'n',   op( A ) = A.
!>
!>       TRANSA = 'T' or 't',   op( A ) = A**T.
!>
!>       TRANSA = 'C' or 'c',   op( A ) = A**T.
!> \endverbatim
!> \param[in] TRANSB
!> \verbatim
!>   TRANSB is CHARACTER*1
!>   On entry, TRANSB specifies the form of op( B ) to be used in
!>   the matrix multiplication as follows:
!>
!>       TRANSB = 'N' or 'n',   op( B ) = B.
!>
!>       TRANSB = 'T' or 't',   op( B ) = B**T.
!>
!>       TRANSB = 'C' or 'c',   op( B ) = B**T.
!> \endverbatim
!> \param[in] M
!> \verbatim
!>   M is INTEGER
!>   On entry, M specifies the number of rows of the matrix
!>   op( A ) and of the matrix C. M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>   N is INTEGER
!>   On entry, N specifies the number of columns of the matrix

```

```

!>      op( B ) and the number of columns of the matrix C. N must be
!>      at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>      K is INTEGER
!>      On entry, K specifies the number of columns of the matrix
!>      op( A ) and the number of rows of the matrix op( B ). K must
!>      be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is DOUBLE PRECISION.
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>      A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is
!>      k when TRANSA = 'N' or 'n', and is m otherwise.
!>      Before entry with TRANSA = 'N' or 'n', the leading m by k
!>      part of the array A must contain the matrix A, otherwise
!>      the leading k by m part of the array A must contain the
!>      matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. When TRANSA = 'N' or 'n' then
!>      LDA must be at least max( 1, m ), otherwise LDA must be at
!>      least max( 1, k ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!>      B is DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is
!>      n when TRANSB = 'N' or 'n', and is k otherwise.
!>      Before entry with TRANSB = 'N' or 'n', the leading k by n
!>      part of the array B must contain the matrix B, otherwise
!>      the leading n by k part of the array B must contain the
!>      matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>      LDB is INTEGER
!>      On entry, LDB specifies the first dimension of B as declared
!>      in the calling (sub) program. When TRANSB = 'N' or 'n' then
!>      LDB must be at least max( 1, k ), otherwise LDB must be at
!>      least max( 1, n ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is DOUBLE PRECISION.
!>      On entry, BETA specifies the scalar beta. When BETA is
!>      supplied as zero then C need not be set on input.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>      C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
!>      Before entry, the leading m by n part of the array C must
!>      contain the matrix C, except when beta is zero, in which
!>      case C need not be set on entry.
!>      On exit, the array C is overwritten by the m by n matrix
!>      ( alpha*op( A )*op( B ) + beta*C ).
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>      LDC is INTEGER
!>      On entry, LDC specifies the first dimension of C as declared
!>      in the calling (sub) program. LDC must be at least
!>      max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup double_blas_level3
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>      Level 3 Blas routine.
!>
!>      -- Written on 8-February-1989.
!>      Jack Dongarra, Argonne National Laboratory.
!>      Iain Duff, AERE Harwell.
!>      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>      Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!> =====
!>      SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!>
!>      -- Reference BLAS level3 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!>      November 2011

```

```

!
! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA,BETA
INTEGER K,LDA,LDB,LDC,M,N
CHARACTER TRANSA,TRANSB
!
! ..
! .. Array Arguments ..
DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
!
! ..
!
=====
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
INTRINSIC MAX
!
! ..
! .. Local Scalars ..
DOUBLE PRECISION TEMP
INTEGER I,INFO,J,L,NCOLA,NROWA,NROWB
LOGICAL NOTA,NOTB
!
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! ..
!
! Set NOTA and NOTB as true if A and B respectively are not
! transposed and set NROWA, NCOLA and NROWB as the number of rows
! and columns of A and the number of rows of B respectively.
!
NOTA = LSAME(TRANSA,'N')
NOTB = LSAME(TRANSB,'N')
IF (NOTA) THEN
    NROWA = M
    NCOLA = K
ELSE
    NROWA = K
    NCOLA = M
END IF
IF (NOTB) THEN
    NROWB = K
ELSE
    NROWB = N
END IF
!
! Test the input parameters.
!
INFO = 0
IF ((.NOT.NOTA) .AND. (.NOT.LSAME(TRANSA,'C'))) .AND.&
    (.NOT.LSAME(TRANSA,'T')) THEN
    INFO = 1
ELSE IF ((.NOT.NOTB) .AND. (.NOT.LSAME(TRANSB,'C'))) .AND.&
    (.NOT.LSAME(TRANSB,'T')) THEN
    INFO = 2
ELSE IF (M.LT.0) THEN
    INFO = 3
ELSE IF (N.LT.0) THEN
    INFO = 4
ELSE IF (K.LT.0) THEN
    INFO = 5
ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
    INFO = 8
ELSE IF (LDB.LT.MAX(1,NROWB)) THEN
    INFO = 10
ELSE IF (LDC.LT.MAX(1,M)) THEN
    INFO = 13
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('DGEMM ',INFO)
    RETURN
END IF
!
! Quick return if possible.
!
IF ((M.EQ.0) .OR. (N.EQ.0) .OR.&
    ((ALPHA.EQ.ZERO) .OR. (K.EQ.0)) .AND. (BETA.EQ.ONE))) RETURN
!
! And if alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
    IF (BETA.EQ.ZERO) THEN
        DO 20 J = 1,N
            DO 10 I = 1,M
                C(I,J) = ZERO
            CONTINUE
        CONTINUE
    ELSE
        DO 40 J = 1,N
            DO 30 I = 1,M
                C(I,J) = BETA*C(I,J)
            CONTINUE
        CONTINUE
    END IF
    RETURN
END IF
!
! Start the operations.
!
IF (NOTB) THEN
    IF (NOTA) THEN
!
!         Form C := alpha*A*B + beta*C.
!
        DO 90 J = 1,N
            IF (BETA.EQ.ZERO) THEN
                DO 50 I = 1,M
                    C(I,J) = ZERO
                CONTINUE

```

```

        ELSE IF (BETA.NE.ONE) THEN
            DO 60 I = 1,M
                C(I,J) = BETA*C(I,J)
            CONTINUE
        END IF
        DO 80 L = 1,K
            IF (B(L,J).NE.ZERO) THEN
                TEMP = ALPHA*B(L,J)
                DO 70 I = 1,M
                    C(I,J) = C(I,J) + TEMP*A(I,L)
                CONTINUE
            END IF
        CONTINUE
    80 CONTINUE
    90 CONTINUE
ELSE
    !
    ! Form C := alpha*A**T*B + beta*C
    !
    DO 120 J = 1,N
        DO 110 I = 1,M
            TEMP = ZERO
            DO 100 L = 1,K
                TEMP = TEMP + A(L,I)*B(L,J)
            CONTINUE
            IF (BETA.EQ.ZERO) THEN
                C(I,J) = ALPHA*TEMP
            ELSE
                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
            END IF
        CONTINUE
    110 CONTINUE
    120 CONTINUE
    END IF
ELSE
    IF (NOTA) THEN
        !
        ! Form C := alpha*A*B**T + beta*C
        !
        DO 170 J = 1,N
            IF (BETA.EQ.ZERO) THEN
                DO 130 I = 1,M
                    C(I,J) = ZERO
                CONTINUE
            ELSE IF (BETA.NE.ONE) THEN
                DO 140 I = 1,M
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            END IF
            DO 160 L = 1,K
                IF (B(J,L).NE.ZERO) THEN
                    TEMP = ALPHA*B(J,L)
                    DO 150 I = 1,M
                        C(I,J) = C(I,J) + TEMP*A(I,L)
                    CONTINUE
                END IF
            CONTINUE
        CONTINUE
    160 CONTINUE
    170 CONTINUE
    ELSE
        !
        ! Form C := alpha*A**T*B**T + beta*C
        !
        DO 200 J = 1,N
            DO 190 I = 1,M
                TEMP = ZERO
                DO 180 L = 1,K
                    TEMP = TEMP + A(L,I)*B(J,L)
                CONTINUE
                IF (BETA.EQ.ZERO) THEN
                    C(I,J) = ALPHA*TEMP
                ELSE
                    C(I,J) = ALPHA*TEMP + BETA*C(I,J)
                END IF
            CONTINUE
        CONTINUE
    190 CONTINUE
    200 CONTINUE
    END IF
    END IF
END IF
RETURN
!
! End of DGEMM .
!
END SUBROUTINE

!> \brief \b DSYMM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER LDA,LDB,LDC,M,N
! CHARACTER SIDE,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DSYMM performs one of the matrix-matrix operations
!>
!> C := alpha*A*B + beta*C,
!>
!> or

```

```

!>
!>   C := alpha*B*A + beta*C,
!>
!> where alpha and beta are scalars, A is a symmetric matrix and B and
!> C are m by n matrices.
!> \endverbatim
!>
!> Arguments:
!> =====
!>
!> \param[in] SIDE
!> \verbatim
!> SIDE is CHARACTER*1
!> On entry, SIDE specifies whether the symmetric matrix A
!> appears on the left or right in the operation as follows:
!>
!>     SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,
!>
!>     SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the symmetric matrix A is to be
!> referenced as follows:
!>
!>     UPLO = 'U' or 'u'   Only the upper triangular part of the
!>                           symmetric matrix is to be referenced.
!>
!>     UPLO = 'L' or 'l'   Only the lower triangular part of the
!>                           symmetric matrix is to be referenced.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix C.
!> M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix C.
!> N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION.
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!> A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is
!> m when SIDE = 'L' or 'l' and is n otherwise.
!> Before entry with SIDE = 'L' or 'l', the m by m part of
!> the array A must contain the symmetric matrix, such that
!> when UPLO = 'U' or 'u', the leading m by m upper triangular
!> part of the array A must contain the upper triangular part
!> of the symmetric matrix and the strictly lower triangular
!> part of A is not referenced, and when UPLO = 'L' or 'l',
!> the leading m by m lower triangular part of the array A
!> must contain the lower triangular part of the symmetric
!> matrix and the strictly upper triangular part of A is not
!> referenced.
!> Before entry with SIDE = 'R' or 'r', the n by n part of
!> the array A must contain the symmetric matrix, such that
!> when UPLO = 'U' or 'u', the leading n by n upper triangular
!> part of the array A must contain the upper triangular part
!> of the symmetric matrix and the strictly lower triangular
!> part of A is not referenced, and when UPLO = 'L' or 'l',
!> the leading n by n lower triangular part of the array A
!> must contain the lower triangular part of the symmetric
!> matrix and the strictly upper triangular part of A is not
!> referenced.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. When SIDE = 'L' or 'l' then
!> LDA must be at least max( 1, m ), otherwise LDA must be at
!> least max( 1, n ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!> B is DOUBLE PRECISION array of DIMENSION ( LDB, n ).
!> Before entry, the leading m by n part of the array B must
!> contain the matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!> LDB is INTEGER
!> On entry, LDB specifies the first dimension of B as declared
!> in the calling (sub) program. LDB must be at least
!> max( 1, m ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!> BETA is DOUBLE PRECISION.
!> On entry, BETA specifies the scalar beta. When BETA is
!> supplied as zero then C need not be set on input.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim

```

```

!>      C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
!>      Before entry, the leading m by n part of the array C must
!>      contain the matrix C, except when beta is zero, in which
!>      case C need not be set on entry.
!>      On exit, the array C is overwritten by the m by n updated
!>      matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>      LDC is INTEGER
!>      On entry, LDC specifies the first dimension of C as declared
!>      in the calling (sub) program. LDC must be at least
!>      max( 1, m ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level3
!
!> \par Further Details:
! =====
!> \verbatim
!>
!>      Level 3 Blas routine.
!>
!>      -- Written on 8-February-1989.
!>      Jack Dongarra, Argonne National Laboratory.
!>      Iain Duff, AERE Harwell.
!>      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>      Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
! =====
SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA,BETA
INTEGER LDA,LDB,LDC,M,N
CHARACTER SIDE,UPLO
!
! ..
! .. Array Arguments ..
DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
!
! ..
! =====
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
INTRINSIC MAX
!
! ..
! .. Local Scalars ..
DOUBLE PRECISION TEMP1,TEMP2
INTEGER I,INFO,J,K,NROWA
LOGICAL UPPER
!
! ..
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! ..
!
! Set NROWA as the number of rows of A.
!
! IF (LSAME(SIDE,'L')) THEN
!     NROWA = M
! ELSE
!     NROWA = N
! END IF
UPPER = LSAME(UPLO,'U')
!
! Test the input parameters.
!
! INFO = 0
! IF ((.NOT.LSAME(SIDE,'L')) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!     INFO = 1
! ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!     INFO = 2
! ELSE IF (M.LT.0) THEN
!     INFO = 3
! ELSE IF (N.LT.0) THEN
!     INFO = 4
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!     INFO = 7
! ELSE IF (LDB.LT.MAX(1,M)) THEN
!     INFO = 9
! ELSE IF (LDC.LT.MAX(1,M)) THEN
!     INFO = 12
! END IF
! IF (INFO.NE.0) THEN
!     CALL XERBLA('DSYMM ',INFO)
!     RETURN
! END IF
!
! Quick return if possible.

```

```

!
! IF ((M.EQ.0) .OR. (N.EQ.0) .OR. &
!   (ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN
!
! And when alpha.eq.zero.
!
! IF (ALPHA.EQ.ZERO) THEN
!   IF (BETA.EQ.ZERO) THEN
!     DO 20 J = 1,N
!       DO 10 I = 1,M
!         C(I,J) = ZERO
!10      CONTINUE
!20    CONTINUE
!   ELSE
!     DO 40 J = 1,N
!       DO 30 I = 1,M
!         C(I,J) = BETA*C(I,J)
!30      CONTINUE
!40    CONTINUE
!   END IF
!   RETURN
! END IF
!
! Start the operations.
!
! IF (LSAME(SIDE,'L')) THEN
!
!   Form C := alpha*A*B + beta*C.
!
!   IF (UPPER) THEN
!     DO 70 J = 1,N
!       DO 60 I = 1,M
!         TEMP1 = ALPHA*B(I,J)
!         TEMP2 = ZERO
!         DO 50 K = 1, I - 1
!           C(K,J) = C(K,J) + TEMP1*A(K,I)
!           TEMP2 = TEMP2 + B(K,J)*A(K,I)
!50        CONTINUE
!         IF (BETA.EQ.ZERO) THEN
!           C(I,J) = TEMP1*A(I,I) + ALPHA*TEMP2
!         ELSE
!           C(I,J) = BETA*C(I,J) + TEMP1*A(I,I) + &
!             ALPHA*TEMP2
!         END IF
!60      CONTINUE
!70    CONTINUE
!   ELSE
!     DO 100 J = 1,N
!       DO 90 I = M, 1, -1
!         TEMP1 = ALPHA*B(I,J)
!         TEMP2 = ZERO
!         DO 80 K = I + 1, M
!           C(K,J) = C(K,J) + TEMP1*A(K,I)
!           TEMP2 = TEMP2 + B(K,J)*A(K,I)
!80        CONTINUE
!         IF (BETA.EQ.ZERO) THEN
!           C(I,J) = TEMP1*A(I,I) + ALPHA*TEMP2
!         ELSE
!           C(I,J) = BETA*C(I,J) + TEMP1*A(I,I) + &
!             ALPHA*TEMP2
!         END IF
!90      CONTINUE
!100   CONTINUE
!   END IF
! ELSE
!
!   Form C := alpha*B*A + beta*C.
!
!   DO 170 J = 1,N
!     TEMP1 = ALPHA*A(J,J)
!     IF (BETA.EQ.ZERO) THEN
!       DO 110 I = 1,M
!         C(I,J) = TEMP1*B(I,J)
!110      CONTINUE
!     ELSE
!       DO 120 I = 1,M
!         C(I,J) = BETA*C(I,J) + TEMP1*B(I,J)
!120      CONTINUE
!     END IF
!     DO 140 K = 1, J - 1
!       IF (UPPER) THEN
!         TEMP1 = ALPHA*A(K,J)
!       ELSE
!         TEMP1 = ALPHA*A(J,K)
!       END IF
!       DO 130 I = 1,M
!         C(I,J) = C(I,J) + TEMP1*B(I,K)
!130      CONTINUE
!140    CONTINUE
!     DO 160 K = J + 1, N
!       IF (UPPER) THEN
!         TEMP1 = ALPHA*A(J,K)
!       ELSE
!         TEMP1 = ALPHA*A(K,J)
!       END IF
!       DO 150 I = 1,M
!         C(I,J) = C(I,J) + TEMP1*B(I,K)
!150      CONTINUE
!160    CONTINUE
!170  CONTINUE
!   END IF
!
! RETURN
!
! End of DSYMM .
!
! END SUBROUTINE
!> \brief \b DTRMM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!

```

```

! Definition:
! =====
!
! SUBROUTINE DTRMM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),B(LDB,*)
! ..
!
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DTRMM performs one of the matrix-matrix operations
!>
!>   B := alpha*op( A )*B,   or   B := alpha*B*op( A ),
!>
!> where alpha is a scalar, B is an m by n matrix, A is a unit, or
!> non-unit, upper or lower triangular matrix and op( A ) is one of
!>
!>   op( A ) = A   or   op( A ) = A**T.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] SIDE
!> \verbatim
!>
!>   SIDE is CHARACTER*1
!>   On entry, SIDE specifies whether op( A ) multiplies B from
!>   the left or right as follows:
!>
!>       SIDE = 'L' or 'l'   B := alpha*op( A )*B.
!>
!>       SIDE = 'R' or 'r'   B := alpha*B*op( A ).
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!>
!>   UPLO is CHARACTER*1
!>   On entry, UPLO specifies whether the matrix A is an upper or
!>   lower triangular matrix as follows:
!>
!>       UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>       UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANSA
!> \verbatim
!>
!>   TRANSA is CHARACTER*1
!>   On entry, TRANSA specifies the form of op( A ) to be used in
!>   the matrix multiplication as follows:
!>
!>       TRANSA = 'N' or 'n'   op( A ) = A.
!>
!>       TRANSA = 'T' or 't'   op( A ) = A**T.
!>
!>       TRANSA = 'C' or 'c'   op( A ) = A**T.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>
!>   DIAG is CHARACTER*1
!>   On entry, DIAG specifies whether or not A is unit triangular
!>   as follows:
!>
!>       DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>       DIAG = 'N' or 'n'   A is not assumed to be unit
!>                           triangular.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!>
!>   M is INTEGER
!>   On entry, M specifies the number of rows of B. M must be at
!>   least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>
!>   N is INTEGER
!>   On entry, N specifies the number of columns of B. N must be
!>   at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>
!>   ALPHA is DOUBLE PRECISION.
!>   On entry, ALPHA specifies the scalar alpha. When alpha is
!>   zero then A is not referenced and B need not be set before
!>   entry.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>
!>   A is DOUBLE PRECISION array of DIMENSION ( LDA, k ), where k is m
!>   when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'.
!>   Before entry with UPLO = 'U' or 'u', the leading k by k
!>   upper triangular part of the array A must contain the upper
!>   triangular matrix and the strictly lower triangular part of
!>   A is not referenced.
!>   Before entry with UPLO = 'L' or 'l', the leading k by k
!>   lower triangular part of the array A must contain the lower
!>   triangular matrix and the strictly upper triangular part of
!>   A is not referenced.

```

```

!>      Note that when DIAG = 'U' or 'u', the diagonal elements of
!>      A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. When SIDE = 'L' or 'l' then
!>      LDA must be at least max( 1, m ), when SIDE = 'R' or 'r'
!>      then LDA must be at least max( 1, n ).
!> \endverbatim
!>
!> \param[in,out] B
!> \verbatim
!>      B is DOUBLE PRECISION array of DIMENSION ( LDB, n ).
!>      Before entry, the leading m by n part of the array B must
!>      contain the matrix B, and on exit is overwritten by the
!>      transformed matrix.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>      LDB is INTEGER
!>      On entry, LDB specifies the first dimension of B as declared
!>      in the calling (sub) program. LDB must be at least
!>      max( 1, m ).
!> \endverbatim
!
!   Authors:
!   =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level3
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>      Level 3 Blas routine.
!>
!>      -- Written on 8-February-1989.
!>      Jack Dongarra, Argonne National Laboratory.
!>      Iain Duff, AERE Harwell.
!>      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>      Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
!   =====
!   SUBROUTINE DTRMM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
!   -- Reference BLAS level3 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!   November 2011
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA
!   INTEGER LDA,LDB,M,N
!   CHARACTER DIAG,SIDE,TRANSA,UPLO
!
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION A(LDA,*),B(LDB,*)
!
!   ..
!
!   =====
!
!   .. External Functions ..
!   LOGICAL LSAME
!   EXTERNAL LSAME
!
!   ..
!   .. External Subroutines ..
!   EXTERNAL XERBLA
!
!   ..
!   .. Intrinsic Functions ..
!   INTRINSIC MAX
!
!   ..
!   .. Local Scalars ..
!   DOUBLE PRECISION TEMP
!   INTEGER I,INFO,J,K,NROWA
!   LOGICAL LSIDE,NOUNIT,UPPER
!
!   ..
!   .. Parameters ..
!   DOUBLE PRECISION ONE,ZERO
!   PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
!   ..
!
!   Test the input parameters.
!
!   LSIDE = LSAME(SIDE,'L')
!   IF (LSIDE) THEN
!       NROWA = M
!   ELSE
!       NROWA = N
!   END IF
!   NOUNIT = LSAME(DIAG,'N')
!   UPPER = LSAME(UPLO,'U')
!
!   INFO = 0
!   IF ((.NOT.LSIDE) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!       INFO = 1
!   ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!       INFO = 2
!   ELSE IF ((.NOT.LSAME(TRANSA,'N')) .AND.&
!            (.NOT.LSAME(TRANSA,'T')) .AND.&
!            (.NOT.LSAME(TRANSA,'C'))) THEN
!       INFO = 3

```

```

ELSE IF ((.NOT.LSAME(DIAG,'U')) .AND. (.NOT.LSAME(DIAG,'N'))) THEN
  INFO = 4
ELSE IF (M.LT.0) THEN
  INFO = 5
ELSE IF (N.LT.0) THEN
  INFO = 6
ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
  INFO = 9
ELSE IF (LDB.LT.MAX(1,M)) THEN
  INFO = 11
END IF
IF (INFO.NE.0) THEN
  CALL XERBLA('DTRMM ',INFO)
  RETURN
END IF
!
! Quick return if possible.
!
IF (M.EQ.0 .OR. N.EQ.0) RETURN
!
! And when alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
  DO 20 J = 1,N
    DO 10 I = 1,M
      B(I,J) = ZERO
    CONTINUE
  CONTINUE
  RETURN
END IF
!
! Start the operations.
!
IF (LSIDE) THEN
  IF (LSAME(TRANSA,'N')) THEN
    !
    ! Form B := alpha*A*B.
    !
    IF (UPPER) THEN
      DO 50 J = 1,N
        DO 40 K = 1,M
          IF (B(K,J).NE.ZERO) THEN
            TEMP = ALPHA*B(K,J)
            DO 30 I = 1,K - 1
              B(I,J) = B(I,J) + TEMP*A(I,K)
            CONTINUE
            IF (NOUNIT) TEMP = TEMP*A(K,K)
            B(K,J) = TEMP
          END IF
        CONTINUE
      CONTINUE
    ELSE
      DO 80 J = 1,N
        DO 70 K = M,1,-1
          IF (B(K,J).NE.ZERO) THEN
            TEMP = ALPHA*B(K,J)
            B(K,J) = TEMP
            IF (NOUNIT) B(K,J) = B(K,J)*A(K,K)
            DO 60 I = K + 1,M
              B(I,J) = B(I,J) + TEMP*A(I,K)
            CONTINUE
          END IF
        CONTINUE
      CONTINUE
    END IF
  ELSE
    !
    ! Form B := alpha*A**T*B.
    !
    IF (UPPER) THEN
      DO 110 J = 1,N
        DO 100 I = M,1,-1
          TEMP = B(I,J)
          IF (NOUNIT) TEMP = TEMP*A(I,I)
          DO 90 K = 1,I - 1
            TEMP = TEMP + A(K,I)*B(K,J)
          CONTINUE
          B(I,J) = ALPHA*TEMP
        CONTINUE
      CONTINUE
    ELSE
      DO 140 J = 1,N
        DO 130 I = 1,M
          TEMP = B(I,J)
          IF (NOUNIT) TEMP = TEMP*A(I,I)
          DO 120 K = I + 1,M
            TEMP = TEMP + A(K,I)*B(K,J)
          CONTINUE
          B(I,J) = ALPHA*TEMP
        CONTINUE
      CONTINUE
    END IF
  END IF
ELSE
  IF (LSAME(TRANSA,'N')) THEN
    !
    ! Form B := alpha*B*A.
    !
    IF (UPPER) THEN
      DO 180 J = N,1,-1
        TEMP = ALPHA
        IF (NOUNIT) TEMP = TEMP*A(J,J)
        DO 150 I = 1,M
          B(I,J) = TEMP*B(I,J)
        CONTINUE
        DO 170 K = 1,J - 1
          IF (A(K,J).NE.ZERO) THEN
            TEMP = ALPHA*A(K,J)
            DO 160 I = 1,M
              B(I,J) = B(I,J) + TEMP*B(I,K)
            CONTINUE
          END IF
        CONTINUE
      CONTINUE
    END IF
  CONTINUE

```

```

        ELSE
            DO 220 J = 1,N
                TEMP = ALPHA
                IF (NOUNIT) TEMP = TEMP*A(J,J)
                DO 190 I = 1,M
                    B(I,J) = TEMP*B(I,J)
190                CONTINUE
                DO 210 K = J + 1,N
                    IF (A(K,J).NE.ZERO) THEN
                        TEMP = ALPHA*A(K,J)
                        DO 200 I = 1,M
                            B(I,J) = B(I,J) + TEMP*B(I,K)
200                        CONTINUE
                    END IF
                CONTINUE
210            CONTINUE
220        END IF
    ELSE
        Form B := alpha*B*A**T.
!
!
        IF (UPPER) THEN
            DO 260 K = 1,N
                DO 240 J = 1,K - 1
                    IF (A(J,K).NE.ZERO) THEN
                        TEMP = ALPHA*A(J,K)
                        DO 230 I = 1,M
                            B(I,J) = B(I,J) + TEMP*B(I,K)
230                        CONTINUE
                    END IF
                CONTINUE
240            TEMP = ALPHA
            IF (NOUNIT) TEMP = TEMP*A(K,K)
            IF (TEMP.NE.ONE) THEN
                DO 250 I = 1,M
                    B(I,K) = TEMP*B(I,K)
250                CONTINUE
            END IF
260        CONTINUE
    ELSE
        DO 300 K = N,1,-1
            DO 280 J = K + 1,N
                IF (A(J,K).NE.ZERO) THEN
                    TEMP = ALPHA*A(J,K)
                    DO 270 I = 1,M
                        B(I,J) = B(I,J) + TEMP*B(I,K)
270                    CONTINUE
                END IF
            CONTINUE
280        TEMP = ALPHA
        IF (NOUNIT) TEMP = TEMP*A(K,K)
        IF (TEMP.NE.ONE) THEN
            DO 290 I = 1,M
                B(I,K) = TEMP*B(I,K)
290            CONTINUE
        END IF
300    CONTINUE
    END IF
    END IF
    END IF
    RETURN
!
! End of DTRMM .
!
    END SUBROUTINE

!> \brief \b DTRSM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE DTRSM (SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPLO
!
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),B(LDB,*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> DTRSM solves one of the matrix equations
!>
!>   op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,
!>
!> where alpha is a scalar, X and B are m by n matrices, A is a unit, or
!> non-unit, upper or lower triangular matrix and op( A ) is one of
!>
!>   op( A ) = A   or   op( A ) = A**T.
!>
!> The matrix X is overwritten on B.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] SIDE
!> \verbatim
!>     SIDE is CHARACTER*1
!>     On entry, SIDE specifies whether op( A ) appears on the left
!>     or right of X as follows:

```

```

!>
!>         SIDE = 'L' or 'l'   op( A ) * X = alpha * B.
!>
!>         SIDE = 'R' or 'r'   X * op( A ) = alpha * B.
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!>         UPLO is CHARACTER*1
!>         On entry, UPLO specifies whether the matrix A is an upper or
!>         lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANSA
!> \verbatim
!>         TRANSA is CHARACTER*1
!>         On entry, TRANSA specifies the form of op( A ) to be used in
!>         the matrix multiplication as follows:
!>
!>         TRANSA = 'N' or 'n'   op( A ) = A.
!>
!>         TRANSA = 'T' or 't'   op( A ) = A**T.
!>
!>         TRANSA = 'C' or 'c'   op( A ) = A**T.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>         DIAG is CHARACTER*1
!>         On entry, DIAG specifies whether or not A is unit triangular
!>         as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!>         M is INTEGER
!>         On entry, M specifies the number of rows of B. M must be at
!>         least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>         N is INTEGER
!>         On entry, N specifies the number of columns of B. N must be
!>         at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>         ALPHA is DOUBLE PRECISION.
!>         On entry, ALPHA specifies the scalar alpha. When alpha is
!>         zero then A is not referenced and B need not be set before
!>         entry.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>         A is DOUBLE PRECISION array of DIMENSION ( LDA, k ),
!>         where k is m when SIDE = 'L' or 'l'
!>         and k is n when SIDE = 'R' or 'r'.
!>         Before entry with UPLO = 'U' or 'u', the leading k by k
!>         upper triangular part of the array A must contain the upper
!>         triangular matrix and the strictly lower triangular part of
!>         A is not referenced.
!>         Before entry with UPLO = 'L' or 'l', the leading k by k
!>         lower triangular part of the array A must contain the lower
!>         triangular matrix and the strictly upper triangular part of
!>         A is not referenced.
!>         Note that when DIAG = 'U' or 'u', the diagonal elements of
!>         A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>         LDA is INTEGER
!>         On entry, LDA specifies the first dimension of A as declared
!>         in the calling (sub) program. When SIDE = 'L' or 'l' then
!>         LDA must be at least max( 1, m ), when SIDE = 'R' or 'r'
!>         then LDA must be at least max( 1, n ).
!> \endverbatim
!>
!> \param[in,out] B
!> \verbatim
!>         B is DOUBLE PRECISION array of DIMENSION ( LDB, n ).
!>         Before entry, the leading m by n part of the array B must
!>         contain the right-hand side matrix B, and on exit is
!>         overwritten by the solution matrix X.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>         LDB is INTEGER
!>         On entry, LDB specifies the first dimension of B as declared
!>         in the calling (sub) program. LDB must be at least
!>         max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!>

```

```

!> \date November 2011
!
!> \ingroup double_blas_level3
!
!> \par Further Details:
! =====
!>
!> \verbatim
!> Level 3 Blas routine.
!>
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
! =====
!       SUBROUTINE DTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPLO
! ..
! .. Array Arguments ..
! DOUBLE PRECISION A(LDA,*),B(LDB,*)
! ..
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX
! ..
! .. Local Scalars ..
! DOUBLE PRECISION TEMP
! INTEGER I,INFO,J,K,NROWA
! LOGICAL LSIDE,NOUNIT,UPPER
! ..
! .. Parameters ..
! DOUBLE PRECISION ONE,ZERO
! PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
! ..
!
! Test the input parameters.
!
!       LSIDE = LSAME(SIDE,'L')
!       IF (LSIDE) THEN
!           NROWA = M
!       ELSE
!           NROWA = N
!       END IF
!       NOUNIT = LSAME(DIAG,'N')
!       UPPER = LSAME(UPLO,'U')
!
!       INFO = 0
!       IF ((.NOT.LSIDE) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!           INFO = 1
!       ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!           INFO = 2
!       ELSE IF ((.NOT.LSAME(TRANSA,'N')) .AND. &
!                (.NOT.LSAME(TRANSA,'T')) .AND. &
!                (.NOT.LSAME(TRANSA,'C'))) THEN
!           INFO = 3
!       ELSE IF ((.NOT.LSAME(DIAG,'U')) .AND. (.NOT.LSAME(DIAG,'N'))) THEN
!           INFO = 4
!       ELSE IF (M.LT.0) THEN
!           INFO = 5
!       ELSE IF (N.LT.0) THEN
!           INFO = 6
!       ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!           INFO = 9
!       ELSE IF (LDB.LT.MAX(1,M)) THEN
!           INFO = 11
!       END IF
!       IF (INFO.NE.0) THEN
!           CALL XERBLA('DTRSM ',INFO)
!           RETURN
!       END IF
!
!       Quick return if possible.
!
!       IF (M.EQ.0 .OR. N.EQ.0) RETURN
!
!       And when alpha.eq.zero.
!
!       IF (ALPHA.EQ.ZERO) THEN
!           DO 20 J = 1,N
!               DO 10 I = 1,M
!                   B(I,J) = ZERO
!               CONTINUE
!           CONTINUE
!           RETURN
!       END IF
!
!       Start the operations.
!
!       IF (LSIDE) THEN
!           IF (LSAME(TRANSA,'N')) THEN
!

```

```

!      Form B := alpha*inv( A )*B.
!
      IF (UPPER) THEN
        DO 60 J = 1,N
          IF (ALPHA.NE.ONE) THEN
            DO 30 I = 1,M
              B(I,J) = ALPHA*B(I,J)
            CONTINUE
          END IF
          DO 50 K = M,1,-1
            IF (B(K,J).NE.ZERO) THEN
              IF (NOUNIT) B(K,J) = B(K,J)/A(K,K)
              DO 40 I = 1,K - 1
                B(I,J) = B(I,J) - B(K,J)*A(I,K)
              CONTINUE
            END IF
          CONTINUE
        CONTINUE
      ELSE
        DO 100 J = 1,N
          IF (ALPHA.NE.ONE) THEN
            DO 70 I = 1,M
              B(I,J) = ALPHA*B(I,J)
            CONTINUE
          END IF
          DO 90 K = 1,M
            IF (B(K,J).NE.ZERO) THEN
              IF (NOUNIT) B(K,J) = B(K,J)/A(K,K)
              DO 80 I = K + 1,M
                B(I,J) = B(I,J) - B(K,J)*A(I,K)
              CONTINUE
            END IF
          CONTINUE
        CONTINUE
      END IF
    ELSE
      Form B := alpha*inv( A**T )*B.
      IF (UPPER) THEN
        DO 130 J = 1,N
          DO 120 I = 1,M
            TEMP = ALPHA*B(I,J)
            DO 110 K = 1,I - 1
              TEMP = TEMP - A(K,I)*B(K,J)
            CONTINUE
            IF (NOUNIT) TEMP = TEMP/A(I,I)
            B(I,J) = TEMP
          CONTINUE
        CONTINUE
      ELSE
        DO 160 J = 1,N
          DO 150 I = M,1,-1
            TEMP = ALPHA*B(I,J)
            DO 140 K = I + 1,M
              TEMP = TEMP - A(K,I)*B(K,J)
            CONTINUE
            IF (NOUNIT) TEMP = TEMP/A(I,I)
            B(I,J) = TEMP
          CONTINUE
        CONTINUE
      END IF
    ELSE
      IF (LSAME(TRANSA,'N')) THEN
        Form B := alpha*B*inv( A ).
        IF (UPPER) THEN
          DO 210 J = 1,N
            IF (ALPHA.NE.ONE) THEN
              DO 170 I = 1,M
                B(I,J) = ALPHA*B(I,J)
              CONTINUE
            END IF
            DO 190 K = 1,J - 1
              IF (A(K,J).NE.ZERO) THEN
                DO 180 I = 1,M
                  B(I,J) = B(I,J) - A(K,J)*B(I,K)
                CONTINUE
              END IF
            CONTINUE
            IF (NOUNIT) THEN
              TEMP = ONE/A(J,J)
              DO 200 I = 1,M
                B(I,J) = TEMP*B(I,J)
              CONTINUE
            END IF
          CONTINUE
        ELSE
          DO 260 J = N,1,-1
            IF (ALPHA.NE.ONE) THEN
              DO 220 I = 1,M
                B(I,J) = ALPHA*B(I,J)
              CONTINUE
            END IF
            DO 240 K = J + 1,N
              IF (A(K,J).NE.ZERO) THEN
                DO 230 I = 1,M
                  B(I,J) = B(I,J) - A(K,J)*B(I,K)
                CONTINUE
              END IF
            CONTINUE
            IF (NOUNIT) THEN
              TEMP = ONE/A(J,J)
              DO 250 I = 1,M
                B(I,J) = TEMP*B(I,J)
              CONTINUE
            END IF
          CONTINUE
        END IF
      ELSE
        Form B := alpha*B*inv( A**T ).

```

```

!
      IF (UPPER) THEN
        DO 310 K = N,1,-1
          IF (NOUNIT) THEN
            TEMP = ONE/A(K,K)
            DO 270 I = 1,M
              B(I,K) = TEMP*B(I,K)
270          CONTINUE
            END IF
            DO 290 J = 1,K - 1
              IF (A(J,K).NE.ZERO) THEN
                TEMP = A(J,K)
                DO 280 I = 1,M
                  B(I,J) = B(I,J) - TEMP*B(I,K)
280                CONTINUE
              END IF
            CONTINUE
            IF (ALPHA.NE.ONE) THEN
              DO 300 I = 1,M
                B(I,K) = ALPHA*B(I,K)
300              CONTINUE
            END IF
            CONTINUE
310          ELSE
            DO 360 K = 1,N
              IF (NOUNIT) THEN
                TEMP = ONE/A(K,K)
                DO 320 I = 1,M
                  B(I,K) = TEMP*B(I,K)
320                CONTINUE
              END IF
              DO 340 J = K + 1,N
                IF (A(J,K).NE.ZERO) THEN
                  TEMP = A(J,K)
                  DO 330 I = 1,M
                    B(I,J) = B(I,J) - TEMP*B(I,K)
330                  CONTINUE
                END IF
              CONTINUE
              IF (ALPHA.NE.ONE) THEN
                DO 350 I = 1,M
                  B(I,K) = ALPHA*B(I,K)
350                CONTINUE
              END IF
            CONTINUE
360          END IF
        END IF
      END IF
!
      RETURN
!
!   End of DTRSM .
!
      END SUBROUTINE

!> \brief \b DSYRK
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE DSYRK (UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA,BETA
!   INTEGER K,LDA,LDC,N
!   CHARACTER TRANS,UPLO
!   ..
!   .. Array Arguments ..
!   DOUBLE PRECISION A(LDA,*),C(LDC,*)
!   ..
!
!> \par Purpose:
!   =====
!>
!> \verbatim
!>
!> DSYRK performs one of the symmetric rank k operations
!>
!>   C := alpha*A*A**T + beta*C,
!>
!> or
!>
!>   C := alpha*A**T*A + beta*C,
!>
!> where alpha and beta are scalars, C is an n by n symmetric matrix
!> and A is an n by k matrix in the first case and a k by n matrix
!> in the second case.
!> \endverbatim
!
!   Arguments:
!   =====
!
!> \param[in] UPLO
!> \verbatim
!>
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array C is to be referenced as
!> follows:
!>
!>     UPLO = 'U' or 'u' Only the upper triangular part of C
!>                       is to be referenced.
!>
!>     UPLO = 'L' or 'l' Only the lower triangular part of C
!>                       is to be referenced.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim

```

```

!>      TRANS is CHARACTER*1
!>      On entry, TRANS specifies the operation to be performed as
!>      follows:
!>
!>      TRANS = 'N' or 'n'   C := alpha*A*A**T + beta*C.
!>
!>      TRANS = 'T' or 't'   C := alpha*A**T*A + beta*C.
!>
!>      TRANS = 'C' or 'c'   C := alpha*A**T*A + beta*C.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>      N is INTEGER
!>      On entry, N specifies the order of the matrix C. N must be
!>      at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>      K is INTEGER
!>      On entry with TRANS = 'N' or 'n', K specifies the number
!>      of columns of the matrix A, and on entry with
!>      TRANS = 'T' or 't' or 'C' or 'c', K specifies the number
!>      of rows of the matrix A. K must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is DOUBLE PRECISION.
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>      A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is
!>      k when TRANS = 'N' or 'n', and is n otherwise.
!>      Before entry with TRANS = 'N' or 'n', the leading n by k
!>      part of the array A must contain the matrix A, otherwise
!>      the leading k by n part of the array A must contain the
!>      matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. When TRANS = 'N' or 'n'
!>      then LDA must be at least max( 1, n ), otherwise LDA must
!>      be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is DOUBLE PRECISION.
!>      On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>      C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
!>      Before entry with UPLO = 'U' or 'u', the leading n by n
!>      upper triangular part of the array C must contain the upper
!>      triangular part of the symmetric matrix and the strictly
!>      lower triangular part of C is not referenced. On exit, the
!>      upper triangular part of the array C is overwritten by the
!>      upper triangular part of the updated matrix.
!>      Before entry with UPLO = 'L' or 'l', the leading n by n
!>      lower triangular part of the array C must contain the lower
!>      triangular part of the symmetric matrix and the strictly
!>      upper triangular part of C is not referenced. On exit, the
!>      lower triangular part of the array C is overwritten by the
!>      lower triangular part of the updated matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>      LDC is INTEGER
!>      On entry, LDC specifies the first dimension of C as declared
!>      in the calling (sub) program. LDC must be at least
!>      max( 1, n ).
!> \endverbatim
!>
!> !
!> ! Authors:
!> ! =====
!> !
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!> !
!> \date November 2011
!> !
!> \ingroup double_blas_level3
!> !
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!> !
!> =====
!> SUBROUTINE DSYRK(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!> !
!> -- Reference BLAS level3 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --

```

```

! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA,BETA
INTEGER K,LDA,LDC,N
CHARACTER TRANS,UPLO
!
! .. Array Arguments ..
DOUBLE PRECISION A(LDA,*),C(LDC,*)
!
! ..
!
=====
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
INTRINSIC MAX
!
! ..
! .. Local Scalars ..
DOUBLE PRECISION TEMP
INTEGER I,INFO,J,L,NROWA
LOGICAL UPPER
!
! ..
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! ..
!
Test the input parameters.
!
IF (LSAME(TRANS,'N')) THEN
    NROWA = N
ELSE
    NROWA = K
END IF
UPPER = LSAME(UPLO,'U')
!
INFO = 0
IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
    INFO = 1
ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
(.NOT.LSAME(TRANS,'T')) .AND. &
(.NOT.LSAME(TRANS,'C'))) THEN
    INFO = 2
ELSE IF (N.LT.0) THEN
    INFO = 3
ELSE IF (K.LT.0) THEN
    INFO = 4
ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
    INFO = 7
ELSE IF (LDC.LT.MAX(1,N)) THEN
    INFO = 10
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('DSYRK ',INFO)
    RETURN
END IF
!
Quick return if possible.
!
IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).OR.&
(K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN
!
And when alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
    IF (UPPER) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 20 J = 1,N
                DO 10 I = 1,J
                    C(I,J) = ZERO
10                CONTINUE
20            CONTINUE
        ELSE
            DO 40 J = 1,N
                DO 30 I = 1,J
                    C(I,J) = BETA*C(I,J)
30                CONTINUE
40            CONTINUE
        END IF
    ELSE
        IF (BETA.EQ.ZERO) THEN
            DO 60 J = 1,N
                DO 50 I = J,N
                    C(I,J) = ZERO
50                CONTINUE
60            CONTINUE
        ELSE
            DO 80 J = 1,N
                DO 70 I = J,N
                    C(I,J) = BETA*C(I,J)
70                CONTINUE
80            CONTINUE
        END IF
    END IF
    RETURN
END IF
!
Start the operations.
!
IF (LSAME(TRANS,'N')) THEN
!
    Form C := alpha*A*A**T + beta*C.
!
    IF (UPPER) THEN
        DO 130 J = 1,N
            IF (BETA.EQ.ZERO) THEN
                DO 90 I = 1,J

```



```

!> matrices in the second case.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the upper or lower
!>     triangular part of the array C is to be referenced as
!>     follows:
!>
!>         UPLO = 'U' or 'u'    Only the upper triangular part of C
!>         is to be referenced.
!>
!>         UPLO = 'L' or 'l'    Only the lower triangular part of C
!>         is to be referenced.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the operation to be performed as
!>     follows:
!>
!>         TRANS = 'N' or 'n'    C := alpha*A*B**T + alpha*B*A**T +
!>                                beta*C.
!>
!>         TRANS = 'T' or 't'    C := alpha*A**T*B + alpha*B**T*A +
!>                                beta*C.
!>
!>         TRANS = 'C' or 'c'    C := alpha*A**T*B + alpha*B**T*A +
!>                                beta*C.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix C. N must be
!>     at least zero.
!> \endverbatim
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with TRANS = 'N' or 'n', K specifies the number
!>     of columns of the matrices A and B, and on entry with
!>     TRANS = 'T' or 't' or 'C' or 'c', K specifies the number
!>     of rows of the matrices A and B. K must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is DOUBLE PRECISION.
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] A
!> \verbatim
!>     A is DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array A must contain the matrix A, otherwise
!>     the leading k by n part of the array A must contain the
!>     matrix A.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDA must be at least max( 1, n ), otherwise LDA must
!>     be at least max( 1, k ).
!> \endverbatim
!> \param[in] B
!> \verbatim
!>     B is DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array B must contain the matrix B, otherwise
!>     the leading k by n part of the array B must contain the
!>     matrix B.
!> \endverbatim
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDB must be at least max( 1, n ), otherwise LDB must
!>     be at least max( 1, k ).
!> \endverbatim
!> \param[in] BETA
!> \verbatim
!>     BETA is DOUBLE PRECISION.
!>     On entry, BETA specifies the scalar beta.
!> \endverbatim
!> \param[in,out] C
!> \verbatim
!>     C is DOUBLE PRECISION array of DIMENSION ( LDC, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array C must contain the upper
!>     triangular part of the symmetric matrix and the strictly
!>     lower triangular part of C is not referenced. On exit, the
!>     upper triangular part of the array C is overwritten by the
!>     upper triangular part of the updated matrix.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array C must contain the lower
!>     triangular part of the symmetric matrix and the strictly

```

```

!>         upper triangular part of C is not referenced. On exit, the
!>         lower triangular part of the array C is overwritten by the
!>         lower triangular part of the updated matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>         LDC is INTEGER
!>         On entry, LDC specifies the first dimension of C as declared
!>         in the calling (sub) program. LDC must be at least
!>         max( 1, n ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level3
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
!> =====
!> SUBROUTINE DSYR2K(UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!>
!> -- Reference BLAS level3 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!> November 2011
!>
!> .. Scalar Arguments ..
!> DOUBLE PRECISION ALPHA,BETA
!> INTEGER K,LDA,LDB,LDC,N
!> CHARACTER TRANS,UPLO
!> ..
!> .. Array Arguments ..
!> DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
!> ..
!>
!> =====
!>
!> .. External Functions ..
!> LOGICAL LSAME
!> EXTERNAL LSAME
!> ..
!> .. External Subroutines ..
!> EXTERNAL XERBLA
!> ..
!> .. Intrinsic Functions ..
!> INTRINSIC MAX
!> ..
!> .. Local Scalars ..
!> DOUBLE PRECISION TEMP1,TEMP2
!> INTEGER I,INFO,J,L,NROWA
!> LOGICAL UPPER
!> ..
!> .. Parameters ..
!> DOUBLE PRECISION ONE,ZERO
!> PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!> ..
!>
!> Test the input parameters.
!>
!> IF (LSAME(TRANS,'N')) THEN
!>   NROWA = N
!> ELSE
!>   NROWA = K
!> END IF
!> UPPER = LSAME(UPLO,'U')
!>
!> INFO = 0
!> IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!>   INFO = 1
!> ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
!>   (.NOT.LSAME(TRANS,'T')) .AND. &
!>   (.NOT.LSAME(TRANS,'C'))) THEN
!>   INFO = 2
!> ELSE IF (N.LT.0) THEN
!>   INFO = 3
!> ELSE IF (K.LT.0) THEN
!>   INFO = 4
!> ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!>   INFO = 7
!> ELSE IF (LDB.LT.MAX(1,NROWA)) THEN
!>   INFO = 9
!> ELSE IF (LDC.LT.MAX(1,N)) THEN
!>   INFO = 12
!> END IF
!> IF (INFO.NE.0) THEN
!>   CALL XERBLA('DSYR2K',INFO)
!>   RETURN
!> END IF
!>
!> Quick return if possible.
!>
!> IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).OR.&

```

```

        (K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN
!
!   And when  alpha.eq.zero.
!
    IF (ALPHA.EQ.ZERO) THEN
        IF (UPPER) THEN
            IF (BETA.EQ.ZERO) THEN
                DO 20 J = 1,N
                    DO 10 I = 1,J
                        C(I,J) = ZERO
                    CONTINUE
                CONTINUE
            ELSE
                DO 40 J = 1,N
                    DO 30 I = 1,J
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                CONTINUE
            END IF
        ELSE
            IF (BETA.EQ.ZERO) THEN
                DO 60 J = 1,N
                    DO 50 I = J,N
                        C(I,J) = ZERO
                    CONTINUE
                CONTINUE
            ELSE
                DO 80 J = 1,N
                    DO 70 I = J,N
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                CONTINUE
            END IF
        END IF
        RETURN
    END IF
!
!   Start the operations.
!
    IF (LSAME(TRANS,'N')) THEN
        Form C := alpha*A*B**T + alpha*B*A**T + C.
        IF (UPPER) THEN
            DO 130 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 90 I = 1,J
                        C(I,J) = ZERO
                    CONTINUE
                ELSE IF (BETA.NE.ONE) THEN
                    DO 100 I = 1,J
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                END IF
                DO 120 L = 1,K
                    IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                        TEMP1 = ALPHA*B(J,L)
                        TEMP2 = ALPHA*A(J,L)
                        DO 110 I = 1,J
                            C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                                B(I,L)*TEMP2
                        CONTINUE
                    END IF
                CONTINUE
            CONTINUE
        CONTINUE
    ELSE
        DO 180 J = 1,N
            IF (BETA.EQ.ZERO) THEN
                DO 140 I = J,N
                    C(I,J) = ZERO
                CONTINUE
            ELSE IF (BETA.NE.ONE) THEN
                DO 150 I = J,N
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            END IF
            DO 170 L = 1,K
                IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                    TEMP1 = ALPHA*B(J,L)
                    TEMP2 = ALPHA*A(J,L)
                    DO 160 I = J,N
                        C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                            B(I,L)*TEMP2
                    CONTINUE
                END IF
            CONTINUE
        CONTINUE
    END IF
    CONTINUE
END IF
ELSE
    Form C := alpha*A**T*B + alpha*B**T*A + C.
    IF (UPPER) THEN
        DO 210 J = 1,N
            DO 200 I = 1,J
                TEMP1 = ZERO
                TEMP2 = ZERO
                DO 190 L = 1,K
                    TEMP1 = TEMP1 + A(L,I)*B(L,J)
                    TEMP2 = TEMP2 + B(L,I)*A(L,J)
                CONTINUE
                IF (BETA.EQ.ZERO) THEN
                    C(I,J) = ALPHA*TEMP1 + ALPHA*TEMP2
                ELSE
                    C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
                        ALPHA*TEMP2
                END IF
            CONTINUE
        CONTINUE
    CONTINUE
    ELSE
        DO 240 J = 1,N
            DO 230 I = J,N
                TEMP1 = ZERO
                TEMP2 = ZERO

```

```

DO 220 L = 1,K
    TEMP1 = TEMP1 + A(L,I)*B(L,J)
    TEMP2 = TEMP2 + B(L,I)*A(L,J)
220 CONTINUE
    IF (BETA.EQ.ZERO) THEN
        C(I,J) = ALPHA*TEMP1 + ALPHA*TEMP2
    ELSE
        C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
            ALPHA*TEMP2
    END IF
230 CONTINUE
240 CONTINUE
    END IF
END IF
RETURN
! End of DSYR2K.
!
END SUBROUTINE

end program

```

APPENDIX A.5 – ZBLAT1.f90

```

program testzblat1
implicit none

! Variables
CHARACTER :: kin

CALL ZBLAT1
print *, 'ZBLAT1 Done.'
read(*, '(A1)', kin
STOP

CONTAINS
!> \brief \b LSAME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! LOGICAL FUNCTION LSAME(CA,CB)
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> LSAME returns .TRUE. if CA is the same letter as CB regardless of
!> case.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] CA
!> \verbatim
!> CA is CHARACTER*1
!> \endverbatim
!>
!> \param[in] CB
!> \verbatim
!> CB is CHARACTER*1
!> CA and CB specify the single characters to be compared.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!> \ingroup aux_blas
!
! =====
! LOGICAL FUNCTION LSAME(CA,CB)
!
! -- Reference BLAS level1 routine (version 3.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
! =====
!
! .. Intrinsic Functions ..
! INTRINSIC ICHAR
!
! .. Local Scalars ..
! INTEGER INTA,INTB,ZCODE
! ..
!
! Test if the characters are equal
!
! LSAME = CA .EQ. CB
! IF (LSAME) RETURN
!
! Now test for equivalence if both characters are alphabetic.
!
! ZCODE = ICHAR('Z')
!
! Use 'Z' rather than 'A' so that ASCII can be detected on Prime
! machines, on which ICHAR returns a value with bit 8 set.
! ICHAR('A') on Prime machines returns 193 which is the same as
! ICHAR('A') on an EBCDIC machine.
!
! INTA = ICHAR(CA)
! INTB = ICHAR(CB)
!
! IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
!
! ASCII is assumed - ZCODE is the ASCII code of either lower or
! upper case 'Z'.
!
! IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
! IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
!
! ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN

```

```

!
!      EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or
!      upper case 'Z'.
!
!      IF (INTA.GE.129 .AND. INTA.LE.137 .OR.&
!          INTA.GE.145 .AND. INTA.LE.153 .OR.&
!          INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
!      IF (INTB.GE.129 .AND. INTB.LE.137 .OR.&
!          INTB.GE.145 .AND. INTB.LE.153 .OR.&
!          INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
!
!      ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
!
!          ASCII is assumed, on Prime machines - ZCODE is the ASCII code
!          plus 128 of either lower or upper case 'Z'.
!
!          IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
!          IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
!      END IF
!      LSAME = INTA .EQ. INTB
!
!      RETURN
!
!      End of LSAME
!
!      END FUNCTION

SUBROUTINE ZBLAT1
!
!      *> \brief \b ZBLAT2
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      PROGRAM ZBLAT1
!
!      !> \par Purpose:
!      !      =====
!      !>
!      !> \verbatim
!      !>
!      !>      Test program for the COMPLEX*16 Level 1 BLAS.
!      !>
!      !>      Based upon the original BLAS test routine together with:
!      !>      F06GAF Example Program Text
!      !> \endverbatim
!
!      !
!      ! Authors:
!      ! =====
!      !
!      !> \author Univ. of Tennessee
!      !> \author Univ. of California Berkeley
!      !> \author Univ. of Colorado Denver
!      !> \author NAG Ltd.
!      !
!      !> \date April 2012
!      !
!      !> \ingroup complex16_blas_testing
!
!      =====
!      SUBROUTINE ZBLAT1
!
!      -- Reference BLAS test routine (version 3.4.1) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!      April 2012
!
!      =====
!
!      .. Parameters ..
!      INTEGER          NOUT
!      PARAMETER        (NOUT=6)
!      !
!      .. Scalars in Common ..
!      INTEGER          ICASE, INCX, INCY, MODE, N
!      LOGICAL          PASS
!      !
!      .. Local Scalars ..
!      DOUBLE PRECISION SFAC
!      INTEGER          IC
!      !
!      .. External Subroutines ..
!      EXTERNAL         CHECK1, CHECK2, HEADER
!      !
!      .. Common blocks ..
!      COMMON            /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
!      !
!      .. Data statements ..
!      DATA             SFAC/9.765625D-4/
!      !
!      .. Executable Statements ..
!      WRITE (NOUT,99999)
!      DO 20 IC = 1, 10
!          ICASE = IC
!          CALL HEADER
!
!          Initialize PASS, INCX, INCY, and MODE for a new case.
!          The value 9999 for INCX, INCY or MODE will appear in the
!          detailed output, if any, for cases that do not involve
!          these parameters.
!
!          PASS = .TRUE.
!          INCX = 9999
!          INCY = 9999
!          MODE = 9999
!          IF (ICASE.LE.5) THEN
!              CALL CHECK2(SFAC)
!          ELSE IF (ICASE.GE.6) THEN
!              CALL CHECK1(SFAC)
!          END IF
!
!          -- Print
!          IF (PASS) WRITE (NOUT,99998)
!      20 CONTINUE
!
!

```

```

99999 FORMAT (' Complex BLAS Test Program Results',/1X)
99998 FORMAT ('          ----- PASS -----')
END SUBROUTINE
SUBROUTINE HEADER
! .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
! .. Scalars in Common ..
INTEGER          ICASE, INCX, INCY, MODE, N
LOGICAL          PASS
! .. Local Arrays ..
CHARACTER*6      L(10)
! .. Common blocks ..
COMMON           /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
! .. Data statements ..
DATA             L(1)/'ZDOTC '/
DATA             L(2)/'ZDOTU '/
DATA             L(3)/'ZAXPY '/
DATA             L(4)/'ZCOPY '/
DATA             L(5)/'ZSWAP '/
DATA             L(6)/'DZNRM2'/
DATA             L(7)/'DZASUM'/
DATA             L(8)/'ZSCAL '/
DATA             L(9)/'ZDSCAL'/
DATA             L(10)/'IZAMAX'/
! .. Executable Statements ..
WRITE (NOUT,99999) ICASE, L(ICASE)
RETURN

99999 FORMAT (' Test of subprogram number',I3,12X,A6)
END SUBROUTINE
SUBROUTINE CHECK1(SFAC)
! .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
! .. Scalar Arguments ..
DOUBLE PRECISION SFAC
! .. Scalars in Common ..
INTEGER          ICASE, INCX, INCY, MODE, N,ii
LOGICAL          PASS
! .. Local Scalars ..
COMPLEX*16       CA
DOUBLE PRECISION SA
INTEGER          I, J, LEN, NP1
! .. Local Arrays ..
COMPLEX*16       CTRUE5(8,5,2), CTRUE6(8,5,2), CV(8,5,2), CX(8),
DOUBLE PRECISION STRUE2(5), STRUE4(5)
INTEGER          ITRUE3(5)
! .. External Functions ..
DOUBLE PRECISION DZASUM, DZNRM2
INTEGER          IZAMAX
! .. External Subroutines ..
EXTERNAL         DZASUM, DZNRM2, IZAMAX
! .. External Subroutines ..
EXTERNAL         ZSCAL, ZDSCAL, CTEST, ITEST1, STEST1
! .. Intrinsic Functions ..
INTRINSIC        MAX
! .. Common blocks ..
COMMON           /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
! .. Data statements ..
DATA             SA, CA/0.3D0, (0.4D0,-0.7D0)/
DATA             ((CV(I,J,1),I=1,8),J=1,5)/(0.1D0,0.1D0), &
(1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
(1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
(1.0D0,2.0D0), (0.3D0,-0.4D0), (3.0D0,4.0D0), &
(3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
(3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
(0.1D0,-0.3D0), (0.5D0,-0.1D0), (5.0D0,6.0D0), &
(5.0D0,6.0D0), (5.0D0,6.0D0), (5.0D0,6.0D0), &
(5.0D0,6.0D0), (5.0D0,6.0D0), (0.1D0,0.1D0), &
(-0.6D0,0.1D0), (0.1D0,-0.3D0), (7.0D0,8.0D0), &
(7.0D0,8.0D0), (7.0D0,8.0D0), (7.0D0,8.0D0), &
(7.0D0,8.0D0), (0.3D0,0.1D0), (0.5D0,0.0D0), &
(0.0D0,0.5D0), (0.0D0,0.2D0), (2.0D0,3.0D0), &
(2.0D0,3.0D0), (2.0D0,3.0D0), (2.0D0,3.0D0)/
DATA             ((CV(I,J,2),I=1,8),J=1,5)/(0.1D0,0.1D0), &
(4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
(4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
(4.0D0,5.0D0), (0.3D0,-0.4D0), (6.0D0,7.0D0), &
(6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
(6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
(0.1D0,-0.3D0), (8.0D0,9.0D0), (0.5D0,-0.1D0), &
(2.0D0,5.0D0), (2.0D0,5.0D0), (2.0D0,5.0D0), &
(2.0D0,5.0D0), (2.0D0,5.0D0), (0.1D0,0.1D0), &
(3.0D0,6.0D0), (-0.6D0,0.1D0), (4.0D0,7.0D0), &
(0.1D0,-0.3D0), (7.0D0,2.0D0), (7.0D0,2.0D0), &
(7.0D0,2.0D0), (0.3D0,0.1D0), (5.0D0,8.0D0), &
(0.5D0,0.0D0), (6.0D0,9.0D0), (0.0D0,0.5D0), &
(8.0D0,3.0D0), (0.0D0,0.2D0), (9.0D0,4.0D0)/
DATA             STRUE2/0.0D0, 0.5D0, 0.6D0, 0.7D0, 0.8D0/
DATA             STRUE4/0.0D0, 0.7D0, 1.0D0, 1.3D0, 1.6D0/
DATA             ((CTRUE5(I,J,1),I=1,8),J=1,5)/(0.1D0,0.1D0), &
(1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
(1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
(1.0D0,2.0D0), (-0.16D0,-0.37D0), (3.0D0,4.0D0), &
(3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
(3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
(-0.17D0,-0.19D0), (0.13D0,-0.39D0), &
(5.0D0,6.0D0), (5.0D0,6.0D0), (5.0D0,6.0D0), &
(5.0D0,6.0D0), (5.0D0,6.0D0), (5.0D0,6.0D0), &
(0.11D0,-0.03D0), (-0.17D0,0.46D0), &
(-0.17D0,-0.19D0), (7.0D0,8.0D0), (7.0D0,8.0D0), &
(7.0D0,8.0D0), (7.0D0,8.0D0), (7.0D0,8.0D0), &
(0.19D0,-0.17D0), (0.20D0,-0.35D0), &
(0.35D0,0.20D0), (0.14D0,0.08D0), &
(2.0D0,3.0D0), (2.0D0,3.0D0), (2.0D0,3.0D0), &
(2.0D0,3.0D0)/
DATA             ((CTRUE5(I,J,2),I=1,8),J=1,5)/(0.1D0,0.1D0), &
(4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
(4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
(4.0D0,5.0D0), (-0.16D0,-0.37D0), (6.0D0,7.0D0), &
(6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
(6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
(-0.17D0,-0.19D0), (8.0D0,9.0D0), &
(0.13D0,-0.39D0), (2.0D0,5.0D0), (2.0D0,5.0D0), &

```

```

(2.0D0,5.0D0), (2.0D0,5.0D0), (2.0D0,5.0D0), &
(0.11D0,-0.03D0), (3.0D0,6.0D0), &
(-0.17D0,0.46D0), (4.0D0,7.0D0), &
(-0.17D0,-0.19D0), (7.0D0,2.0D0), (7.0D0,2.0D0), &
(7.0D0,2.0D0), (0.19D0,-0.17D0), (5.0D0,8.0D0), &
(0.20D0,-0.35D0), (6.0D0,9.0D0), &
(0.35D0,0.20D0), (8.0D0,3.0D0), &
(0.14D0,0.08D0), (9.0D0,4.0D0)/
DATA
  ((CTRUE6(I,J,1),I=1,8),J=1,5)/(0.1D0,0.1D0), &
  (1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
  (1.0D0,2.0D0), (1.0D0,2.0D0), (1.0D0,2.0D0), &
  (1.0D0,2.0D0), (0.09D0,-0.12D0), (3.0D0,4.0D0), &
  (3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
  (3.0D0,4.0D0), (3.0D0,4.0D0), (3.0D0,4.0D0), &
  (0.03D0,-0.09D0), (0.15D0,-0.03D0), &
  (5.0D0,6.0D0), (5.0D0,6.0D0), (5.0D0,6.0D0), &
  (5.0D0,6.0D0), (5.0D0,6.0D0), (5.0D0,6.0D0), &
  (0.03D0,0.03D0), (-0.18D0,0.03D0), &
  (0.03D0,-0.09D0), (7.0D0,8.0D0), (7.0D0,8.0D0), &
  (7.0D0,8.0D0), (7.0D0,8.0D0), (7.0D0,8.0D0), &
  (0.09D0,0.03D0), (0.15D0,0.00D0), &
  (0.00D0,0.15D0), (0.00D0,0.06D0), (2.0D0,3.0D0), &
  (2.0D0,3.0D0), (2.0D0,3.0D0), (2.0D0,3.0D0)/
DATA
  ((CTRUE6(I,J,2),I=1,8),J=1,5)/(0.1D0,0.1D0), &
  (4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
  (4.0D0,5.0D0), (4.0D0,5.0D0), (4.0D0,5.0D0), &
  (4.0D0,5.0D0), (0.09D0,-0.12D0), (6.0D0,7.0D0), &
  (6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
  (6.0D0,7.0D0), (6.0D0,7.0D0), (6.0D0,7.0D0), &
  (0.03D0,-0.09D0), (8.0D0,9.0D0), &
  (0.15D0,-0.03D0), (2.0D0,5.0D0), (2.0D0,5.0D0), &
  (2.0D0,5.0D0), (2.0D0,5.0D0), (2.0D0,5.0D0), &
  (0.03D0,0.03D0), (3.0D0,6.0D0), &
  (-0.18D0,0.03D0), (4.0D0,7.0D0), &
  (0.03D0,-0.09D0), (7.0D0,2.0D0), (7.0D0,2.0D0), &
  (7.0D0,2.0D0), (0.09D0,0.03D0), (5.0D0,8.0D0), &
  (0.15D0,0.00D0), (6.0D0,9.0D0), (0.00D0,0.15D0), &
  (8.0D0,3.0D0), (0.00D0,0.06D0), (9.0D0,4.0D0)/
DATA
  ITRUE3/0, 1, 2, 2, 2/
! .. Executable Statements ..
DO I = 1,8
  DO J = 1,5
    print *,I,J
    print *,CTRUE5(I,J,1)
    print *,CTRUE5(I,J,2)
    print *,CTRUE6(I,J,1)
    print *,CTRUE6(I,J,2)
    print *,CV(I,J,1)
    print *,CV(I,J,2)
72  END DO
71  END DO
! STOP
DO 60 INCX = 1, 2
  DO 40 NP1 = 1, 5
    N = NP1 - 1
    LEN = 2*MAX(N,1)
    .. Set vector arguments ..
    DO 20 I = 1, LEN
      CX(I) = CV(I,NP1,INCX)
20  CONTINUE
    IF (ICASE.EQ.6) THEN
      .. DZNRM2 ..
      print *,n,incx
      DO 22 ii=1,8
        print *,cx(ii)
22  CONTINUE
      print *,DZNRM2(N,CX,INCX)
      print *,'END DZNRM2***'
      CALL STEST1(DZNRM2(N,CX,INCX),STRUE2(NP1),STRUE2(NP1),
      ELSE IF (ICASE.EQ.7) THEN
        .. DZASUM ..
        print *,n,incx
        DO 22 ii=1,8
          print *,cx(ii)
22  CONTINUE
        print *,DZASUM(N,CX,INCX)
        print *,'END DZASUM***'
        CALL STEST1(DZASUM(N,CX,INCX),STRUE4(NP1),STRUE4(NP1),
      ELSE IF (ICASE.EQ.8) THEN
        .. ZSCAL ..
        CALL ZSCAL(N,CA,CX,INCX)
        CALL CTEST(LEN,CX,CTRUE5(1,NP1,INCX),CTRUE5(1,NP1,INCX),
      ELSE IF (ICASE.EQ.9) THEN
        .. ZDSCAL ..
        CALL ZDSCAL(N,SA,CX,INCX)
        CALL CTEST(LEN,CX,CTRUE6(1,NP1,INCX),CTRUE6(1,NP1,INCX),
      ELSE IF (ICASE.EQ.10) THEN
        .. IZAMAX ..
        CALL ITEST1(IZAMAX(N,CX,INCX),ITRUE3(NP1))
      ELSE
        WRITE (NOUT,*) ' Shouldn''t be here in CHECK1'
        STOP
      END IF
40  CONTINUE
60  CONTINUE
!
  INCX = 1
  IF (ICASE.EQ.8) THEN
    ZSCAL
    Add a test for alpha equal to zero.
    CA = (0.0D0,0.0D0)
    DO 80 I = 1, 5
      MWPCF(I) = (0.0D0,0.0D0)
      MWPCS(I) = (1.0D0,1.0D0)
80  CONTINUE
    CALL ZSCAL(5,CA,CX,INCX)
    CALL CTEST(5,CX,MWPCF,MWPCS,SFAC)
    ELSE IF (ICASE.EQ.9) THEN
      ZDSCAL
      Add a test for alpha equal to zero.
      SA = 0.0D0
      DO 100 I = 1, 5
        MWPCF(I) = (0.0D0,0.0D0)

```

```

100      MWPCS(I) = (1.0D0,1.0D0)
      CONTINUE
      CALL ZDSCAL(5,SA,CX,INCX)
      CALL CTEST(5,CX,MWPCT,MWPCS,SFAC)
!      Add a test for alpha equal to one.
      SA = 1.0D0
      DO 120 I = 1, 5
        MWPCT(I) = CX(I)
        MWPCS(I) = CX(I)
120    CONTINUE
      CALL ZDSCAL(5,SA,CX,INCX)
      CALL CTEST(5,CX,MWPCT,MWPCS,SFAC)
!      Add a test for alpha equal to minus one.
      SA = -1.0D0
      DO 140 I = 1, 5
        MWPCT(I) = -CX(I)
        MWPCS(I) = -CX(I)
140    CONTINUE
      CALL ZDSCAL(5,SA,CX,INCX)
      CALL CTEST(5,CX,MWPCT,MWPCS,SFAC)
      END IF
      RETURN
      END SUBROUTINE
      SUBROUTINE CHECK2(SFAC)
!      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
!      .. Scalar Arguments ..
      DOUBLE PRECISION SFAC
!      .. Scalars in Common ..
      INTEGER          ICASE, INCX, INCY, MODE, N
      LOGICAL          PASS
!      .. Local Scalars ..
      COMPLEX*16        CA
      INTEGER          I, J, KI, KN, KSIZE, LENX, LENY, MX, MY
!      .. Local Arrays ..
      COMPLEX*16        CDOT(1), CSIZE1(4), CSIZE2(7,2), CSIZE3(14), &
        CT10X(7,4,4), CT10Y(7,4,4), CT6(4,4), CT7(4,4), &
        CT8(7,4,4), CX(7), CX1(7), CY(7), CY1(7)
      INTEGER          INCXS(4), INCYS(4), LENS(4,2), NS(4)
!      .. External Functions ..
!      COMPLEX*16        ZDOTC, ZDOTU
!      EXTERNAL          ZDOTC, ZDOTU
!      .. External Subroutines ..
!      EXTERNAL          ZAXPY, ZCOPY, ZSWAP, CTEST
!      .. Intrinsic Functions ..
      INTRINSIC          ABS, MIN
!      .. Common blocks ..
      COMMON              /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
!      .. Data statements ..
      DATA                CA/(0.4D0,-0.7D0)/
      DATA                INCXS/1, 2, -2, -1/
      DATA                INCYS/1, -2, 1, -2/
      DATA                LENS/1, 1, 2, 4, 1, 1, 3, 7/
      DATA                NS/0, 1, 2, 4/
      DATA                CX1/(0.7D0,-0.8D0), (-0.4D0,-0.7D0), &
        (-0.1D0,-0.9D0), (0.2D0,-0.8D0), &
        (-0.9D0,-0.4D0), (0.1D0,0.4D0), (-0.6D0,0.6D0)/
      DATA                CY1/(0.6D0,-0.6D0), (-0.9D0,0.5D0), &
        (0.7D0,-0.6D0), (0.1D0,-0.5D0), (-0.1D0,-0.2D0), &
        (-0.5D0,-0.3D0), (0.8D0,-0.7D0)/
      DATA                ((CT8(I,J,1),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.32D0,-1.41D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.32D0,-1.41D0), &
        (-1.55D0,0.5D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.32D0,-1.41D0), (-1.55D0,0.5D0), &
        (0.03D0,-0.89D0), (-0.38D0,-0.96D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0)/
      DATA                ((CT8(I,J,2),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.32D0,-1.41D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (-0.07D0,-0.89D0), &
        (-0.9D0,0.5D0), (0.42D0,-1.41D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (-0.9D0,0.5D0), &
        (0.78D0,0.06D0), (-0.9D0,0.5D0), &
        (0.06D0,-0.13D0), (0.1D0,-0.5D0), &
        (-0.77D0,-0.49D0), (-0.5D0,-0.3D0), &
        (0.52D0,-1.51D0)/
      DATA                ((CT8(I,J,3),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.32D0,-1.41D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (-0.07D0,-0.89D0), &
        (-1.18D0,-0.31D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.78D0,0.06D0), (-1.54D0,0.97D0), &
        (0.03D0,-0.89D0), (-0.18D0,-1.31D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0)/
      DATA                ((CT8(I,J,4),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.32D0,-1.41D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.32D0,-1.41D0), (-0.9D0,0.5D0), &
        (0.05D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
        (0.0D0,0.0D0), (0.0D0,0.0D0), (0.32D0,-1.41D0), &
        (-0.9D0,0.5D0), (0.05D0,-0.6D0), (0.1D0,-0.5D0), &
        (-0.77D0,-0.49D0), (-0.5D0,-0.3D0), &
        (0.32D0,-1.16D0)/
      DATA                CT7/(0.0D0,0.0D0), (-0.06D0,-0.90D0), &
        (0.65D0,-0.47D0), (-0.34D0,-1.22D0), &
        (0.0D0,0.0D0), (-0.06D0,-0.90D0), &
        (-0.59D0,-1.46D0), (-1.04D0,-0.04D0), &
        (0.0D0,0.0D0), (-0.06D0,-0.90D0), &
        (-0.83D0,0.59D0), (0.07D0,-0.37D0), &
        (0.0D0,0.0D0), (-0.06D0,-0.90D0), &
        (-0.76D0,-1.15D0), (-1.33D0,-1.82D0)/

```

```

DATA      CT6/(0.0D0,0.0D0), (0.90D0,0.06D0), &
          (0.91D0,-0.77D0), (1.80D0,-0.10D0), &
          (0.0D0,0.0D0), (0.90D0,0.06D0), (1.45D0,0.74D0), &
          (0.20D0,0.90D0), (0.0D0,0.0D0), (0.90D0,0.06D0), &
          (-0.55D0,0.23D0), (0.83D0,-0.39D0), &
          (0.0D0,0.0D0), (0.90D0,0.06D0), (1.04D0,0.79D0), &
          (1.95D0,1.22D0)/
DATA      ((CT10X(I,J,1),I=1,7),J=1,4)/(0.7D0,-0.8D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.6D0,-0.6D0), (-0.9D0,0.5D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.6D0,-0.6D0), &
          (-0.9D0,0.5D0), (0.7D0,-0.6D0), (0.1D0,-0.5D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0)/
DATA      ((CT10X(I,J,2),I=1,7),J=1,4)/(0.7D0,-0.8D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.7D0,-0.6D0), (-0.4D0,-0.7D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.8D0,-0.7D0), &
          (-0.4D0,-0.7D0), (-0.1D0,-0.2D0), &
          (0.2D0,-0.8D0), (0.7D0,-0.6D0), (0.1D0,0.4D0), &
          (0.6D0,-0.6D0)/
DATA      ((CT10X(I,J,3),I=1,7),J=1,4)/(0.7D0,-0.8D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (-0.9D0,0.5D0), (-0.4D0,-0.7D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.1D0,-0.5D0), &
          (-0.4D0,-0.7D0), (0.7D0,-0.6D0), (0.2D0,-0.8D0), &
          (-0.9D0,0.5D0), (0.1D0,0.4D0), (0.6D0,-0.6D0)/
DATA      ((CT10X(I,J,4),I=1,7),J=1,4)/(0.7D0,-0.8D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.6D0,-0.6D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.6D0,-0.6D0), (0.7D0,-0.6D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.6D0,-0.6D0), &
          (0.7D0,-0.6D0), (-0.1D0,-0.2D0), (0.8D0,-0.7D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0)/
DATA      ((CT10Y(I,J,1),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.7D0,-0.8D0), (-0.4D0,-0.7D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.7D0,-0.8D0), &
          (-0.4D0,-0.7D0), (-0.1D0,-0.9D0), &
          (0.2D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0)/
DATA      ((CT10Y(I,J,2),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (-0.1D0,-0.9D0), (-0.9D0,0.5D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (-0.6D0,0.6D0), &
          (-0.9D0,0.5D0), (-0.9D0,-0.4D0), (0.1D0,-0.5D0), &
          (-0.1D0,-0.9D0), (-0.5D0,-0.3D0), &
          (0.7D0,-0.8D0)/
DATA      ((CT10Y(I,J,3),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (-0.1D0,-0.9D0), (0.7D0,-0.8D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (-0.6D0,0.6D0), &
          (-0.9D0,-0.4D0), (-0.1D0,-0.9D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0)/
DATA      ((CT10Y(I,J,4),I=1,7),J=1,4)/(0.6D0,-0.6D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (-0.1D0,-0.9D0), (-0.9D0,0.5D0), &
          (0.7D0,-0.8D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (-0.6D0,0.6D0), &
          (-0.9D0,0.5D0), (-0.4D0,-0.7D0), (0.1D0,-0.5D0), &
          (-0.1D0,-0.9D0), (-0.5D0,-0.3D0), &
          (0.2D0,-0.8D0)/
DATA      CSIZE1/(0.0D0,0.0D0), (0.9D0,0.9D0), &
          (1.63D0,1.73D0), (2.90D0,2.78D0)/
DATA      CSIZE3/(0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (1.0D0,0.0D0), (0.0D0,0.0D0), (1.17D0,1.17D0), &
          (1.17D0,1.17D0), (1.17D0,1.17D0), &
          (1.17D0,1.17D0), (1.17D0,1.17D0), &
          (1.17D0,1.17D0)/
DATA      CSIZE2/(0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (0.0D0,0.0D0), &
          (0.0D0,0.0D0), (0.0D0,0.0D0), (1.54D0,1.54D0), &
          (1.54D0,1.54D0), (1.54D0,1.54D0), &
          (1.54D0,1.54D0), (1.54D0,1.54D0), &
          (1.54D0,1.54D0), (1.54D0,1.54D0)/
!      .. Executable Statements ..
DO 60 KI = 1, 4
    INCX = INCXS(KI)
    INCY = INCYS(KI)
    MX = ABS(INCX)
    MY = ABS(INCY)
!
    DO 40 KN = 1, 4

```

```

      N = NS(KN)
      KSIZE = MIN(2,KN)
      LENX = LENS(KN,MX)
      LENY = LENS(KN,MY)
!      .. initialize all argument arrays ..
      DO 20 I = 1, 7
        CX(I) = CX1(I)
        CY(I) = CY1(I)
20    CONTINUE
      IF (ICASE.EQ.1) THEN
!      .. ZDOTC ..
        CDOT(1) = ZDOTC(N,CX,INCX,CY,INCY)
        CALL CTEST(1,CDOT,CT6(KN,KI),CSIZE1(KN),SFAC)
      ELSE IF (ICASE.EQ.2) THEN
!      .. ZDOTU ..
        CDOT(1) = ZDOTU(N,CX,INCX,CY,INCY)
        CALL CTEST(1,CDOT,CT7(KN,KI),CSIZE1(KN),SFAC)
      ELSE IF (ICASE.EQ.3) THEN
!      .. ZAXPY ..
        CALL ZAXPY(N,CX,CA,INCX,CY,INCY)
        CALL CTEST(LENY,CY,CT8(1,KN,KI),CSIZE2(1,KSIZE),SFAC)
      ELSE IF (ICASE.EQ.4) THEN
!      .. ZCOPY ..
        CALL ZCOPY(N,CX,INCX,CY,INCY)
        CALL CTEST(LENY,CY,CT10Y(1,KN,KI),CSIZE3,1.0D0)
      ELSE IF (ICASE.EQ.5) THEN
!      .. ZSWAP ..
        CALL ZSWAP(N,CX,INCX,CY,INCY)
        CALL CTEST(LENX,CX,CT10X(1,KN,KI),CSIZE3,1.0D0)
        CALL CTEST(LENY,CY,CT10Y(1,KN,KI),CSIZE3,1.0D0)
      ELSE
        WRITE (NOUT,*) ' Shouldn''t be here in CHECK2'
        STOP
      END IF
!
40    CONTINUE
60    CONTINUE
      RETURN
      END SUBROUTINE
      SUBROUTINE STEST(LEN,SCOMP,STRUE,SSIZE,SFAC)
!      ***** STEST *****
!
!      THIS SUBR COMPARES ARRAYS SCOMP() AND STRUE() OF LENGTH LEN TO
!      SEE IF THE TERM BY TERM DIFFERENCES, MULTIPLIED BY SFAC, ARE
!      NEGLIGIBLE.
!
!      C. L. LAWSON, JPL, 1974 DEC 10
!
!      .. Parameters ..
      INTEGER      NOUT
      DOUBLE PRECISION ZERO
      PARAMETER    (NOUT=6, ZERO=0.0D0)
!      .. Scalar Arguments ..
      DOUBLE PRECISION SFAC
      INTEGER      LEN
!      .. Array Arguments ..
      DOUBLE PRECISION SCOMP(LEN), SSIZE(LEN), STRUE(LEN)
!      .. Scalars in Common ..
      INTEGER      ICASE, INCX, INCY, MODE, N
      LOGICAL      PASS
!      .. Local Scalars ..
      DOUBLE PRECISION SD
      INTEGER      I
!      .. External Functions ..
      DOUBLE PRECISION SDIFF
      EXTERNAL     SDIFF
!      .. Intrinsic Functions ..
      INTRINSIC    ABS
!      .. Common blocks ..
      COMMON       /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
!      .. Executable Statements ..
!
      DO I=1,LEN
        write(*,1021),I,SCOMP(I),STRUE(I),SSIZE(I),SFAC,LEN
      END DO
!
      DO 40 I = 1, LEN
        SD = SCOMP(I) - STRUE(I)
        IF (ABS(SFAC*SD) .LE. ABS(SSIZE(I))*EPSILON(ZERO)) &
          GO TO 40
!
!      HERE      SCOMP(I) IS NOT CLOSE TO STRUE(I).
!
        IF (.NOT. PASS) GO TO 20
        PRINT FAIL MESSAGE AND HEADER.
!
        PASS = .FALSE.
        WRITE (NOUT,99999)
        WRITE (NOUT,99998)
20      WRITE (NOUT,99997) ICASE, N, INCX, INCY, MODE, I, SCOMP(I), STRUE(I), SD, SSIZE(I)
40    CONTINUE
      RETURN
!
1021  FORMAT (I3,4F12.3,I3)
99999 FORMAT ('          FAIL')
99998 FORMAT ('/' CASE N INCX INCY MODE I          ', &
              ' COMP(I)          TRUE(I)  DIFFERENCE', &
              ' SIZE(I)',/I3)
99997 FORMAT (I3,I4,I3,3I5,I3,2D36.8,2D12.4)
      END SUBROUTINE
      SUBROUTINE STEST1(SCOMP1,STRUE1,SSIZE,SFAC)
!      ***** STEST1 *****
!
!      THIS IS AN INTERFACE SUBROUTINE TO ACCOMMODATE THE FORTRAN
!      REQUIREMENT THAT WHEN A DUMMY ARGUMENT IS AN ARRAY, THE
!      ACTUAL ARGUMENT MUST ALSO BE AN ARRAY OR AN ARRAY ELEMENT.
!
!      C.L. LAWSON, JPL, 1978 DEC 6
!
!      .. Scalar Arguments ..
      DOUBLE PRECISION SCOMP1, SFAC, STRUE1
!      .. Array Arguments ..
      DOUBLE PRECISION SSIZE(*)
!      .. Local Arrays ..
      DOUBLE PRECISION SCOMP(1), STRUE(1)

```

```

! .. External Subroutines ..
! EXTERNAL          STEST
! .. Executable Statements ..
!
! SCOMP(1) = SCOMP1
! STRUE(1) = STRUE1
! CALL STEST(1,SCOMP,STRUE,SSIZE,SFAC)
!
! RETURN
! END SUBROUTINE
! DOUBLE PRECISION FUNCTION SDIFF(SA,SB)
! ***** SDIFF *****
! COMPUTES DIFFERENCE OF TWO NUMBERS.  C. L. LAWSON, JPL 1974 FEB 15
!
! .. Scalar Arguments ..
! DOUBLE PRECISION          SA, SB
! .. Executable Statements ..
! SDIFF = SA - SB
! RETURN
! END FUNCTION
! SUBROUTINE CTEST(LEN,CCOMP,CTRUE,Csize,SFAC)
! ***** CTEST *****
!
! C.L. LAWSON, JPL, 1978 DEC 6
!
! .. Scalar Arguments ..
! DOUBLE PRECISION SFAC
! INTEGER          LEN
! .. Array Arguments ..
! COMPLEX*16       CCOMP(LEN), Csize(LEN), CTRUE(LEN)
! .. Local Scalars ..
! INTEGER          I
! .. Local Arrays ..
! DOUBLE PRECISION SCOMP(20), Ssize(20), STRUE(20)
! .. External Subroutines ..
! EXTERNAL          STEST
! .. Intrinsic Functions ..
! INTRINSIC          DIMAG, DBLE
! .. Executable Statements ..
! DO 20 I = 1, LEN
!   SCOMP(2*I-1) = DBLE(CCOMP(I))
!   SCOMP(2*I) = DIMAG(CCOMP(I))
!   STRUE(2*I-1) = DBLE(CTRUE(I))
!   STRUE(2*I) = DIMAG(CTRUE(I))
!   Ssize(2*I-1) = DBLE(Csize(I))
!   Ssize(2*I) = DIMAG(Csize(I))
! 20 CONTINUE
!
! CALL STEST(2*LEN,SCOMP,STRUE,Ssize,SFAC)
! RETURN
! END SUBROUTINE
! SUBROUTINE ITEST1(ICOMP,ITRUE)
! ***** ITEST1 *****
!
! THIS SUBROUTINE COMPARES THE VARIABLES ICOMP AND ITRUE FOR
! EQUALITY.
! C. L. LAWSON, JPL, 1974 DEC 10
!
! .. Parameters ..
! INTEGER          NOUT
! PARAMETER        (NOUT=6)
! .. Scalar Arguments ..
! INTEGER          ICOMP, ITRUE
! .. Scalars in Common ..
! INTEGER          ICASE, INCX, INCY, MODE, N
! LOGICAL          PASS
! .. Local Scalars ..
! INTEGER          ID
! .. Common blocks ..
! COMMON           /COMBLA/ICASE, N, INCX, INCY, MODE, PASS
! .. Executable Statements ..
! IF (ICOMP.EQ.ITRUE) GO TO 40
!
! HERE ICOMP IS NOT EQUAL TO ITRUE.
!
! IF (.NOT. PASS) GO TO 20
! PRINT FAIL MESSAGE AND HEADER.
!
! PASS = .FALSE.
! WRITE (NOUT,99999)
! WRITE (NOUT,99998)
! 20 ID = ICOMP - ITRUE
! WRITE (NOUT,99997) ICASE, N, INCX, INCY, MODE, ICOMP, ITRUE, ID
! 40 CONTINUE
! RETURN
!
! 99999 FORMAT ('          FAIL')
! 99998 FORMAT (' CASE  N INCX INCY MODE          ', &
!              ' COMP          TRUE    DIFFERENCE', &
!              /1X)
! 99997 FORMAT (1X,I4,I3,3I5,2I36,I12)
! END SUBROUTINE
!
! . Data statements ..
! DATA          L(1)/'ZDOTC '/
! DATA          L(2)/'ZDOTU '/
! DATA          L(3)/'ZAXPY '/
! DATA          L(4)/'ZCOPY '/
! DATA          L(5)/'ZSWAP '/
! DATA          L(6)/'DZNRN2'/ DCABS1
! DATA          L(7)/'DZASUM'/
! DATA          L(8)/'ZSCAL '/
! DATA          L(9)/'ZDSCAL'/
! DATA          L(10)/'IZAMAX'/
! COMPLEX*16 FUNCTION ZDOTC(N,ZX,INCX,ZY,INCY)
!> \brief \b ZDOTC
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====

```

```

!
! COMPLEX*16 FUNCTION ZDOTC(N,ZX,INCX,ZY,INCY)
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZDOTC forms the dot product of a vector.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level1
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> jack dongarra, 3/11/78.
!> modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
! COMPLEX*16 FUNCTION ZDOTC(N,ZX,INCX,ZY,INCY)
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
! =====
!
! .. Local Scalars ..
! COMPLEX*16 ZTEMP
! INTEGER I,IX,IY
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG
! ..
! ZTEMP = (0.0d0,0.0d0)
! ZDOTC = (0.0d0,0.0d0)
! IF (N.LE.0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!     code for both increments equal to 1
!
!     DO I = 1,N
!         ZTEMP = ZTEMP + DCONJG(ZX(I))*ZY(I)
!     END DO
! ELSE
!
!     code for unequal increments or equal increments
!     not equal to 1
!
!     IX = 1
!     IY = 1
!     IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!     IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!     DO I = 1,N
!         ZTEMP = ZTEMP + DCONJG(ZX(IX))*ZY(IY)
!         IX = IX + INCX
!         IY = IY + INCY
!     END DO
! END IF
! ZDOTC = ZTEMP
! RETURN
! END FUNCTION
! COMPLEX*16 FUNCTION ZDOTU(N,ZX,INCX,ZY,INCY)
! *> \brief \b ZDOTU
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! COMPLEX*16 FUNCTION ZDOTU(N,ZX,INCX,ZY,INCY)
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
!> \par Purpose:

```

```

! =====
!>
!> \verbatim
!>      ZDOTU forms the dot product of two vectors.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!>      jack dongarra, 3/11/78.
!>      modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!>      COMPLEX*16 FUNCTION ZDOTU(N,ZX,INCX,ZY,INCY)
!>
!>      -- Reference BLAS level1 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd. ---
!>      November 2011
!
!>      .. Scalar Arguments ..
!>      INTEGER INCX,INCY,N
!>      ..
!>      .. Array Arguments ..
!>      COMPLEX*16 ZX(*),ZY(*)
!>      ..
!>
!> =====
!>      .. Local Scalars ..
!>      COMPLEX*16 ZTEMP
!>      INTEGER I,IX,IY
!>      ..
!>      ZTEMP = (0.0d0,0.0d0)
!>      ZDOTU = (0.0d0,0.0d0)
!>      IF (N.LE.0) RETURN
!>      IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!>
!>          code for both increments equal to 1
!>
!>          DO I = 1,N
!>              ZTEMP = ZTEMP + ZX(I)*ZY(I)
!>          END DO
!>      ELSE
!>
!>          code for unequal increments or equal increments
!>          not equal to 1
!>
!>          IX = 1
!>          IY = 1
!>          IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!>          IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!>          DO I = 1,N
!>              ZTEMP = ZTEMP + ZX(IX)*ZY(IY)
!>              IX = IX + INCX
!>              IY = IY + INCY
!>          END DO
!>      END IF
!>      ZDOTU = ZTEMP
!>      RETURN
!>      END FUNCTION
!>      SUBROUTINE ZAXPY(N,ZA,ZX,INCX,ZY,INCY)
!> \brief \b ZAXPY
!>
!>      ===== DOCUMENTATION =====
!>
!>      Online html documentation available at
!>      http://www.netlib.org/lapack/explore-html/
!>
!>      Definition:
!>      =====
!>
!>      SUBROUTINE ZAXPY(N,ZA,ZX,INCX,ZY,INCY)
!>
!>      .. Scalar Arguments ..
!>      COMPLEX*16 ZA
!>      INTEGER INCX,INCY,N
!>      ..
!>      .. Array Arguments ..
!>      COMPLEX*16 ZX(*),ZY(*)
!>      ..
!>
!> \par Purpose:
!> =====
!>
!> \verbatim
!>      ZAXPY constant times a vector plus a vector.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.

```

```

!
!> \date November 2011
!
!> \ingroup complex16_blas_level1
!
!> \par Further Details:
!
!> =====
!> \verbatim
!>
!>     jack dongarra, 3/11/78.
!>     modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZAXPY (N,ZA,ZX,INCX,ZY,INCY)
!>
!> -- Reference BLAS level1 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ZA
!> INTEGER INCX,INCY,N
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 ZX(*),ZY(*)
!> ..
!> =====
!>
!> .. Local Scalars ..
!> INTEGER I,IX,IY
!> ..
!> .. External Functions ..
!> DOUBLE PRECISION DCABS1
!> EXTERNAL DCABS1
!> ..
!> IF (N.LE.0) RETURN
!> IF (DCABS1(ZA).EQ.0.0d0) RETURN
!> IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!>
!>     code for both increments equal to 1
!>
!>     DO I = 1,N
!>         ZY(I) = ZY(I) + ZA*ZX(I)
!>     END DO
!> ELSE
!>
!>     code for unequal increments or equal increments
!>     not equal to 1
!>
!>     IX = 1
!>     IY = 1
!>     IF (INCX.LT.0) IX = (-N+1)*INCX + 1
!>     IF (INCY.LT.0) IY = (-N+1)*INCY + 1
!>     DO I = 1,N
!>         ZY(IY) = ZY(IY) + ZA*ZX(IX)
!>         IX = IX + INCX
!>         IY = IY + INCY
!>     END DO
!> END IF
!>
!> RETURN
!> END SUBROUTINE
!> SUBROUTINE ZCOPY(N,ZX,INCX,ZY,INCY)
!> *> \brief \b ZCOPY
!>
!> ===== DOCUMENTATION =====
!>
!> Online html documentation available at
!>     http://www.netlib.org/lapack/explore-html/
!>
!> Definition:
!> =====
!>
!> SUBROUTINE ZCOPY(N,ZX,INCX,ZY,INCY)
!>
!> .. Scalar Arguments ..
!> INTEGER INCX,INCY,N
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 ZX(*),ZY(*)
!> ..
!>
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!>     ZCOPY copies a vector, x, to a vector, y.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level1
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>     jack dongarra, linpack, 4/11/78.
!>     modified 12/3/93, array(1) declarations changed to array(*)
!>

```

```

!> \endverbatim
!>
! =====
! SUBROUTINE ZCOPY(N,ZX,INCX,ZY,INCY)
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
!
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
! =====
!
! .. Local Scalars ..
! INTEGER I,IX,IY
!
! ..
! IF (N.LE.0) RETURN
! IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
! code for both increments equal to 1
!
! DO I = 1,N
!   ZY(I) = ZX(I)
! END DO
! ELSE
!
! code for unequal increments or equal increments
! not equal to 1
!
! IX = 1
! IY = 1
! IF (INCX.LT.0) IX = (-N+1)*INCX + 1
! IF (INCY.LT.0) IY = (-N+1)*INCY + 1
! DO I = 1,N
!   ZY(IY) = ZX(IX)
!   IX = IX + INCX
!   IY = IY + INCY
! END DO
! END IF
! RETURN
! END SUBROUTINE
! SUBROUTINE ZSWAP(N,ZX,INCX,ZY,INCY)
! *> \brief \b ZSWAP
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZSWAP(N,ZX,INCX,ZY,INCY)
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
!
!> \par Purpose:
! =====
!> \verbatim
!> ZSWAP interchanges two vectors.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!> jack dongarra, 3/11/78.
!> modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
! SUBROUTINE ZSWAP(N,ZX,INCX,ZY,INCY)
!
! -- Reference BLAS level1 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,INCY,N
! ..
! .. Array Arguments ..
! COMPLEX*16 ZX(*),ZY(*)
! ..
!
! =====
!
!

```

```

! .. Local Scalars ..
COMPLEX*16 ZTEMP
INTEGER I,IX,IY
!
! ..
IF (N.LE.0) RETURN
IF (INCX.EQ.1 .AND. INCY.EQ.1) THEN
!
!   code for both increments equal to 1
DO I = 1,N
    ZTEMP = ZX(I)
    ZX(I) = ZY(I)
    ZY(I) = ZTEMP
END DO
ELSE
!
!   code for unequal increments or equal increments not equal
!   to 1
IX = 1
IY = 1
IF (INCX.LT.0) IX = (-N+1)*INCX + 1
IF (INCY.LT.0) IY = (-N+1)*INCY + 1
DO I = 1,N
    ZTEMP = ZX(IX)
    ZX(IX) = ZY(IY)
    ZY(IY) = ZTEMP
    IX = IX + INCX
    IY = IY + INCY
END DO
END IF
RETURN
END SUBROUTINE
DOUBLE PRECISION FUNCTION DZNRM2(N,X,INCX)
*> \brief \b DZNRM2
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!   DOUBLE PRECISION FUNCTION DZNRM2(N,X,INCX)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,N
!
!   .. Array Arguments ..
!   COMPLEX*16 X(*)
!   ..
!
!> \par Purpose:
! =====
!> \verbatim
!> DZNRM2 returns the euclidean norm of a vector via the function
!> name, so that
!>
!>   DZNRM2 := sqrt( x**H*x )
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!>
!> -- This version written on 25-October-1982.
!>   Modified on 14-October-1993 to inline the call to ZLASSQ.
!>   Sven Hammarling, Nag Ltd.
!> \endverbatim
!>
! =====
!   DOUBLE PRECISION FUNCTION DZNRM2(N,X,INCX)
!
!   -- Reference BLAS level1 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
!   .. Scalar Arguments ..
!   INTEGER INCX,N
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 X(*)
!   ..
! =====
!
!   .. Parameters ..
!   DOUBLE PRECISION ONE,ZERO
!   PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
!   .. Local Scalars ..
!   DOUBLE PRECISION NORM,SCALE,SSQ,TEMP
!   INTEGER IX
!
!   ..
!   .. Intrinsic Functions ..
!   INTRINSIC ABS,DBLE,DIMAG,SQRT
!   ..

```

```

!      print *, 'DZNRM2===IN'
!      IF (N.LT.1 .OR. INCX.LT.1) THEN
!          NORM = ZERO
!      ELSE
!          SCALE = ZERO
!          SSQ = ONE
!          The following loop is equivalent to this call to the LAPACK
!          auxiliary routine:
!          CALL ZLASSQ( N, X, INCX, SCALE, SSQ )
!      DO 10 IX = 1,1 + (N-1)*INCX,INCX
!          IF (DBLE(X(IX)).NE.ZERO) THEN
!              print *, 'xinx',X(IX)
!              TEMP = ABS(DBLE(X(IX)))
!              IF (SCALE.LT.TEMP) THEN
!                  SSQ = ONE + SSQ* (SCALE/TEMP)**2
!                  SCALE = TEMP
!              ELSE
!                  SSQ = SSQ + (TEMP/SCALE)**2
!                  END IF
!              END IF
!              print *, 'R',SSQ
!              IF (DIMAG(X(IX)).NE.ZERO) THEN
!                  print *, 'xini',X(IX)
!                  TEMP = ABS(DIMAG(X(IX)))
!                  IF (SCALE.LT.TEMP) THEN
!                      SSQ = ONE + SSQ* (SCALE/TEMP)**2
!                      SCALE = TEMP
!                  ELSE
!                      SSQ = SSQ + (TEMP/SCALE)**2
!                      END IF
!                  END IF
!              END IF
!              print *, 'I',SSQ
!              CONTINUE
!              NORM = SCALE*SQRT(SSQ)
!          END IF
!      print *, 'DZNRM2===OUT'
!      DZNRM2 = NORM
!      RETURN
!
!      End of DZNRM2.
!
!      END FUNCTION
!      DOUBLE PRECISION FUNCTION DCABS1(Z)
!      *> \brief \b DCABS1
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      DOUBLE PRECISION FUNCTION DCABS1(Z)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 Z
!      ..
!      ..
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!> DCABS1 computes absolute value of a double complex number
!> \endverbatim
!
!      Authors:
!      =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!      =====
!      DOUBLE PRECISION FUNCTION DCABS1(Z)
!
!      -- Reference BLAS level1 routine (version 3.4.0) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!      November 2011
!
!      .. Scalar Arguments ..
!      COMPLEX*16 Z
!      ..
!      ..
!
!      =====
!
!      .. Intrinsic Functions ..
!      INTRINSIC ABS,DBLE,DIMAG
!
!      DCABS1 = ABS(DBLE(Z)) + ABS(DIMAG(Z))
!      RETURN
!      END FUNCTION
!      DOUBLE PRECISION FUNCTION DZASUM(N,ZX,INCX)
!> \brief \b DZASUM
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====

```

```

!      DOUBLE PRECISION FUNCTION DZASUM(N,ZX,INCX)
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> DZASUM takes the sum of the absolute values.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup double_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!>
!> jack dongarra, 3/11/78.
!> modified 3/93 to return if incx .le. 0.
!> modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!      DOUBLE PRECISION FUNCTION DZASUM(N,ZX,INCX)
!
!      -- Reference BLAS level1 routine (version 3.4.0) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!      November 2011
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
! =====
!
!      .. Local Scalars ..
!      DOUBLE PRECISION STEMP
!      INTEGER I,NINCX
!      ..
!      .. External Functions ..
!      DOUBLE PRECISION DCABS1
!      EXTERNAL DCABS1
!      ..
!      DZASUM = 0.0d0
!      STEMP = 0.0d0
!      IF (N.LE.0 .OR. INCX.LE.0) RETURN
!      IF (INCX.EQ.1) THEN
!
!          code for increment equal to 1
!
!          DO I = 1,N
!              STEMP = STEMP + DCABS1(ZX(I))
!          END DO
!      ELSE
!
!          code for increment not equal to 1
!
!          NINCX = N*INCX
!          DO I = 1,NINCX,INCX
!              STEMP = STEMP + DCABS1(ZX(I))
!          END DO
!      END IF
!      DZASUM = STEMP
!      RETURN
!      END FUNCTION
!      SUBROUTINE ZSCAL(N,ZA,ZX,INCX)
!      *> \brief \b ZSCAL
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE ZSCAL(N,ZA,ZX,INCX)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 ZA
!      INTEGER INCX,N
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!>

```

```

!> ZSCAL scales a vector by a constant.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level1
!
!> \par Further Details:
! =====
!> \verbatim
!> jack dongarra, 3/11/78.
!> modified 3/93 to return if incx .le. 0.
!> modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZSCAL(N,ZA,ZX,INCX)
!>
!> -- Reference BLAS level1 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ZA
!> INTEGER INCX,N
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 ZX(*)
!> ..
!> =====
!>
!> .. Local Scalars ..
!> INTEGER I,NINCX
!> ..
!> IF (N.LE.0 .OR. INCX.LE.0) RETURN
!> IF (INCX.EQ.1) THEN
!>
!> code for increment equal to 1
!>
!> DO I = 1,N
!> ZX(I) = ZA*ZX(I)
!> END DO
!> ELSE
!>
!> code for increment not equal to 1
!>
!> NINCX = N*INCX
!> DO I = 1,NINCX,INCX
!> ZX(I) = ZA*ZX(I)
!> END DO
!> END IF
!> RETURN
!> END SUBROUTINE
!> SUBROUTINE ZDSCAL(N,DA,ZX,INCX)
!> *> \brief \b ZDSCAL
!>
!> ===== DOCUMENTATION =====
!>
!> Online html documentation available at
!> http://www.netlib.org/lapack/explore-html/
!>
!> Definition:
!> =====
!>
!> SUBROUTINE ZDSCAL(N,DA,ZX,INCX)
!>
!> .. Scalar Arguments ..
!> DOUBLE PRECISION DA
!> INTEGER INCX,N
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 ZX(*)
!> ..
!>
!>
!> \par Purpose:
!> =====
!> \verbatim
!> ZDSCAL scales a vector by a constant.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level1
!>
!> \par Further Details:
!> =====
!> \verbatim
!> jack dongarra, 3/11/78.
!> modified 3/93 to return if incx .le. 0.

```

```

!>      modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!      SUBROUTINE ZDSCAL(N,DA,ZX,INCX)
!
!      -- Reference BLAS level1 routine (version 3.4.0) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!      November 2011
!
!      .. Scalar Arguments ..
!      DOUBLE PRECISION DA
!      INTEGER INCX,N
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
! =====
!
!      .. Local Scalars ..
!      INTEGER I,NINCX
!
!      .. Intrinsic Functions ..
!      INTRINSIC DCMPLX
!
!      IF (N.LE.0 .OR. INCX.LE.0) RETURN
!      IF (INCX.EQ.1) THEN
!
!          code for increment equal to 1
!
!          DO I = 1,N
!              ZX(I) = DCMPLX(DA,0.0d0)*ZX(I)
!          END DO
!      ELSE
!
!          code for increment not equal to 1
!
!          NINCX = N*INCX
!          DO I = 1,NINCX,INCX
!              ZX(I) = DCMPLX(DA,0.0d0)*ZX(I)
!          END DO
!      END IF
!      RETURN
!      END SUBROUTINE
!      INTEGER FUNCTION IZAMAX(N,ZX,INCX)
!      *> \brief \b IZAMAX
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      INTEGER FUNCTION IZAMAX(N,ZX,INCX)
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
!> \par Purpose:
!> =====
!> \verbatim
!>
!>      IZAMAX finds the index of element having max. absolute value.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup aux_blas
!>
!> \par Further Details:
!> =====
!> \verbatim
!>
!>      jack dongarra, 1/15/85.
!>      modified 3/93 to return if incx .le. 0.
!>      modified 12/3/93, array(1) declarations changed to array(*)
!> \endverbatim
!>
! =====
!      INTEGER FUNCTION IZAMAX(N,ZX,INCX)
!
!      -- Reference BLAS level1 routine (version 3.4.0) --
!      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!      November 2011
!
!      .. Scalar Arguments ..
!      INTEGER INCX,N
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 ZX(*)
!      ..
!
! =====
!

```

```

!      .. Local Scalars ..
      DOUBLE PRECISION DMAX
      INTEGER I,IX
!
!      ..
!      .. External Functions ..
!      DOUBLE PRECISION DCABS1
!      EXTERNAL DCABS1
!
!      ..
      IZAMAX = 0
      IF (N.LT.1 .OR. INCX.LE.0) RETURN
      IZAMAX = 1
      IF (N.EQ.1) RETURN
      IF (INCX.EQ.1) THEN
!
!         code for increment equal to 1
!
         DMAX = DCABS1(ZX(1))
         DO I = 2,N
            IF (DCABS1(ZX(I)).GT.DMAX) THEN
               IZAMAX = I
               DMAX = DCABS1(ZX(I))
            END IF
         END DO
      ELSE
!
!         code for increment not equal to 1
!
         IX = 1
         DMAX = DCABS1(ZX(1))
         IX = IX + INCX
         DO I = 2,N
            IF (DCABS1(ZX(IX)).GT.DMAX) THEN
               IZAMAX = I
               DMAX = DCABS1(ZX(IX))
            END IF
            IX = IX + INCX
         END DO
      END IF
      RETURN
      END FUNCTION
END PROGRAM

```

APPENDIX A.6 – ZBLAT2.f90

```

program testzblat2
implicit none

! Variables
CHARACTER :: kin

CALL ZBLAT2
print *, 'ZBLAT2 Done.'

CONTAINS
!> \brief \b LSAME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! LOGICAL FUNCTION LSAME(CA,CB)
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> LSAME returns .TRUE. if CA is the same letter as CB regardless of
!> case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] CA
!> \verbatim
!> CA is CHARACTER*1
!> \endverbatim
!> \param[in] CB
!> \verbatim
!> CB is CHARACTER*1
!> CA and CB specify the single characters to be compared.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup aux_blas
!
! =====
! LOGICAL FUNCTION LSAME(CA,CB)
!
! -- Reference BLAS level1 routine (version 3.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
! =====
!
! .. Intrinsic Functions ..
! INTRINSIC ICHAR
! ..
! .. Local Scalars ..
! INTEGER INTA,INTB,ZCODE
! ..
!
! Test if the characters are equal
!
! LSAME = CA .EQ. CB
! IF (LSAME) RETURN
!
! Now test for equivalence if both characters are alphabetic.
!
! ZCODE = ICHAR('Z')
!
! Use 'Z' rather than 'A' so that ASCII can be detected on Prime
! machines, on which ICHAR returns a value with bit 8 set.
! ICHAR('A') on Prime machines returns 193 which is the same as
! ICHAR('A') on an EBCDIC machine.
!
! INTA = ICHAR(CA)
! INTB = ICHAR(CB)
!
! IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
!
! ASCII is assumed - ZCODE is the ASCII code of either lower or
! upper case 'Z'.
!
! IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
! IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
!
! ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN
!
! EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or

```

```

!      upper case 'Z'.
!
!      IF (INTA.GE.129 .AND. INTA.LE.137 .OR.&
!        INTA.GE.145 .AND. INTA.LE.153 .OR.&
!        INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
!      IF (INTB.GE.129 .AND. INTB.LE.137 .OR.&
!        INTB.GE.145 .AND. INTB.LE.153 .OR.&
!        INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
!
!      ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
!
!        ASCII is assumed, on Prime machines - ZCODE is the ASCII code
!        plus 128 of either lower or upper case 'Z'.
!
!        IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
!        IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
!      END IF
!      LSAME = INTA .EQ. INTB
!
!      RETURN
!
!      End of LSAME
!
!      END FUNCTION
!> \brief \b ZBLAT2
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      PROGRAM ZBLAT2
!
!> \par Purpose:
!      =====
!>
!> \verbatim
!>
!> Test program for the COMPLEX*16          Level 2 Blas.
!>
!> The program must be driven by a short data file. The first 18 records
!> of the file are read using list-directed input, the last 17 records
!> are read using the format ( A6, L2 ). An annotated example of a data
!> file can be obtained by deleting the first 3 characters from the
!> following 35 lines:
!> 'zblat2.out'      NAME OF SUMMARY OUTPUT FILE
!> 6                 UNIT NUMBER OF SUMMARY FILE
!> 'CBLA2T.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
!> -1                UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
!> F                 LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
!> F                 LOGICAL FLAG, T TO STOP ON FAILURES.
!> T                 LOGICAL FLAG, T TO TEST ERROR EXITS.
!> 16.0              THRESHOLD VALUE OF TEST RATIO
!> 6                 NUMBER OF VALUES OF N
!> 0 1 2 3 5 9       VALUES OF N
!> 4                 NUMBER OF VALUES OF K
!> 0 1 2 4            VALUES OF K
!> 4                 NUMBER OF VALUES OF INCX AND INCY
!> 1 2 -1 -2         VALUES OF INCX AND INCY
!> 3                 NUMBER OF VALUES OF ALPHA
!> (0.0,0.0) (1.0,0.0) (0.7,-0.9)  VALUES OF ALPHA
!> 3                 NUMBER OF VALUES OF BETA
!> (0.0,0.0) (1.0,0.0) (1.3,-1.1)  VALUES OF BETA
!> ZGEMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZGBMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHEMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHBMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHPMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTRMV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTRBV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTRSV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTBSV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTPSV T PUT F FOR NO TEST. SAME COLUMNS.
!> ZGERC T PUT F FOR NO TEST. SAME COLUMNS.
!> ZGERU T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHER T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHER2 T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHPR T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHPR2 T PUT F FOR NO TEST. SAME COLUMNS.
!>
!> Further Details
!> =====
!>
!> See:
!>
!> Dongarra J. J., Du Croz J. J., Hammarling S. and Hanson R. J..
!> An extended set of Fortran Basic Linear Algebra Subprograms.
!>
!> Technical Memoranda Nos. 41 (revision 3) and 81, Mathematics
!> and Computer Science Division, Argonne National Laboratory,
!> 9700 South Cass Avenue, Argonne, Illinois 60439, US.
!>
!> Or
!>
!> NAG Technical Reports TR3/87 and TR4/87, Numerical Algorithms
!> Group Ltd., NAG Central Office, 256 Banbury Road, Oxford
!> OX2 7DE, UK, and Numerical Algorithms Group Inc., 1101 31st
!> Street, Suite 100, Downers Grove, Illinois 60515-1263, USA.
!>
!>
!> -- Written on 10-August-1987.
!> Richard Hanson, Sandia National Labs.
!> Jeremy Du Croz, NAG Central Office.
!>
!> 10-9-00: Change STATUS='NEW' to 'UNKNOWN' so that the testers
!> can be run multiple times without deleting generated
!> output files (susan)
!> \endverbatim
!

```

```

! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date April 2012
!
!> \ingroup complex16_blas_testing
!
! =====
!
SUBROUTINE ZBLAT2
!
! -- Reference BLAS test routine (version 3.4.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! April 2012
!
! =====
!
! .. Parameters ..
INTEGER          NIN
PARAMETER        ( NIN = 5 )
INTEGER          NSUBS
PARAMETER        ( NSUBS = 17 )
COMPLEX*16       ZERO, ONE
PARAMETER        ( ZERO = ( 0.0D0, 0.0D0 ), ONE = ( 1.0D0, 0.0D0 ) )
DOUBLE PRECISION RZERO
PARAMETER        ( RZERO = 0.0D0 )
INTEGER          NMAX, INCMAX
PARAMETER        ( NMAX = 65, INCMAX = 2 )
INTEGER          NINMAX, NIDMAX, NKBMAX, NALMAX, NBEMAX
PARAMETER        ( NINMAX = 7, NIDMAX = 9, NKBMAX = 7, &
                  NALMAX = 7, NBEMAX = 7 )
!
! .. Local Scalars ..
DOUBLE PRECISION EPS, ERR, THRESH
INTEGER          I, ISNUM, J, N, NALF, NBET, NIDIM, NINC, NKB, &
                  NOUT, NTRA
LOGICAL          FATAL, LTESTT, REWI, SAME, SFATAL, TRACE,      TSTERR
CHARACTER*1      TRANS
CHARACTER*6      SNAMET
CHARACTER*32     SNAPS, SUMMRY
!
! .. Local Arrays ..
COMPLEX*16       A( NMAX, NMAX ), AA( NMAX*NMAX ), &
                  ALF( NALMAX ), AS( NMAX*NMAX ), BET( NBEMAX ), &
                  X( NMAX ), XS( NMAX*INCMAX ), &
                  XX( NMAX*INCMAX ), Y( NMAX ), &
                  YS( NMAX*INCMAX ), YT( NMAX ), &
                  YY( NMAX*INCMAX ), Z( 2*NMAX )
DOUBLE PRECISION G( NMAX )
INTEGER          IDIM( NIDMAX ), INC( NINMAX ), KB( NKBMAX )
LOGICAL          LTEST( NSUBS )
CHARACTER*6      SNAMES( NSUBS )
!
! .. External Functions ..
DOUBLE PRECISION DDIFF
LOGICAL          LZE
EXTERNAL         DDIFF, LZE
!
! .. External Subroutines ..
EXTERNAL         ZCHK1, ZCHK2, ZCHK3, ZCHK4, ZCHK5, ZCHK6,
$               ZCHKE, ZMVCH
!
! .. Intrinsic Functions ..
INTRINSIC        ABS, MAX, MIN
!
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
CHARACTER*6      SRNAMT
!
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUTC, OK, LERR
COMMON           /SRNAMC/SRNAMT
!
! .. Data statements ..
DATA             SNAMES/'ZGEMV ', 'ZGBMV ', 'ZHEMV ', 'ZHBMV ', &
                  'ZHPMV ', 'ZTRMV ', 'ZTBMV ', 'ZTPMV ', &
                  'ZTRSV ', 'ZTBSV ', 'ZTPSV ', 'ZGERC ', &
                  'ZGERU ', 'ZHER ', 'ZHPR ', 'ZHER2 ', &
                  'ZHPR2 '/
!
! .. Executable Statements ..
!
! Read name and unit number for summary output file and open file.
!
READ( NIN, FMT = * ) SUMMRY
READ( NIN, FMT = * ) NOUT
OPEN( NOUT, FILE = SUMMRY, STATUS = 'UNKNOWN' )
NOUTC = NOUT
!
! Read name and unit number for snapshot output file and open file.
!
READ( NIN, FMT = * ) SNAPS
READ( NIN, FMT = * ) NTRA
TRACE = NTRA.GE.0
IF( TRACE ) THEN
    OPEN( NTRA, FILE = SNAPS, STATUS = 'UNKNOWN' )
END IF
!
! Read the flag that directs rewinding of the snapshot file.
READ( NIN, FMT = * ) REWI
REWI = REWI.AND.TRACE
!
! Read the flag that directs stopping on any failure.
READ( NIN, FMT = * ) SFATAL
!
! Read the flag that indicates whether error exits are to be tested.
READ( NIN, FMT = * ) TSTERR
!
! Read the threshold value of the test ratio
READ( NIN, FMT = * ) THRESH
!
! Read and check the parameter values for the tests.
!
!
! Values of N
READ( NIN, FMT = * ) NIDIM
IF( NIDIM.LT.1.OR.NIDIM.GT.NIDMAX ) THEN
    WRITE( NOUT, FMT = 9997 ) 'N', NIDMAX
    GO TO 230
END IF
READ( NIN, FMT = * ) ( IDIM( I ), I = 1, NIDIM )
DO 10 I = 1, NIDIM

```

```

        IF( IDIM( I ).LT.0.OR.IDIM( I ).GT.NMAX )THEN
            WRITE( NOUT, FMT = 9996 )NMAX
            GO TO 230
        END IF
10 CONTINUE
    Values of K
    READ( NIN, FMT = * )NKB
    IF( NKB.LT.1.OR.NKB.GT.NKBMAX )THEN
        WRITE( NOUT, FMT = 9997 )'K', NKBMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( KB( I ), I = 1, NKB )
    DO 20 I = 1, NKB
        IF( KB( I ).LT.0 )THEN
            WRITE( NOUT, FMT = 9995 )
            GO TO 230
        END IF
20 CONTINUE
    Values of INCX and INCY
    READ( NIN, FMT = * )NINC
    IF( NINC.LT.1.OR.NINC.GT.NINMAX )THEN
        WRITE( NOUT, FMT = 9997 )'INCX AND INCY', NINMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( INC( I ), I = 1, NINC )
    DO 30 I = 1, NINC
        IF( INC( I ).EQ.0.OR.ABS( INC( I ) ).GT.INCMAX )THEN
            WRITE( NOUT, FMT = 9994 )INCMAX
            GO TO 230
        END IF
30 CONTINUE
    Values of ALPHA
    READ( NIN, FMT = * )NALF
    IF( NALF.LT.1.OR.NALF.GT.NALMAX )THEN
        WRITE( NOUT, FMT = 9997 )'ALPHA', NALMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( ALF( I ), I = 1, NALF )
    Values of BETA
    READ( NIN, FMT = * )NBET
    IF( NBET.LT.1.OR.NBET.GT.NBEMAX )THEN
        WRITE( NOUT, FMT = 9997 )'BETA', NBEMAX
        GO TO 230
    END IF
    READ( NIN, FMT = * )( BET( I ), I = 1, NBET )
    Report values of parameters.
    WRITE( NOUT, FMT = 9993 )
    WRITE( NOUT, FMT = 9992 )( IDIM( I ), I = 1, NIDIM )
    WRITE( NOUT, FMT = 9991 )( KB( I ), I = 1, NKB )
    WRITE( NOUT, FMT = 9990 )( INC( I ), I = 1, NINC )
    WRITE( NOUT, FMT = 9989 )( ALF( I ), I = 1, NALF )
    WRITE( NOUT, FMT = 9988 )( BET( I ), I = 1, NBET )
    IF( .NOT.TSTERR )THEN
        WRITE( NOUT, FMT = * )
        WRITE( NOUT, FMT = 9980 )
    END IF
    WRITE( NOUT, FMT = * )
    WRITE( NOUT, FMT = 9999 )THRESH
    WRITE( NOUT, FMT = * )
    Read names of subroutines and flags which indicate
    whether they are to be tested.
    DO 40 I = 1, NSUBS
        LTEST( I ) = .FALSE.
40 CONTINUE
50 READ( NIN, FMT = 9984, END = 80 )SNAMET, LTESTT
    DO 60 I = 1, NSUBS
        IF( SNAMET.EQ.SNAMES( I ) )&
            GO TO 70
60 CONTINUE
    WRITE( NOUT, FMT = 9986 )SNAMET
    STOP
70 LTEST( I ) = LTESTT
    GO TO 50
80 CONTINUE
    CLOSE ( NIN )
    Compute EPS (the machine precision).
    EPS = EPSILON(RZERO)
    WRITE( NOUT, FMT = 9998 )EPS
    Check the reliability of ZMVCH using exact data.
    N = MIN( 32, NMAX )
    DO 120 J = 1, N
        DO 110 I = 1, N
            A( I, J ) = MAX( I - J + 1, 0 )
110 CONTINUE
            X( J ) = J
            Y( J ) = ZERO
120 CONTINUE
            DO 130 J = 1, N
                YY( J ) = J*( ( J + 1)*J )/2 - ( ( J + 1 )*J*( J - 1 ) )/3
130 CONTINUE
            YY holds the exact result. On exit from ZMVCH YT holds
            the result computed by ZMVCH.
            TRANS = 'N'
            CALL ZMVCH( TRANS, N, N, ONE, A, NMAX, X, 1, ZERO, Y, 1, YT, G,&
                YY, EPS, ERR, FATAL, NOUT, .TRUE. )
            SAME = LZE( YY, YT, N )
            IF( .NOT.SAME.OR.ERR.NE.RZERO )THEN
                WRITE( NOUT, FMT = 9985 )TRANS, SAME, ERR
                STOP
            END IF
            TRANS = 'T'
            CALL ZMVCH( TRANS, N, N, ONE, A, NMAX, X, -1, ZERO, Y, -1, YT, G,&
                YY, EPS, ERR, FATAL, NOUT, .TRUE. )
            SAME = LZE( YY, YT, N )
            IF( .NOT.SAME.OR.ERR.NE.RZERO )THEN

```

```

        WRITE( NOUT, FMT = 9985 )TRANS, SAME, ERR
        STOP
    END IF
!
!   Test each subroutine in turn.
!
DO 210 ISNUM = 1, NSUBS
    WRITE( NOUT, FMT = * )
    IF( .NOT. LTEST( ISNUM ) ) THEN
!       Subprogram is not to be tested.
        WRITE( NOUT, FMT = 9983 )SNAMES( ISNUM )
    ELSE
        SRNAMT = SNAMES( ISNUM )
!       Test error exits.
        IF( TSERR ) THEN
            CALL ZCHK4( ISNUM, SNAMES( ISNUM ), NOUT )
            WRITE( NOUT, FMT = * )
        END IF
!       Test computations.
        INFOT = 0
        OK = .TRUE.
        FATAL = .FALSE.
        GO TO ( 140, 140, 150, 150, 150, 160, 160, &
                160, 160, 160, 160, 170, 170, 180, &
                180, 190, 190 )ISNUM
!       Test ZGEMV, 01, and ZGBMV, 02.
140    CALL ZCHK1( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF, &
                NBET, BET, NINC, INC, NMAX, INCMAX, A, AA, AS, &
                X, XX, XS, Y, YY, YS, YT, G )
!       GO TO 200
!       Test ZHEMV, 03, ZHBMV, 04, and ZHPMV, 05.
150    CALL ZCHK2( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF, &
                NBET, BET, NINC, INC, NMAX, INCMAX, A, AA, AS, &
                X, XX, XS, Y, YY, YS, YT, G )
!       GO TO 200
!       Test ZTRMV, 06, ZTBMV, 07, ZTPMV, 08,
!       ZTRSV, 09, ZTBSV, 10, and ZTPSV, 11.
160    CALL ZCHK3( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NKB, KB, NINC, INC, &
                NMAX, INCMAX, A, AA, AS, Y, YY, YS, YT, G, Z )
!       GO TO 200
!       Test ZGERC, 12, ZGERU, 13.
170    CALL ZCHK4( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, &
                NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, &
                YT, G, Z )
!       GO TO 200
!       Test ZHER, 14, and ZHPR, 15.
180    CALL ZCHK5( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, &
                NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, &
                YT, G, Z )
!       GO TO 200
!       Test ZHER2, 16, and ZHPR2, 17.
190    CALL ZCHK6( SNAMES( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
                REWI, FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, &
                NMAX, INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, &
                YT, G, Z )
!
200    IF( FATAL.AND.SFATAL ) &
        GO TO 220
    END IF
210 CONTINUE
    WRITE( NOUT, FMT = 9982 )
    GO TO 240
!
220 CONTINUE
    WRITE( NOUT, FMT = 9981 )
    GO TO 240
!
230 CONTINUE
    WRITE( NOUT, FMT = 9987 )
!
240 CONTINUE
    IF( TRACE ) &
        CLOSE ( NTRA )
    CLOSE ( NOUT )
!
9999 FORMAT( ' ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LES', &
            'S THAN', F8.2 )
9998 FORMAT( ' RELATIVE MACHINE PRECISION IS TAKEN TO BE', 1P, D9.1 )
9997 FORMAT( ' NUMBER OF VALUES OF ', A, ' IS LESS THAN 1 OR GREATER ', &
            'THAN ', I2 )
9996 FORMAT( ' VALUE OF N IS LESS THAN 0 OR GREATER THAN ', I2 )
9995 FORMAT( ' VALUE OF K IS LESS THAN 0' )
9994 FORMAT( ' ABSOLUTE VALUE OF INCX OR INCY IS 0 OR GREATER THAN ', &
            I2 )
9993 FORMAT( ' TESTS OF THE COMPLEX*16          LEVEL 2 BLAS', '/' THE F', &
            'OLLOWING PARAMETER VALUES WILL BE USED:' )
9992 FORMAT( '   FOR N              ', 9I6 )
9991 FORMAT( '   FOR K              ', 7I6 )
9990 FORMAT( '   FOR INCX AND INCY  ', 7I6 )
9989 FORMAT( '   FOR ALPHA          ', &
            7( '(', F4.1, ',', F4.1, ')', ' ', : ) )
9988 FORMAT( '   FOR BETA           ', &
            7( '(', F4.1, ',', F4.1, ')', ' ', : ) )
9987 FORMAT( ' AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM', &
            '/' ***** TESTS ABANDONED ***** )
9986 FORMAT( ' SUBPROGRAM NAME ', A6, ' NOT RECOGNIZED', '/' ***** T', &
            'ESTS ABANDONED ***** )
9985 FORMAT( ' ERROR IN ZMVCH - IN-LINE DOT PRODUCTS ARE BEING EVALU', &
            'ATED WRONGLY.', '/' ZMVCH WAS CALLED WITH TRANS = ', A1, &
            ' AND RETURNED SAME = ', L1, ' AND ERR = ', F12.3, ' ', /&
            ' THIS MAY BE DUE TO FAULTS IN THE ARITHMETIC OR THE COMPILER.' &
            '/' ***** TESTS ABANDONED ***** )
9984 FORMAT( A6, L2 )
9983 FORMAT( 1X, A6, ' WAS NOT TESTED' )
9982 FORMAT( '/' END OF TESTS' )
9981 FORMAT( '/' ***** FATAL ERROR - TESTS ABANDONED ***** )
9980 FORMAT( ' ERROR-EXITS WILL NOT BE TESTED' )
!
!   End of ZBLAT2.

```

```

!
!      END SUBROUTINE
!      SUBROUTINE ZCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!          FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF, NBET,&
!          BET, NINC, INC, NMAX, INCMAX, A, AA, AS, X, XX,&
!          XS, Y, YY, YS, YT, G )
!
!      Tests ZGEMV and ZGBMV.
!
!      Auxiliary routine for test program for Level 2 Blas.
!
!      -- Written on 10-August-1987.
!      Richard Hanson, Sandia National Labs.
!      Jeremy Du Croz, NAG Central Office.
!
!      .. Parameters ..
!      COMPLEX*16      ZERO, HALF
!      PARAMETER       ( ZERO = ( 0.0D0, 0.0D0 ),&
!          HALF = ( 0.5D0, 0.0D0 ) )
!      DOUBLE PRECISION RZERO
!      PARAMETER       ( RZERO = 0.0D0 )
!      .. Scalar Arguments ..
!      DOUBLE PRECISION EPS, THRESH
!      INTEGER          INCMAX, NALF, NBET, NIDIM, NINC, NKB, NMAX,&
!          NOUT, NTRA
!      LOGICAL          FATAL, REWI, TRACE
!      CHARACTER*6      SNAME
!      .. Array Arguments ..
!      COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
!          AS( NMAX*NMAX ), BET( NBET ), X( NMAX ),&
!          XS( NMAX*INCMAX ), XX( NMAX*INCMAX ),&
!          Y( NMAX ), YS( NMAX*INCMAX ), YT( NMAX ),&
!          YY( NMAX*INCMAX )
!      DOUBLE PRECISION G( NMAX )
!      INTEGER          IDIM( NIDIM ), INC( NINC ), KB( NKB )
!      .. Local Scalars ..
!      COMPLEX*16      ALPHA, ALS, BETA, BLS, TRANSL
!      DOUBLE PRECISION ERR, ERRMAX
!      INTEGER          I, IA, IB, IC, IKU, IM, IN, INCX, INCXS, INCY,&
!          INCYS, IX, IY, KL, KLS, KU, KUS, LAA, LDA,&
!          LDAS, LX, LY, M, ML, MS, N, NARGS, NC, ND, NK,&
!          NL, NS
!      LOGICAL          BANDED, FULL, NULL, RESET, SAME, TRAN
!      CHARACTER*1      TRANS, TRANSS
!      CHARACTER*3      ICH
!      .. Local Arrays ..
!      LOGICAL          ISAME( 13 )
!      .. External Functions ..
!      LOGICAL          LZE, LZERES
!      EXTERNAL         LZE, LZERES
!      .. External Subroutines ..
!      EXTERNAL         ZGBMV, ZGEMV, ZMAKE, ZMVCH
!      .. Intrinsic Functions ..
!      INTRINSIC        ABS, MAX, MIN
!      .. Scalars in Common ..
!      INTEGER          INFOT, NOUTC
!      LOGICAL          LERR, OK
!      .. Common blocks ..
!      COMMON            /INFOC/INFOT, NOUTC, OK, LERR
!      .. Data statements ..
!      DATA             ICH/'NTC'/
!      .. Executable Statements ..
!      FULL = SNAME( 3: 3 ).EQ.'E'
!      BANDED = SNAME( 3: 3 ).EQ.'B'
!      Define the number of arguments.
!      IF( FULL )THEN
!          NARGS = 11
!      ELSE IF( BANDED )THEN
!          NARGS = 13
!      END IF
!
!      NC = 0
!      RESET = .TRUE.
!      ERRMAX = RZERO
!
!      DO 120 IN = 1, NIDIM
!          N = IDIM( IN )
!          ND = N/2 + 1
!
!          DO 110 IM = 1, 2
!              IF( IM.EQ.1 )&
!                  M = MAX( N - ND, 0 )
!              IF( IM.EQ.2 )&
!                  M = MIN( N + ND, NMAX )
!
!              IF( BANDED )THEN
!                  NK = NKB
!              ELSE
!                  NK = 1
!              END IF
!              DO 100 IKU = 1, NK
!                  IF( BANDED )THEN
!                      KU = KB( IKU )
!                      KL = MAX( KU - 1, 0 )
!                  ELSE
!                      KU = N - 1
!                      KL = M - 1
!                  END IF
!                  Set LDA to 1 more than minimum value if room.
!                  IF( BANDED )THEN
!                      LDA = KL + KU + 1
!                  ELSE
!                      LDA = M
!                  END IF
!                  IF( LDA.LT.NMAX )&
!                      LDA = LDA + 1
!                  Skip tests if not enough room.
!                  IF( LDA.GT.NMAX )&
!                      GO TO 100
!                  LAA = LDA*N
!                  NULL = N.LE.0.OR.M.LE.0
!
!                  Generate the matrix A.

```

```

TRANSL = ZERO
CALL ZMAKE( SNAME( 2: 3 ), ' ', ' ', M, N, A, NMAX, AA,&
LDA, KL, KU, RESET, TRANSL )

!
DO 90 IC = 1, 3
  TRANS = ICH( IC: IC )
  TRAN = TRANS.EQ.'T'.OR.TRANS.EQ.'C'

!
  IF( TRAN )THEN
    ML = N
    NL = M
  ELSE
    ML = M
    NL = N
  END IF

DO 80 IX = 1, NINC
  INCX = INC( IX )
  LX = ABS( INCX )*NL

  Generate the vector X.

  TRANSL = HALF
  CALL ZMAKE( 'GE', ' ', ' ', 1, NL, X, 1, XX,&
  ABS( INCX ), 0, NL - 1, RESET, TRANSL )
  IF( NL.GT.1 )THEN
    X( NL/2 ) = ZERO
    XX( 1 + ABS( INCX )*( NL/2 - 1 ) ) = ZERO
  END IF

DO 70 IY = 1, NINC
  INCY = INC( IY )
  LY = ABS( INCY )*ML

DO 60 IA = 1, NALF
  ALPHA = ALF( IA )

DO 50 IB = 1, NBET
  BETA = BET( IB )

  Generate the vector Y.

  TRANSL = ZERO
  CALL ZMAKE( 'GE', ' ', ' ', 1, ML, Y, 1,&
  YY, ABS( INCY ), 0, ML - 1,&
  RESET, TRANSL )

  NC = NC + 1

  Save every datum before calling the
  subroutine.

  TRANSS = TRANS
  MS = M
  NS = N
  KLS = KL
  KUS = KU
  ALS = ALPHA
  DO 10 I = 1, LAA
    AS( I ) = AA( I )
  CONTINUE
  LDAS = LDA
  DO 20 I = 1, LX
    XS( I ) = XX( I )
  CONTINUE
  INCXS = INCX
  BLS = BETA
  DO 30 I = 1, LY
    YS( I ) = YY( I )
  CONTINUE
  INCYS = INCY

  10
  20
  30
  Call the subroutine.

  IF( FULL )THEN
    IF( TRACE )&
      WRITE( NTRA, FMT = 9994 )NC, SNAME,&
      TRANS, M, N, ALPHA, LDA, INCX, BETA,&
      INCY
    IF( REWI )&
      REWIND NTRA
    CALL ZGEMV( TRANS, M, N, ALPHA, AA,&
      LDA, XX, INCX, BETA, YY,&
      INCY )
    ELSE IF( BANDED )THEN
      IF( TRACE )&
        WRITE( NTRA, FMT = 9995 )NC, SNAME,&
        TRANS, M, N, KL, KU, ALPHA, LDA,&
        INCX, BETA, INCY
      IF( REWI )&
        REWIND NTRA
      CALL ZGBMV( TRANS, M, N, KL, KU, ALPHA,&
        AA, LDA, XX, INCX, BETA,&
        YY, INCY )
    END IF

  Check if error-exit was taken incorrectly.

  IF( .NOT.OK )THEN
    WRITE( NOUT, FMT = 9993 )
    FATAL = .TRUE.
    GO TO 130
  END IF

  See what data changed inside subroutines.

  ISAME( 1 ) = TRANS.EQ.TRANS
  ISAME( 2 ) = MS.EQ.M
  ISAME( 3 ) = NS.EQ.N
  IF( FULL )THEN
    ISAME( 4 ) = ALS.EQ.ALPHA
    ISAME( 5 ) = LZE( AS, AA, LAA )
    ISAME( 6 ) = LDAS.EQ.LDA
    ISAME( 7 ) = LZE( XS, XX, LX )

```



```

9993 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
'*****' )
!
! End of ZCHK1.
!
END SUBROUTINE
SUBROUTINE ZCHK2( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
FATAL, NIDIM, IDIM, NKB, KB, NALF, ALF, NBET,&
BET, NINC, INC, NMAX, INCMAX, A, AA, AS, X, XX,&
XS, Y, YY, YS, YT, G )
!
! Tests ZHEMV, ZHBMV and ZHPMV.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
COMPLEX*16 ZERO, HALF
PARAMETER ( ZERO = ( 0.0D0, 0.0D0 ),&
HALF = ( 0.5D0, 0.0D0 ) )
DOUBLE PRECISION RZERO
PARAMETER ( RZERO = 0.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER INCMAX, NALF, NBET, NIDIM, NINC, NKB, NMAX,&
NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
! .. Array Arguments ..
COMPLEX*16 A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
AS( NMAX*NMAX ), BET( NBET ), X( NMAX ),&
XS( NMAX*INCMAX ), XX( NMAX*INCMAX ),&
Y( NMAX ), YS( NMAX*INCMAX ), YT( NMAX ),&
YY( NMAX*INCMAX )
DOUBLE PRECISION G( NMAX )
INTEGER IDIM( NIDIM ), INC( NINC ), KB( NKB )
! .. Local Scalars ..
COMPLEX*16 ALPHA, ALS, BETA, BLS, TRANSL
DOUBLE PRECISION ERR, ERRMAX
INTEGER I, IA, IB, IC, IK, IN, INCX, INCXS, INCY,&
INCY, IX, IY, K, KS, LAA, LDA, LDAS, LX, LY,&
N, NARGS, NC, NK, NS
LOGICAL BANDED, FULL, NULL, PACKED, RESET, SAME
CHARACTER*1 UPLO, UPLOS
CHARACTER*2 ICH
! .. Local Arrays ..
LOGICAL ISAME( 13 )
! .. External Functions ..
LOGICAL LZE, LZRES
EXTERNAL LZE, LZRES
! .. External Subroutines ..
EXTERNAL ZHEMV, ZHEMV, ZHPMV, ZMAKE, ZMVCH
! .. Intrinsic Functions ..
INTRINSIC ABS, MAX
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
DATA ICH/'UL'/
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'E'
BANDED = SNAME( 3: 3 ).EQ.'B'
PACKED = SNAME( 3: 3 ).EQ.'P'
! Define the number of arguments.
IF( FULL )THEN
NARGS = 10
ELSE IF( BANDED )THEN
NARGS = 11
ELSE IF( PACKED )THEN
NARGS = 9
END IF
!
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
DO 110 IN = 1, NIDIM
N = IDIM( IN )
!
IF( BANDED )THEN
NK = NKB
ELSE
NK = 1
END IF
DO 100 IK = 1, NK
IF( BANDED )THEN
K = KB( IK )
ELSE
K = N - 1
END IF
! Set LDA to 1 more than minimum value if room.
IF( BANDED )THEN
LDA = K + 1
ELSE
LDA = N
END IF
IF( LDA.LT.NMAX )&
LDA = LDA + 1
! Skip tests if not enough room.
IF( LDA.GT.NMAX )&
GO TO 100
IF( PACKED )THEN
LAA = ( N*( N + 1 ) )/2
ELSE
LAA = LDA*N
END IF
NULL = N.LE.0
!
DO 90 IC = 1, 2

```

```

!
!      UPLO = ICH( IC: IC )
!
!      Generate the matrix A.
!
      TRANSL = ZERO
      CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A, NMAX, AA, &
        LDA, K, K, RESET, TRANSL )
!
      DO 80 IX = 1, NINC
        INCX = INC( IX )
        LX = ABS( INCX ) * N
!
!      Generate the vector X.
!
      TRANSL = HALF
      CALL ZMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX, &
        ABS( INCX ), 0, N - 1, RESET, TRANSL )
      IF( N.GT.1 ) THEN
        X( N/2 ) = ZERO
        XX( 1 + ABS( INCX ) * ( N/2 - 1 ) ) = ZERO
      END IF
!
      DO 70 IY = 1, NINC
        INCY = INC( IY )
        LY = ABS( INCY ) * N
!
!      DO 60 IA = 1, NALF
!        ALPHA = ALF( IA )
!
!      DO 50 IB = 1, NBET
!        BETA = BET( IB )
!
!      Generate the vector Y.
!
      TRANSL = ZERO
      CALL ZMAKE( 'GE', ' ', ' ', 1, N, Y, 1, YY, &
        ABS( INCY ), 0, N - 1, RESET, &
        TRANSL )
!
      NC = NC + 1
!
!      Save every datum before calling the
!      subroutine.
!
      UPLOS = UPLO
      NS = N
      KS = K
      ALS = ALPHA
      DO 10 I = 1, LAA
        AS( I ) = AA( I )
      CONTINUE
      LDAS = LDA
      DO 20 I = 1, LX
        XS( I ) = XX( I )
      CONTINUE
      INCXS = INCX
      BLS = BETA
      DO 30 I = 1, LY
        YS( I ) = YY( I )
      CONTINUE
      INCYS = INCY
!
!      Call the subroutine.
!
      IF( FULL ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9993 ) NC, SNAME, &
            UPLO, N, ALPHA, LDA, INCX, BETA, INCY
        IF( REWI ) &
          REWIND NTRA
        CALL ZHEMV( UPLO, N, ALPHA, AA, LDA, XX, &
          INCX, BETA, YY, INCY )
      ELSE IF( Banded ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9994 ) NC, SNAME, &
            UPLO, N, K, ALPHA, LDA, INCX, BETA, &
            INCY
        IF( REWI ) &
          REWIND NTRA
        CALL ZHBMV( UPLO, N, K, ALPHA, AA, LDA, &
          XX, INCX, BETA, YY, INCY )
      ELSE IF( PACKED ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9995 ) NC, SNAME, &
            UPLO, N, ALPHA, INCX, BETA, INCY
        IF( REWI ) &
          REWIND NTRA
        CALL ZHPMV( UPLO, N, ALPHA, AA, XX, INCX, &
          BETA, YY, INCY )
      END IF
!
!      Check if error-exit was taken incorrectly.
!
      IF( .NOT.OK ) THEN
        WRITE( NOUT, FMT = 9992 )
        FATAL = .TRUE.
        GO TO 120
      END IF
!
!      See what data changed inside subroutines.
!
      ISAME( 1 ) = UPLO.EQ.UPLOS
      ISAME( 2 ) = NS.EQ.N
      IF( FULL ) THEN
        ISAME( 3 ) = ALS.EQ.ALPHA
        ISAME( 4 ) = LZE( AS, AA, LAA )
        ISAME( 5 ) = LDAS.EQ.LDA
        ISAME( 6 ) = LZE( XS, XX, LX )
        ISAME( 7 ) = INCXS.EQ.INCX
        ISAME( 8 ) = BLS.EQ.BETA
        IF( NULL ) THEN
          ISAME( 9 ) = LZE( YS, YY, LY )
        ELSE
          ISAME( 9 ) = LZERES( 'GE', ' ', 1, N, &

```

```

                                YS, YY, ABS( INCY ) )
                                END IF
                                ISAME( 10 ) = INCYS.EQ.INCY
                                ELSE IF( Banded )THEN
                                ISAME( 3 ) = KS.EQ.K
                                ISAME( 4 ) = ALS.EQ.ALPHA
                                ISAME( 5 ) = LZE( AS, AA, LAA )
                                ISAME( 6 ) = LDAS.EQ.LDA
                                ISAME( 7 ) = LZE( XS, XX, LX )
                                ISAME( 8 ) = INCXS.EQ.INCX
                                ISAME( 9 ) = BLS.EQ.BETA
                                IF( NULL )THEN
                                ISAME( 10 ) = LZE( YS, YY, LY )
                                ELSE
                                ISAME( 10 ) = LZERES( 'GE', ' ', 1, N,&
                                YS, YY, ABS( INCY ) )
                                END IF
                                ISAME( 11 ) = INCYS.EQ.INCY
                                ELSE IF( Packed )THEN
                                ISAME( 3 ) = ALS.EQ.ALPHA
                                ISAME( 4 ) = LZE( AS, AA, LAA )
                                ISAME( 5 ) = LZE( XS, XX, LX )
                                ISAME( 6 ) = INCXS.EQ.INCX
                                ISAME( 7 ) = BLS.EQ.BETA
                                IF( NULL )THEN
                                ISAME( 8 ) = LZE( YS, YY, LY )
                                ELSE
                                ISAME( 8 ) = LZERES( 'GE', ' ', 1, N,&
                                YS, YY, ABS( INCY ) )
                                END IF
                                ISAME( 9 ) = INCYS.EQ.INCY
                                END IF
!
!                               If data was incorrectly changed, report and
!                               return.
!
                                SAME = .TRUE.
                                DO 40 I = 1, NARGS
                                SAME = SAME.AND.ISAME( I )
                                IF( .NOT.ISAME( I ) )&
                                WRITE( NOUT, FMT = 9998 )I
                                40 CONTINUE
                                IF( .NOT.SAME )THEN
                                FATAL = .TRUE.
                                GO TO 120
                                END IF
!
!                               IF( .NOT.NULL )THEN
!
!                               Check the result.
!
                                CALL ZMVCH( 'N', N, N, ALPHA, A, NMAX, X,&
                                INCX, BETA, Y, INCY, YT, G,&
                                YY, EPS, ERR, FATAL, NOUT,&
                                .TRUE. )
                                ERRMAX = MAX( ERRMAX, ERR )
                                If got really bad answer, report and
                                return.
                                IF( FATAL )&
                                GO TO 120
                                ELSE
                                Avoid repeating tests with N.le.0
                                GO TO 110
                                END IF
!
!                               50 CONTINUE
!
!                               60 CONTINUE
!
!                               70 CONTINUE
!
!                               80 CONTINUE
!
!                               90 CONTINUE
!
!                               100 CONTINUE
!
!                               110 CONTINUE
!
!                               Report result.
!
                                IF( ERRMAX.LT.THRESH )THEN
                                WRITE( NOUT, FMT = 9999 )SNAME, NC
                                ELSE
                                WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
                                END IF
                                GO TO 130
!
!                               120 CONTINUE
!                               WRITE( NOUT, FMT = 9996 )SNAME
!                               IF( FULL )THEN
!                               WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, N, ALPHA, LDA, INCX,&
!                               BETA, INCY
!                               ELSE IF( Banded )THEN
!                               WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, N, K, ALPHA, LDA,&
!                               INCX, BETA, INCY
!                               ELSE IF( Packed )THEN
!                               WRITE( NOUT, FMT = 9995 )NC, SNAME, UPLO, N, ALPHA, INCX,&
!                               BETA, INCY
!                               END IF
!
!                               130 CONTINUE
!                               RETURN
!
!                               9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!                               'S)' )
!                               9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!                               'ANGED INCORRECTLY *****' )
!                               9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!                               'ALLS)', '/' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!                               ' - SUSPECT *****' )
!                               9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
!                               9995 FORMAT( 1X, I6, ': ', A6, '(', I1, ', ', I1, ', ', I3, ', ', I3, ', ', F4.1, ', ',&
!                               F4.1, ', ', AP, X, ', ', I2, ', ', I2, ', ', F4.1, ', ', F4.1, ', ', Y, ', ', I2,&

```

```

          ' )
          . ' )
9994 FORMAT( 1X, I6, ': ', A6, '(', A1, '(', I3, ' ', ' ', '(', 'I', &
          F4.1, ' ', ' ', F4.1, ' ', A, ' ', I3, ' ', X, ' ', I2, ' ', ' ', F4.1, ' ', ' ', &
          F4.1, ' ', Y, ' ', I2, ' ' )
9993 FORMAT( 1X, I6, ': ', A6, '(', A1, '(', I3, ' ', ' ', F4.1, ' ', ' ', &
          F4.1, ' ', A, ' ', I3, ' ', X, ' ', I2, ' ', ' ', F4.1, ' ', ' ', F4.1, ' ', ' ', &
          'Y', I2, ' ' )
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *', &
          '*****' )
!
! End of ZCHK2.
!
END SUBROUTINE
SUBROUTINE ZCHK3( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, &
          FATAL, NIDIM, IDIM, NKB, KB, NINC, INC, NMAX, &
          INCMAX, A, AA, AS, X, XX, XS, XT, G, Z )
!
! Tests ZTRMV, ZTBMV, ZTPMV, ZTRSV, ZTBSV and ZTPSV.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
COMPLEX*16      ZERO, HALF, ONE
PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ), &
          HALF = ( 0.5D0, 0.0D0 ), &
          ONE = ( 1.0D0, 0.0D0 ) )
DOUBLE PRECISION RZERO
PARAMETER      ( RZERO = 0.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER        INCMAX, NIDIM, NINC, NKB, NMAX, NOUT, NTRA
LOGICAL        FATAL, REWI, TRACE
CHARACTER*6    SNAME
! .. Array Arguments ..
COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), &
          AS( NMAX*NMAX ), X( NMAX ), XS( NMAX*INCMAX ), &
          XT( NMAX ), XX( NMAX*INCMAX ), Z( NMAX )
DOUBLE PRECISION G( NMAX )
INTEGER        IDIM( NIDIM ), INC( NINC ), KB( NKB )
! .. Local Scalars ..
COMPLEX*16      TRANSL
DOUBLE PRECISION ERR, ERRMAX
INTEGER        I, ICD, ICT, ICU, IK, IN, INCX, INCXS, IX, K, &
          KS, LAA, LDA, LDAS, LX, N, NARGS, NC, NK, NS
LOGICAL        BANDED, FULL, NULL, PACKED, RESET, SAME
CHARACTER*1    DIAG, DIAGS, TRANS, TRANSS, UPLO, UPLOS
CHARACTER*2    ICHD, ICHU
CHARACTER*3    ICHT
! .. Local Arrays ..
LOGICAL        ISAME( 13 )
! .. External Functions ..
LOGICAL        LZE, LZERES
EXTERNAL       LZE, LZERES
! .. External Subroutines ..
EXTERNAL       ZMAKE, ZMVCH, ZTBMV, ZTBSV, ZTPMV, ZTPSV, &
          ZTRMV, ZTRSV
! .. Intrinsic Functions ..
INTRINSIC      ABS, MAX
! .. Scalars in Common ..
INTEGER        INFOT, NOUTC
LOGICAL        LERR, OK
! .. Common blocks ..
COMMON         /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
DATA          ICHU/'UL'/, ICHT/'NTC'/, ICHD/'UN'/
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'R'
BANDED = SNAME( 3: 3 ).EQ.'B'
PACKED = SNAME( 3: 3 ).EQ.'P'
! Define the number of arguments.
IF( FULL )THEN
    NARGS = 8
ELSE IF( BANDED )THEN
    NARGS = 9
ELSE IF( PACKED )THEN
    NARGS = 7
END IF
!
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
! Set up zero vector for ZMVCH.
DO 10 I = 1, NMAX
    Z( I ) = ZERO
10 CONTINUE
!
DO 110 IN = 1, NIDIM
    N = IDIM( IN )
!
    IF( BANDED )THEN
        NK = NKB
    ELSE
        NK = 1
    END IF
    DO 100 IK = 1, NK
        IF( BANDED )THEN
            K = KB( IK )
        ELSE
            K = N - 1
        END IF
        ! Set LDA to 1 more than minimum value if room.
        IF( BANDED )THEN
            LDA = K + 1
        ELSE
            LDA = N
        END IF
        IF( LDA.LT.NMAX )&
            LDA = LDA + 1
        ! Skip tests if not enough room.
        IF( LDA.GT.NMAX )&

```

```

GO TO 100
IF( PACKED )THEN
  LAA = ( N*( N + 1 ) )/2
ELSE
  LAA = LDA*N
END IF
NULL = N.LE.0

!
DO 90 ICU = 1, 2
  UPLO = ICHU( ICU: ICU )
!
  DO 80 ICT = 1, 3
    TRANS = ICHT( ICT: ICT )
!
    DO 70 ICD = 1, 2
      DIAG = ICHD( ICD: ICD )
!
      Generate the matrix A.

      TRANSL = ZERO
      CALL ZMAKE( SNAME( 2: 3 ), UPLO, DIAG, N, N, A,&
        NMAX, AA, LDA, K, K, RESET, TRANSL )
!
      DO 60 IX = 1, NINC
        INCX = INC( IX )
        LX = ABS( INCX )*N
!
        Generate the vector X.

        TRANSL = HALF
        CALL ZMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX,&
          ABS( INCX ), 0, N - 1, RESET,&
          TRANSL )
        IF( N.GT.1 )THEN
          X( N/2 ) = ZERO
          XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
        END IF
!
        NC = NC + 1
!
        Save every datum before calling the subroutine.
!
        UPLOS = UPLO
        TRANSS = TRANS
        DIAGS = DIAG
        NS = N
        KS = K
        DO 20 I = 1, LAA
          AS( I ) = AA( I )
20      CONTINUE
        LDAS = LDA
        DO 30 I = 1, LX
          XS( I ) = XX( I )
30      CONTINUE
        INCXS = INCX
!
        Call the subroutine.
!
        IF( SNAME( 4: 5 ).EQ.'MV' )THEN
          IF( FULL )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9993 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, LDA, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTRMV( UPLO, TRANS, DIAG, N, AA, LDA,&
              XX, INCX )
          ELSE IF( Banded )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9994 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, K, LDA, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTBMV( UPLO, TRANS, DIAG, N, K, AA,&
              LDA, XX, INCX )
          ELSE IF( PACKED )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9995 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTPMV( UPLO, TRANS, DIAG, N, AA, XX,&
              INCX )
          END IF
        ELSE IF( SNAME( 4: 5 ).EQ.'SV' )THEN
          IF( FULL )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9993 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, LDA, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTRSV( UPLO, TRANS, DIAG, N, AA, LDA,&
              XX, INCX )
          ELSE IF( Banded )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9994 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, K, LDA, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTBSV( UPLO, TRANS, DIAG, N, K, AA,&
              LDA, XX, INCX )
          ELSE IF( PACKED )THEN
            IF( TRACE )&
              WRITE( NTRA, FMT = 9995 )NC, SNAME,&
                UPLO, TRANS, DIAG, N, INCX
            IF( REWI )&
              REWIND NTRA
            CALL ZTPSV( UPLO, TRANS, DIAG, N, AA, XX,&
              INCX )
          END IF
        END IF
!
        Check if error-exit was taken incorrectly.
!

```

```

IF( .NOT.OK )THEN
  WRITE( NOUT, FMT = 9992 )
  FATAL = .TRUE.
  GO TO 120
END IF

!
! See what data changed inside subroutines.
!
ISAME( 1 ) = UPLO.EQ.UPLOS
ISAME( 2 ) = TRANS.EQ.TRANS
ISAME( 3 ) = DIAG.EQ.DIAGS
ISAME( 4 ) = NS.EQ.N
IF( FULL )THEN
  ISAME( 5 ) = LZE( AS, AA, LAA )
  ISAME( 6 ) = LDAS.EQ.LDA
  IF( NULL )THEN
    ISAME( 7 ) = LZE( XS, XX, LX )
  ELSE
    ISAME( 7 ) = LZERES( 'GE', ' ', 1, N, XS, &
      XX, ABS( INCX ) )
  END IF
  ISAME( 8 ) = INCXS.EQ.INCX
ELSE IF( Banded )THEN
  ISAME( 5 ) = KS.EQ.K
  ISAME( 6 ) = LZE( AS, AA, LAA )
  ISAME( 7 ) = LDAS.EQ.LDA
  IF( NULL )THEN
    ISAME( 8 ) = LZE( XS, XX, LX )
  ELSE
    ISAME( 8 ) = LZERES( 'GE', ' ', 1, N, XS, &
      XX, ABS( INCX ) )
  END IF
  ISAME( 9 ) = INCXS.EQ.INCX
ELSE IF( Packed )THEN
  ISAME( 5 ) = LZE( AS, AA, LAA )
  IF( NULL )THEN
    ISAME( 6 ) = LZE( XS, XX, LX )
  ELSE
    ISAME( 6 ) = LZERES( 'GE', ' ', 1, N, XS, &
      XX, ABS( INCX ) )
  END IF
  ISAME( 7 ) = INCXS.EQ.INCX
END IF

!
! If data was incorrectly changed, report and
! return.
!
SAME = .TRUE.
DO 40 I = 1, NARGS
  SAME = SAME.AND.ISAME( I )
  IF( .NOT.ISAME( I ) )&
    WRITE( NOUT, FMT = 9998 ) I
CONTINUE
40 IF( .NOT.SAME )THEN
  FATAL = .TRUE.
  GO TO 120
END IF

!
! IF( .NOT.NULL )THEN
!   IF( SNAME( 4: 5 ).EQ.'MV' )THEN
!
!     Check the result.
!
!     CALL ZMVCH( TRANS, N, N, ONE, A, NMAX, X, &
!       INCX, ZERO, Z, INCX, XT, G, &
!       XX, EPS, ERR, FATAL, NOUT, &
!       .TRUE. )
!     ELSE IF( SNAME( 4: 5 ).EQ.'SV' )THEN
!
!       Compute approximation to original vector.
!
!       DO 50 I = 1, N
!         Z( I ) = XX( 1 + ( I - 1 ) ) * &
!           ABS( INCX ) )
!         XX( 1 + ( I - 1 ) ) * ABS( INCX ) ) &
!           = X( I )
!       CONTINUE
!       CALL ZMVCH( TRANS, N, N, ONE, A, NMAX, Z, &
!         INCX, ZERO, X, INCX, XT, G, &
!         XX, EPS, ERR, FATAL, NOUT, &
!         .FALSE. )
!     END IF
!     ERRMAX = MAX( ERRMAX, ERR )
!     If got really bad answer, report and return.
!     IF( FATAL )&
!       GO TO 120
!   ELSE
!     Avoid repeating tests with N.le.0.
!     GO TO 110
!   END IF
!
! 60 CONTINUE
!
! 70 CONTINUE
!
! 80 CONTINUE
!
! 90 CONTINUE
!
! 100 CONTINUE
!
! 110 CONTINUE
!
! Report result.
!
IF( ERRMAX.LT.THRESH )THEN
  WRITE( NOUT, FMT = 9999 ) SNAME, NC
ELSE
  WRITE( NOUT, FMT = 9997 ) SNAME, NC, ERRMAX
END IF
GO TO 130

!
120 CONTINUE
  WRITE( NOUT, FMT = 9996 ) SNAME

```

```

IF( FULL )THEN
  WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, TRANS, DIAG, N, LDA, &
  INCX
ELSE IF( Banded )THEN
  WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, TRANS, DIAG, N, K, &
  LDA, INCX
ELSE IF( Packed )THEN
  WRITE( NOUT, FMT = 9995 )NC, SNAME, UPLO, TRANS, DIAG, N, INCX
END IF
!
130 CONTINUE
RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL', &
'S') )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH', &
'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C', &
'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2, &
' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ), I3, ' ', AP, ' ', &
'X', ' ', I2, ' )' )
9994 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ), 2( I3, ' ', ' ), &
' A', ' ', I3, ' ', X, ' ', I2, ' )' )
9993 FORMAT( 1X, I6, ': ', A6, '( ', 3( ' ', A1, ' ', ' ), I3, ' ', A, ' ', &
I3, ' ', X, ' ', I2, ' )' )
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *', &
'*****' )
!
! End of ZCHK3.
!
END SUBROUTINE
SUBROUTINE ZCHK4( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, &
FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX, &
INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G, &
Z )
!
! Tests ZGERC and ZGERU.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
COMPLEX*16 ZERO, HALF, ONE
PARAMETER
( ZERO = ( 0.0D0, 0.0D0 ), &
HALF = ( 0.5D0, 0.0D0 ), &
ONE = ( 1.0D0, 0.0D0 ) )
DOUBLE PRECISION RZERO
PARAMETER ( RZERO = 0.0D0 )
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
! .. Array Arguments ..
COMPLEX*16 A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), &
AS( NMAX*NMAX ), X( NMAX ), XS( NMAX*INCMAX ), &
XX( NMAX*INCMAX ), Y( NMAX ), &
YS( NMAX*INCMAX ), YT( NMAX ), &
YY( NMAX*INCMAX ), Z( NMAX )
DOUBLE PRECISION G( NMAX )
INTEGER IDIM( NIDIM ), INC( NINC )
! .. Local Scalars ..
COMPLEX*16 ALPHA, ALS, TRANSL
DOUBLE PRECISION ERR, ERRMAX
INTEGER I, IA, IM, IN, INCX, INCXS, INCY, INCYS, IX, &
IY, J, LAA, LDA, LDAS, LX, LY, M, MS, N, NARGS, &
NC, ND, NS
LOGICAL CONJ, NULL, RESET, SAME
! .. Local Arrays ..
COMPLEX*16 W( 1 )
LOGICAL ISAME( 13 )
! .. External Functions ..
LOGICAL LZE, LZERS
EXTERNAL LZE, LZERS
! .. External Subroutines ..
EXTERNAL ZGERC, ZGERU, ZMAKE, ZMVCH
! .. Intrinsic Functions ..
INTRINSIC ABS, DCONJG, MAX, MIN
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
! .. Executable Statements ..
CONJ = SNAME( 5:5 ).EQ.'C'
! Define the number of arguments.
NARGS = 9
!
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
DO 120 IN = 1, NIDIM
  N = IDIM( IN )
  ND = N/2 + 1
!
  DO 110 IM = 1, 2
    IF( IM.EQ.1 )&
      M = MAX( N - ND, 0 )
    IF( IM.EQ.2 )&
      M = MIN( N + ND, NMAX )
!
    Set LDA to 1 more than minimum value if room.
    LDA = M
    IF( LDA.LT.NMAX )&
      LDA = LDA + 1
! Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
      GO TO 110

```

```

LAA = LDA*N
NULL = N.LE.0.OR.M.LE.0
!
DO 100 IX = 1, NINC
  INCX = INC( IX )
  LX = ABS( INCX )*M
!
!
  Generate the vector X.
!
  TRANSL = HALF
  CALL ZMAKE( 'GE', ' ', ' ', ' ', 1, M, X, 1, XX, ABS( INCX ), &
    0, M - 1, RESET, TRANSL )
  IF( M.GT.1 )THEN
    X( M/2 ) = ZERO
    XX( 1 + ABS( INCX )*( M/2 - 1 ) ) = ZERO
  END IF
!
DO 90 IY = 1, NINC
  INCY = INC( IY )
  LY = ABS( INCY )*N
!
!
  Generate the vector Y.
!
  TRANSL = ZERO
  CALL ZMAKE( 'GE', ' ', ' ', ' ', 1, N, Y, 1, YY, &
    ABS( INCY ), 0, N - 1, RESET, TRANSL )
  IF( N.GT.1 )THEN
    Y( N/2 ) = ZERO
    YY( 1 + ABS( INCY )*( N/2 - 1 ) ) = ZERO
  END IF
!
DO 80 IA = 1, NALF
  ALPHA = ALF( IA )
!
!
  Generate the matrix A.
!
  TRANSL = ZERO
  CALL ZMAKE( SNAME( 2: 3 ), ' ', ' ', M, N, A, NMAX, &
    AA, LDA, M - 1, N - 1, RESET, TRANSL )
!
NC = NC + 1
!
!
  Save every datum before calling the subroutine.
!
  MS = M
  NS = N
  ALS = ALPHA
  DO 10 I = 1, LAA
    AS( I ) = AA( I )
10  CONTINUE
  LDAS = LDA
  DO 20 I = 1, LX
    XS( I ) = XX( I )
20  CONTINUE
  INCXS = INCX
  DO 30 I = 1, LY
    YS( I ) = YY( I )
30  CONTINUE
  INCYS = INCY
!
!
  Call the subroutine.
!
  IF( TRACE )&
    WRITE( NTRA, FMT = 9994 )NC, SNAME, M, N, &
      ALPHA, INCX, INCY, LDA
  IF( CONJ )THEN
    IF( REWI )&
      REWIND NTRA
    CALL ZGERC( M, N, ALPHA, XX, INCX, YY, INCY, AA, &
      LDA )
  ELSE
    IF( REWI )&
      REWIND NTRA
    CALL ZGERU( M, N, ALPHA, XX, INCX, YY, INCY, AA, &
      LDA )
  END IF
!
!
  Check if error-exit was taken incorrectly.
!
  IF( .NOT.OK )THEN
    WRITE( NOUT, FMT = 9993 )
    FATAL = .TRUE.
    GO TO 140
  END IF
!
!
  See what data changed inside subroutine.
!
  ISAME( 1 ) = MS.EQ.M
  ISAME( 2 ) = NS.EQ.N
  ISAME( 3 ) = ALS.EQ.ALPHA
  ISAME( 4 ) = LZE( XS, XX, LX )
  ISAME( 5 ) = INCXS.EQ.INCX
  ISAME( 6 ) = LZE( YS, YY, LY )
  ISAME( 7 ) = INCYS.EQ.INCY
  IF( NULL )THEN
    ISAME( 8 ) = LZE( AS, AA, LAA )
  ELSE
    ISAME( 8 ) = LZERS( 'GE', ' ', M, N, AS, AA, &
      LDA )
  END IF
  ISAME( 9 ) = LDAS.EQ.LDA
!
!
  If data was incorrectly changed, report and return.
!
  SAME = .TRUE.
  DO 40 I = 1, NARGS
    SAME = SAME.AND.ISAME( I )
    IF( .NOT.ISAME( I ) )&
      WRITE( NOUT, FMT = 9998 )I
40  CONTINUE
  IF( .NOT.SAME )THEN
    FATAL = .TRUE.
    GO TO 140
  END IF

```

```

!
!       IF( .NOT.NULL )THEN
!
!           Check the result column by column.
!
!           IF( INCX.GT.0 )THEN
!               DO 50 I = 1, M
!                   Z( I ) = X( I )
!                   CONTINUE
!           ELSE
!               DO 60 I = 1, M
!                   Z( I ) = X( M - I + 1 )
!                   CONTINUE
!           END IF
!           DO 70 J = 1, N
!               IF( INCY.GT.0 )THEN
!                   W( 1 ) = Y( J )
!               ELSE
!                   W( 1 ) = Y( N - J + 1 )
!               END IF
!               IF( CONJ )&
!                   W( 1 ) = DCONJG( W( 1 ) )
!               CALL ZMVCH( 'N', M, 1, ALPHA, Z, NMAX, W, 1,&
!                   ONE, A( 1, J ), 1, YT, G,&
!                   AA( 1 + ( J - 1 ) * LDA ), EPS,&
!                   ERR, FATAL, NOUT, .TRUE. )
!               ERRMAX = MAX( ERRMAX, ERR )
!               If got really bad answer, report and return.
!               IF( FATAL )&
!                   GO TO 130
!           CONTINUE
!       70
!       ELSE
!           Avoid repeating tests with M.le.0 or N.le.0.
!           GO TO 110
!       END IF
!
!       80
!       CONTINUE
!
!       90
!       CONTINUE
!
!       100
!       CONTINUE
!
!       110
!       CONTINUE
!
!       120
!       CONTINUE
!
!       Report result.
!
!       IF( ERRMAX.LT.THRESH )THEN
!           WRITE( NOUT, FMT = 9999 )SNAME, NC
!       ELSE
!           WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
!       END IF
!       GO TO 150
!
!       130
!       CONTINUE
!       WRITE( NOUT, FMT = 9995 )J
!
!       140
!       CONTINUE
!       WRITE( NOUT, FMT = 9996 )SNAME
!       WRITE( NOUT, FMT = 9994 )NC, SNAME, M, N, ALPHA, INCX, INCY, LDA
!
!       150
!       CONTINUE
!       RETURN
!
!       9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!           'S)' )
!       9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!           'ANGED INCORRECTLY *****' )
!       9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!           'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!           ' - SUSPECT *****' )
!       9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
!       9995 FORMAT( ' THESE ARE THE RESULTS FOR COLUMN ', I3 )
!       9994 FORMAT( 1X, I6, ': ', A6, '( ', 2( I3, ' ', ' ), '( ', F4.1, ' ', ' ', F4.1,&
!           ' ), X, ', I2, ' ', Y, ', I2, ' ', A, ', I3, ' ',
!           ' ', ' )
!       9993 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
!           '*****' )
!
!       End of ZCHK4.
!
!       END SUBROUTINE
!       SUBROUTINE ZCHK5( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!           FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX,&
!           INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G,&
!           Z )
!
!       Tests ZHER and ZHPR.
!
!       Auxiliary routine for test program for Level 2 Blas.
!
!       -- Written on 10-August-1987.
!       Richard Hanson, Sandia National Labs.
!       Jeremy Du Croz, NAG Central Office.
!
!       .. Parameters ..
!       COMPLEX*16
!       PARAMETER
!           ( ZERO = ( 0.0D0, 0.0D0 ),&
!             HALF = ( 0.5D0, 0.0D0 ),&
!             ONE = ( 1.0D0, 0.0D0 ) )
!       DOUBLE PRECISION
!       RZERO
!       PARAMETER
!           ( RZERO = 0.0D0 )
!       .. Scalar Arguments ..
!       DOUBLE PRECISION
!       EPS, THRESH
!       INTEGER
!       INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
!       LOGICAL
!       FATAL, REWI, TRACE
!       CHARACTER*6
!       SNAME
!       .. Array Arguments ..
!       COMPLEX*16
!       A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
!       AS( NMAX*NMAX ), X( NMAX ), XS( NMAX*INCMAX ),&
!       XX( NMAX*INCMAX ), Y( NMAX ),&
!       YS( NMAX*INCMAX ), YT( NMAX ),&
!       YY( NMAX*INCMAX ), Z( NMAX )

```

```

DOUBLE PRECISION  G( NMAX )
INTEGER            IDIM( NIDIM ), INC( NINC )
!
! .. Local Scalars ..
COMPLEX*16         ALPHA, TRANSL
DOUBLE PRECISION  ERR, ERRMAX, RALPHA, RALS
INTEGER            I, IA, IC, IN, INCX, INCXS, IX, J, JA, JJ, LAA, &
!                   LDA, LDAS, LJ, LX, N, NARGS, NC, NS
LOGICAL            FULL, NULL, PACKED, RESET, SAME, UPPER
CHARACTER*1        UPLO, UPLOS
CHARACTER*2        ICH
!
! .. Local Arrays ..
COMPLEX*16         W( 1 )
LOGICAL            ISAME( 13 )
!
! .. External Functions ..
LOGICAL            LZE, LZERES
!
! .. External Subroutines ..
EXTERNAL           ZHER, ZHPR, ZMAKE, ZMVCH
!
! .. Intrinsic Functions ..
INTRINSIC          ABS, DBLE, DCMPLX, DCONJG, MAX
!
! .. Scalars in Common ..
INTEGER            INFOT, NOUTC
LOGICAL            LERR, OK
!
! .. Common blocks ..
COMMON             /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA              ICH/'UL'/
!
! .. Executable Statements ..
FULL = SNAME( 3: 3 ).EQ.'E'
PACKED = SNAME( 3: 3 ).EQ.'P'
!
! Define the number of arguments.
IF( FULL )THEN
    NARGS = 7
ELSE IF( PACKED )THEN
    NARGS = 6
END IF
!
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
DO 100 IN = 1, NIDIM
    N = IDIM( IN )
    ! Set LDA to 1 more than minimum value if room.
    LDA = N
    IF( LDA.LT.NMAX )&
        LDA = LDA + 1
    ! Skip tests if not enough room.
    IF( LDA.GT.NMAX )&
        GO TO 100
    IF( PACKED )THEN
        LAA = ( N*( N + 1 ) )/2
    ELSE
        LAA = LDA*N
    END IF
!
DO 90 IC = 1, 2
    UPLO = ICH( IC: IC )
    UPPER = UPLO.EQ.'U'
!
DO 80 IX = 1, NINC
    INCX = INC( IX )
    LX = ABS( INCX )*N
!
    Generate the vector X.
!
    TRANSL = HALF
    CALL ZMAKE( 'GE', ' ', ' ', 1, N, X, 1, XX, ABS( INCX ),&
        0, N - 1, RESET, TRANSL )
    IF( N.GT.1 )THEN
        X( N/2 ) = ZERO
        XX( 1 + ABS( INCX )*( N/2 - 1 ) ) = ZERO
    END IF
!
DO 70 IA = 1, NALF
    RALPHA = DBLE( ALF( IA ) )
    ALPHA = DCMPLX( RALPHA, RZERO )
    NULL = N.LE.0.OR.RALPHA.EQ.RZERO
!
    Generate the matrix A.
!
    TRANSL = ZERO
    CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A, NMAX,&
        AA, LDA, N - 1, N - 1, RESET, TRANSL )
!
    NC = NC + 1
!
    Save every datum before calling the subroutine.
!
    UPLOS = UPLO
    NS = N
    RALS = RALPHA
    DO 10 I = 1, LAA
        AS( I ) = AA( I )
10    CONTINUE
    LDAS = LDA
    DO 20 I = 1, LX
        XS( I ) = XX( I )
20    CONTINUE
    INCXS = INCX
!
    Call the subroutine.
!
    IF( FULL )THEN
        IF( TRACE )&
            WRITE( NTRA, FMT = 9993 )NC, SNAME, UPLO, N,&
                RALPHA, INCX, LDA
        IF( REWI )&
            REWIND NTRA
        CALL ZHER( UPLO, N, RALPHA, XX, INCX, AA, LDA )
    ELSE IF( PACKED )THEN
        IF( TRACE )&
            WRITE( NTRA, FMT = 9994 )NC, SNAME, UPLO, N,&
                RALPHA, INCX

```

```

        IF( REWI )&
            REWIND NTRA
        CALL ZHPR( UPLO, N, RALPHA, XX, INCX, AA )
    END IF

!
!   Check if error-exit was taken incorrectly.
!
    IF( .NOT.OK )THEN
        WRITE( NOUT, FMT = 9992 )
        FATAL = .TRUE.
        GO TO 120
    END IF

!
!   See what data changed inside subroutines.
!
    ISAME( 1 ) = UPLO.EQ.UPLOS
    ISAME( 2 ) = NS.EQ.N
    ISAME( 3 ) = RALS.EQ.RALPHA
    ISAME( 4 ) = LZE( XS, XX, LX )
    ISAME( 5 ) = INCXS.EQ.INCX
    IF( NULL )THEN
        ISAME( 6 ) = LZE( AS, AA, LAA )
    ELSE
        ISAME( 6 ) = LZERS( SNAME( 2: 3 ), UPLO, N, N, AS,&
            AA, LDA )
    END IF
    IF( .NOT.PACKED )THEN
        ISAME( 7 ) = LDAS.EQ.LDA
    END IF

!
!   If data was incorrectly changed, report and return.
!
    SAME = .TRUE.
    DO 30 I = 1, NARGS
        SAME = SAME.AND.ISAME( I )
        IF( .NOT.ISAME( I ) )&
            WRITE( NOUT, FMT = 9998 )I
    30 CONTINUE
    IF( .NOT.SAME )THEN
        FATAL = .TRUE.
        GO TO 120
    END IF

!
!   Check the result column by column.
!
    IF( INCX.GT.0 )THEN
        DO 40 I = 1, N
            Z( I ) = X( I )
    40 CONTINUE
    ELSE
        DO 50 I = 1, N
            Z( I ) = X( N - I + 1 )
    50 CONTINUE
    END IF
    JA = 1
    DO 60 J = 1, N
        W( 1 ) = DCONJG( Z( J ) )
        IF( UPPER )THEN
            JJ = 1
            LJ = J
        ELSE
            JJ = J
            LJ = N - J + 1
        END IF
        CALL ZMVCH( 'N', LJ, 1, ALPHA, Z( JJ ), LJ, W,&
            1, ONE, A( JJ, J ), 1, YT, G,&
            AA( JA ), EPS, ERR, FATAL, NOUT,&
            .TRUE. )
        IF( FULL )THEN
            IF( UPPER )THEN
                JA = JA + LDA
            ELSE
                JA = JA + LDA + 1
            END IF
        ELSE
            JA = JA + LJ
        END IF
        ERRMAX = MAX( ERRMAX, ERR )
        If got really bad answer, report and return.
        IF( FATAL )&
            GO TO 110
    60 CONTINUE
    ELSE
!
!   Avoid repeating tests if N.le.0.
!
        IF( N.LE.0 )&
            GO TO 100
    END IF

!
!   70 CONTINUE
!
!   80 CONTINUE
!
!   90 CONTINUE
!
!   100 CONTINUE
!
!   Report result.
!
    IF( ERRMAX.LT.THRESH )THEN
        WRITE( NOUT, FMT = 9999 )SNAME, NC
    ELSE
        WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
    END IF
    GO TO 130

!
!   110 CONTINUE
!       WRITE( NOUT, FMT = 9995 )J
!
!   120 CONTINUE
        WRITE( NOUT, FMT = 9996 )SNAME
        IF( FULL )THEN
            WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, N, RALPHA, INCX, LDA

```

```

        ELSE IF( PACKED ) THEN
            WRITE( NOUT, FMT = 9994 ) NC, SNAME, UPLO, N, RALPHA, INCX
        END IF
!
! 130 CONTINUE
        RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL', &
'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH', &
'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C', &
'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2, &
' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( '          THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '(', '', A1, '', I3, ', ', F4.1, ', X', &
I2, ', AP)
          ' )
9993 FORMAT( 1X, I6, ': ', A6, '(', '', A1, '', I3, ', ', F4.1, ', X', &
I2, ', A', I3, ' )
          ' )
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *', &
'*****' )
!
!      End of ZCHK5.
!
!
        END SUBROUTINE
        SUBROUTINE ZCHK6( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, &
FATAL, NIDIM, IDIM, NALF, ALF, NINC, INC, NMAX, &
INCMAX, A, AA, AS, X, XX, XS, Y, YY, YS, YT, G, &
Z )
!
! Tests ZHER2 and ZHPR2.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
        COMPLEX*16      ZERO, HALF, ONE
        PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ), &
HALF = ( 0.5D0, 0.0D0 ), &
ONE = ( 1.0D0, 0.0D0 ) )
        DOUBLE PRECISION RZERO
        PARAMETER      ( RZERO = 0.0D0 )
! .. Scalar Arguments ..
        DOUBLE PRECISION EPS, THRESH
        INTEGER          INCMAX, NALF, NIDIM, NINC, NMAX, NOUT, NTRA
        LOGICAL          FATAL, REWI, TRACE
        CHARACTER*6      SNAME
! .. Array Arguments ..
        COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), &
AS( NMAX*NMAX ), X( NMAX ), XS( NMAX*INCMAX ), &
XX( NMAX*INCMAX ), Y( NMAX ), &
YS( NMAX*INCMAX ), YT( NMAX ), &
YY( NMAX*INCMAX ), Z( NMAX, 2 )
        DOUBLE PRECISION G( NMAX )
        INTEGER          IDIM( NIDIM ), INC( NINC )
! .. Local Scalars ..
        COMPLEX*16      ALPHA, ALS, TRANSL
        DOUBLE PRECISION ERR, ERRMAX
        INTEGER          I, IA, IC, IN, INCX, INCXS, INCY, INCYS, IX, &
IY, J, JA, JJ, LAA, LDA, LDAS, LJ, LX, LY, N, &
NARGS, NC, NS
        LOGICAL          FULL, NULL, PACKED, RESET, SAME, UPPER
        CHARACTER*1      UPLO, UPLOS
        CHARACTER*2      ICH
! .. Local Arrays ..
        COMPLEX*16      W( 2 )
        LOGICAL          ISAME( 13 )
! .. External Functions ..
        LOGICAL          LZE, LZERS
        EXTERNAL          LZE, LZERS
! .. External Subroutines ..
        EXTERNAL          ZHER2, ZHPR2, ZMAKE, ZMVCH
! .. Intrinsic Functions ..
        INTRINSIC        ABS, DCONJG, MAX
! .. Scalars in Common ..
        INTEGER          INFOT, NOUTC
        LOGICAL          LERR, OK
! .. Common blocks ..
        COMMON            /INFOC/INFOT, NOUTC, OK, LERR
! .. Data statements ..
        DATA             ICH/'UL'/
! .. Executable Statements ..
        FULL = SNAME( 3: 3 ).EQ.'E'
        PACKED = SNAME( 3: 3 ).EQ.'P'
        Define the number of arguments.
        IF( FULL ) THEN
            NARGS = 9
        ELSE IF( PACKED ) THEN
            NARGS = 8
        END IF
!
        NC = 0
        RESET = .TRUE.
        ERRMAX = RZERO
!
        DO 140 IN = 1, NIDIM
            N = IDIM( IN )
            Set LDA to 1 more than minimum value if room.
            LDA = N
            IF( LDA.LT.NMAX ) &
                LDA = LDA + 1
            Skip tests if not enough room.
            IF( LDA.GT.NMAX ) &
                GO TO 140
            IF( PACKED ) THEN
                LAA = ( N*( N + 1 ) )/2
            ELSE
                LAA = LDA*N
            END IF

```

```

DO 130 IC = 1, 2
  UPLO = ICH( IC: IC )
  UPPER = UPLO.EQ.'U'

!
!
DO 120 IX = 1, NINC
  INCX = INC( IX )
  LX = ABS( INCX ) * N

!
!
  Generate the vector X.

  TRANSL = HALF
  CALL ZMAKE( 'GE', ' ', ' ', ' ', 1, N, X, 1, XX, ABS( INCX ), &
    0, N - 1, RESET, TRANSL )
  IF( N.GT.1 ) THEN
    X( N/2 ) = ZERO
    XX( 1 + ABS( INCX ) * ( N/2 - 1 ) ) = ZERO
  END IF

!
DO 110 IY = 1, NINC
  INCY = INC( IY )
  LY = ABS( INCY ) * N

!
!
  Generate the vector Y.

  TRANSL = ZERO
  CALL ZMAKE( 'GE', ' ', ' ', ' ', 1, N, Y, 1, YY, &
    ABS( INCY ), 0, N - 1, RESET, TRANSL )
  IF( N.GT.1 ) THEN
    Y( N/2 ) = ZERO
    YY( 1 + ABS( INCY ) * ( N/2 - 1 ) ) = ZERO
  END IF

!
DO 100 IA = 1, NALF
  ALPHA = ALF( IA )
  NULL = N.LE.0.OR.ALPHA.EQ.ZERO

!
!
  Generate the matrix A.

  TRANSL = ZERO
  CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, A, &
    NMAX, AA, LDA, N - 1, N - 1, RESET, &
    TRANSL )

!
NC = NC + 1

!
!
  Save every datum before calling the subroutine.

  UPLOS = UPLO
  NS = N
  ALS = ALPHA
  DO 10 I = 1, LAA
    AS( I ) = AA( I )
  CONTINUE
  LDAS = LDA
  DO 20 I = 1, LX
    XS( I ) = XX( I )
  CONTINUE
  INCXS = INCX
  DO 30 I = 1, LY
    YS( I ) = YY( I )
  CONTINUE
  INCYS = INCY

!
!
  Call the subroutine.

  IF( FULL ) THEN
    IF( TRACE ) &
      WRITE( NTRA, FMT = 9993 ) NC, SNAME, UPLO, N, &
        ALPHA, INCX, INCY, LDA
    IF( REWI ) &
      REWIND NTRA
    CALL ZHER2( UPLO, N, ALPHA, XX, INCX, YY, INCY, &
      AA, LDA )
  ELSE IF( PACKED ) THEN
    IF( TRACE ) &
      WRITE( NTRA, FMT = 9994 ) NC, SNAME, UPLO, N, &
        ALPHA, INCX, INCY
    IF( REWI ) &
      REWIND NTRA
    CALL ZHPR2( UPLO, N, ALPHA, XX, INCX, YY, INCY, &
      AA )
  END IF

!
!
  Check if error-exit was taken incorrectly.

  IF( .NOT.OK ) THEN
    WRITE( NOUT, FMT = 9992 )
    FATAL = .TRUE.
    GO TO 160
  END IF

!
!
  See what data changed inside subroutines.

  ISAME( 1 ) = UPLO.EQ.UPLOS
  ISAME( 2 ) = NS.EQ.N
  ISAME( 3 ) = ALS.EQ.ALPHA
  ISAME( 4 ) = LZE( XS, XX, LX )
  ISAME( 5 ) = INCXS.EQ.INCX
  ISAME( 6 ) = LZE( YS, YY, LY )
  ISAME( 7 ) = INCYS.EQ.INCY
  IF( NULL ) THEN
    ISAME( 8 ) = LZE( AS, AA, LAA )
  ELSE
    ISAME( 8 ) = LZERES( SNAME( 2: 3 ), UPLO, N, N, &
      AS, AA, LDA )
  END IF
  IF( .NOT.PACKED ) THEN
    ISAME( 9 ) = LDAS.EQ.LDA
  END IF

!
!
  If data was incorrectly changed, report and return.

  SAME = .TRUE.
  DO 40 I = 1, NARGS

```

```

        SAME = SAME.AND.ISAME( I )
        IF( .NOT.ISAME( I ) )&
            WRITE( NOUT, FMT = 9998 ) I
40      CONTINUE
        IF( .NOT.SAME )THEN
            FATAL = .TRUE.
            GO TO 160
        END IF

        IF( .NOT.NULL )THEN
            Check the result column by column.

            IF( INCX.GT.0 )THEN
                DO 50 I = 1, N
                    Z( I, 1 ) = X( I )
50              CONTINUE
            ELSE
                DO 60 I = 1, N
                    Z( I, 1 ) = X( N - I + 1 )
60              CONTINUE
            END IF
            IF( INCY.GT.0 )THEN
                DO 70 I = 1, N
                    Z( I, 2 ) = Y( I )
70              CONTINUE
            ELSE
                DO 80 I = 1, N
                    Z( I, 2 ) = Y( N - I + 1 )
80              CONTINUE
            END IF
            JA = 1
            DO 90 J = 1, N
                W( 1 ) = ALPHA*DCONJG( Z( J, 2 ) )
                W( 2 ) = DCONJG( ALPHA )*DCONJG( Z( J, 1 ) )
                IF( UPPER )THEN
                    JJ = 1
                    LJ = J
                ELSE
                    JJ = J
                    LJ = N - J + 1
                END IF
                CALL ZMVCH( 'N', LJ, 2, ONE, Z( JJ, 1 ),&
                    NMAX, W, 1, ONE, A( JJ, J ), 1,&
                    YT, G, AA( JA ), EPS, ERR, FATAL,&
                    NOUT, .TRUE. )
                IF( FULL )THEN
                    IF( UPPER )THEN
                        JA = JA + LDA
                    ELSE
                        JA = JA + LDA + 1
                    END IF
                ELSE
                    JA = JA + LJ
                END IF
                ERRMAX = MAX( ERRMAX, ERR )
                If got really bad answer, report and return.
                IF( FATAL )&
                    GO TO 150
90            CONTINUE
        ELSE
            Avoid repeating tests with N.le.0.
            IF( N.LE.0 )&
                GO TO 140
        END IF

100      CONTINUE
!
110      CONTINUE
!
120      CONTINUE
!
130      CONTINUE
!
140      CONTINUE
!
        Report result.
!
        IF( ERRMAX.LT.THRESH )THEN
            WRITE( NOUT, FMT = 9999 ) SNAME, NC
        ELSE
            WRITE( NOUT, FMT = 9997 ) SNAME, NC, ERRMAX
        END IF
        GO TO 170
!
150      CONTINUE
        WRITE( NOUT, FMT = 9995 ) J
!
160      CONTINUE
        WRITE( NOUT, FMT = 9996 ) SNAME
        IF( FULL )THEN
            WRITE( NOUT, FMT = 9993 ) NC, SNAME, UPLO, N, ALPHA, INCX,&
                INCY, LDA
        ELSE IF( PACKED )THEN
            WRITE( NOUT, FMT = 9994 ) NC, SNAME, UPLO, N, ALPHA, INCX, INCY
        END IF
!
170      CONTINUE
        RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
    'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
    'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
    'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
    ' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( '          THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '(', '', A1, '', I3, '(', F4.1, ', ',&
    F4.1, '), X, ', I2, ', Y, ', I2, ', A)',&
    ' ',&
    ' ' )
9993 FORMAT( 1X, I6, ': ', A6, '(', '', A1, '', I3, '(', F4.1, ', ',&
    F4.1, '), X, ', I2, ', Y, ', I2, ', A, ', I3, ' ',&
    ' ',&
    ' ' )

```



```

CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL ZHBMV( 'U', 0, 1, ALPHA, A, 1, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL ZHBMV( 'U', 0, 0, ALPHA, A, 1, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 11
CALL ZHBMV( 'U', 0, 0, ALPHA, A, 1, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
50 INFOT = 1
CALL ZHPMV( '/', 0, ALPHA, A, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZHPMV( 'U', -1, ALPHA, A, X, 1, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL ZHPMV( 'U', 0, ALPHA, A, X, 0, BETA, Y, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZHPMV( 'U', 0, ALPHA, A, X, 1, BETA, Y, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
60 INFOT = 1
CALL ZTRMV( '/', 'N', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTRMV( 'U', '/', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTRMV( 'U', 'N', '/', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTRMV( 'U', 'N', 'N', -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL ZTRMV( 'U', 'N', 'N', 2, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL ZTRMV( 'U', 'N', 'N', 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
70 INFOT = 1
CALL ZTBMV( '/', 'N', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTBMV( 'U', '/', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTBMV( 'U', 'N', '/', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTBMV( 'U', 'N', 'N', -1, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZTBMV( 'U', 'N', 'N', 0, -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZTBMV( 'U', 'N', 'N', 0, 1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZTBMV( 'U', 'N', 'N', 0, 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
80 INFOT = 1
CALL ZTPMV( '/', 'N', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTPMV( 'U', '/', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTPMV( 'U', 'N', '/', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTPMV( 'U', 'N', 'N', -1, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZTPMV( 'U', 'N', 'N', 0, A, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
90 INFOT = 1
CALL ZTRSV( '/', 'N', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTRSV( 'U', '/', 'N', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTRSV( 'U', 'N', '/', 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTRSV( 'U', 'N', 'N', -1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 6
CALL ZTRSV( 'U', 'N', 'N', 2, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 8
CALL ZTRSV( 'U', 'N', 'N', 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
100 INFOT = 1
CALL ZTBSV( '/', 'N', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTBSV( 'U', '/', 'N', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTBSV( 'U', 'N', '/', 0, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTBSV( 'U', 'N', 'N', -1, 0, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZTBSV( 'U', 'N', 'N', 0, -1, A, 1, X, 1 )

```

```

CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZTBSV( 'U', 'N', 'N', 0, 1, A, 1, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZTBSV( 'U', 'N', 'N', 0, 0, A, 1, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
110 INFOT = 1
CALL ZTBSV( '/', 'N', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZTBSV( 'U', '/', 'N', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZTBSV( 'U', 'N', '/', 0, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZTBSV( 'U', 'N', 'N', -1, A, X, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZTBSV( 'U', 'N', 'N', 0, A, X, 0 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
120 INFOT = 1
CALL ZGERC( -1, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZGERC( 0, -1, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGERC( 0, 0, ALPHA, X, 0, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZGERC( 0, 0, ALPHA, X, 1, Y, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZGERC( 2, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
130 INFOT = 1
CALL ZGERU( -1, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZGERU( 0, -1, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGERU( 0, 0, ALPHA, X, 0, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZGERU( 0, 0, ALPHA, X, 1, Y, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZGERU( 2, 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
140 INFOT = 1
CALL ZHER( '/', 0, RALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZHER( 'U', -1, RALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZHER( 'U', 0, RALPHA, X, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZHER( 'U', 2, RALPHA, X, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
150 INFOT = 1
CALL ZHPR( '/', 0, RALPHA, X, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZHPR( 'U', -1, RALPHA, X, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZHPR( 'U', 0, RALPHA, X, 0, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
160 INFOT = 1
CALL ZHER2( '/', 0, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZHER2( 'U', -1, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZHER2( 'U', 0, ALPHA, X, 0, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZHER2( 'U', 0, ALPHA, X, 1, Y, 0, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZHER2( 'U', 2, ALPHA, X, 1, Y, 1, A, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 180
170 INFOT = 1
CALL ZHPR2( '/', 0, ALPHA, X, 1, Y, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZHPR2( 'U', -1, ALPHA, X, 1, Y, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZHPR2( 'U', 0, ALPHA, X, 0, Y, 1, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZHPR2( 'U', 0, ALPHA, X, 1, Y, 0, A )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
180 IF( OK )THEN
    WRITE( NOUT, FMT = 9999 )SRNAMT
ELSE
    WRITE( NOUT, FMT = 9998 )SRNAMT
END IF
RETURN

```

```

!
9999 FORMAT( ' ', A6, ' PASSED THE TESTS OF ERROR-EXITS' )
9998 FORMAT( ' ***** ', A6, ' FAILED THE TESTS OF ERROR-EXITS *****', &
'***' )
!
! End of ZCHKE.
!
END SUBROUTINE
SUBROUTINE ZMAKE( TYPE, UPLO, DIAG, M, N, A, NMAX, AA, LDA, KL, &
KU, RESET, TRANSL )
!
! Generates values for an M by N matrix A within the bandwidth
! defined by KL and KU.
! Stores the values in the array AA in the data structure required
! by the routine, with unwanted elements set to rogue value.
!
! TYPE is 'GE', 'GB', 'HE', 'HB', 'HP', 'TR', 'TB' OR 'TP'.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Parameters ..
COMPLEX*16 ZERO, ONE
PARAMETER ( ZERO = ( 0.0D0, 0.0D0 ), &
ONE = ( 1.0D0, 0.0D0 ) )
COMPLEX*16 ROGUE
PARAMETER ( ROGUE = ( -1.0D10, 1.0D10 ) )
DOUBLE PRECISION RZERO
PARAMETER ( RZERO = 0.0D0 )
DOUBLE PRECISION RROGUE
PARAMETER ( RROGUE = -1.0D10 )
! .. Scalar Arguments ..
COMPLEX*16 TRANSL
INTEGER KL, KU, LDA, M, N, NMAX
LOGICAL RESET
CHARACTER*1 DIAG, UPLO
CHARACTER*2 TYPE
! .. Array Arguments ..
COMPLEX*16 A( NMAX, * ), AA( * )
! .. Local Scalars ..
INTEGER I, I1, I2, I3, IBEG, IEND, IOFF, J, JJ, KK
LOGICAL GEN, LOWER, SYM, TRI, UNIT, UPPER
! .. External Functions ..
COMPLEX*16 ZBEG
EXTERNAL ZBEG
! .. Intrinsic Functions ..
INTRINSIC DBLE, DCMPLX, DCONJG, MAX, MIN
! .. Executable Statements ..
GEN = TYPE( 1: 1 ).EQ. 'G'
SYM = TYPE( 1: 1 ).EQ. 'H'
TRI = TYPE( 1: 1 ).EQ. 'T'
UPPER = ( SYM.OR.TRI ).AND.UPLO.EQ. 'U'
LOWER = ( SYM.OR.TRI ).AND.UPLO.EQ. 'L'
UNIT = TRI.AND.DIAG.EQ. 'U'
!
! Generate data in array A.
!
DO 20 J = 1, N
DO 10 I = 1, M
IF( GEN.OR.( UPPER.AND.I.LE.J ).OR.( LOWER.AND.I.GE.J ) ) &
THEN
IF( ( I.LE.J.AND.J - I.LE.KU ).OR.&
( I.GE.J.AND.I - J.LE.KL ) ) THEN
A( I, J ) = ZBEG( RESET ) + TRANSL
ELSE
A( I, J ) = ZERO
END IF
IF( I.NE.J ) THEN
IF( SYM ) THEN
A( J, I ) = DCONJG( A( I, J ) )
ELSE IF( TRI ) THEN
A( J, I ) = ZERO
END IF
END IF
END IF
10 CONTINUE
IF( SYM ) &
A( J, J ) = DCMPLX( DBLE( A( J, J ) ), RZERO )
IF( TRI ) &
A( J, J ) = A( J, J ) + ONE
IF( UNIT ) &
A( J, J ) = ONE
20 CONTINUE
!
! Store elements in array AS in data structure required by routine.
!
IF( TYPE.EQ. 'GE' ) THEN
DO 50 J = 1, N
DO 30 I = 1, M
AA( I + ( J - 1 ) * LDA ) = A( I, J )
30 CONTINUE
DO 40 I = M + 1, LDA
AA( I + ( J - 1 ) * LDA ) = ROGUE
40 CONTINUE
50 CONTINUE
ELSE IF( TYPE.EQ. 'GB' ) THEN
DO 60 J = 1, N
DO 60 I1 = 1, KU + 1 - J
AA( I1 + ( J - 1 ) * LDA ) = ROGUE
60 CONTINUE
DO 70 I2 = I1, MIN( KL + KU + 1, KU + 1 + M - J )
AA( I2 + ( J - 1 ) * LDA ) = A( I2 + J - KU - 1, J )
70 CONTINUE
DO 80 I3 = I2, LDA
AA( I3 + ( J - 1 ) * LDA ) = ROGUE
80 CONTINUE
90 CONTINUE
ELSE IF( TYPE.EQ. 'HE'.OR.TYPE.EQ. 'TR' ) THEN
DO 130 J = 1, N
IF( UPPER ) THEN
IBEG = 1

```

```

        IF( UNIT ) THEN
            IEND = J - 1
        ELSE
            IEND = J
        END IF
    ELSE
        IF( UNIT ) THEN
            IBEG = J + 1
        ELSE
            IBEG = J
        END IF
        IEND = N
    END IF
    DO 100 I = 1, IBEG - 1
        AA( I + ( J - 1 ) * LDA ) = ROGUE
100    CONTINUE
    DO 110 I = IBEG, IEND
        AA( I + ( J - 1 ) * LDA ) = A( I, J )
110    CONTINUE
    DO 120 I = IEND + 1, LDA
        AA( I + ( J - 1 ) * LDA ) = ROGUE
120    CONTINUE
    IF( SYM ) THEN
        JJ = J + ( J - 1 ) * LDA
        AA( JJ ) = DCMPLX( DBLE( AA( JJ ) ), RROGUE )
    END IF
130    CONTINUE
    ELSE IF( TYPE.EQ.'HB'.OR.TYPE.EQ.'TB' ) THEN
        DO 170 J = 1, N
            IF( UPPER ) THEN
                KK = KL + 1
                IBEG = MAX( 1, KL + 2 - J )
                IF( UNIT ) THEN
                    IEND = KL
                ELSE
                    IEND = KL + 1
                END IF
            ELSE
                KK = 1
                IF( UNIT ) THEN
                    IBEG = 2
                ELSE
                    IBEG = 1
                END IF
                IEND = MIN( KL + 1, 1 + M - J )
            END IF
            DO 140 I = 1, IBEG - 1
                AA( I + ( J - 1 ) * LDA ) = ROGUE
140            CONTINUE
            DO 150 I = IBEG, IEND
                AA( I + ( J - 1 ) * LDA ) = A( I + J - KK, J )
150            CONTINUE
            DO 160 I = IEND + 1, LDA
                AA( I + ( J - 1 ) * LDA ) = ROGUE
160            CONTINUE
            IF( SYM ) THEN
                JJ = KK + ( J - 1 ) * LDA
                AA( JJ ) = DCMPLX( DBLE( AA( JJ ) ), RROGUE )
            END IF
170            CONTINUE
        ELSE IF( TYPE.EQ.'HP'.OR.TYPE.EQ.'TP' ) THEN
            IOFF = 0
            DO 190 J = 1, N
                IF( UPPER ) THEN
                    IBEG = 1
                    IEND = J
                ELSE
                    IBEG = J
                    IEND = N
                END IF
                DO 180 I = IBEG, IEND
                    IOFF = IOFF + 1
                    AA( IOFF ) = A( I, J )
                    IF( I.EQ.J ) THEN
                        IF( UNIT ) &
                            AA( IOFF ) = ROGUE
                        IF( SYM ) &
                            AA( IOFF ) = DCMPLX( DBLE( AA( IOFF ) ), RROGUE )
                    END IF
180                CONTINUE
190            CONTINUE
        END IF
        RETURN
    !
    !   End of ZMAKE.
    !
    !
    !   END SUBROUTINE
    SUBROUTINE ZMVCH( TRANS, M, N, ALPHA, A, NMAX, X, INCX, BETA, Y, &
        INCY, YT, G, YY, EPS, ERR, FATAL, NOUT, MV )
    !
    !   Checks the results of the computational tests.
    !
    !   Auxiliary routine for test program for Level 2 Blas.
    !
    !   -- Written on 10-August-1987.
    !   Richard Hanson, Sandia National Labs.
    !   Jeremy Du Croz, NAG Central Office.
    !
    !
    !   .. Parameters ..
    COMPLEX*16      ZERO
    PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ) )
    DOUBLE PRECISION RZERO, RONE
    PARAMETER      ( RZERO = 0.0D0, RONE = 1.0D0 )
    !
    !   .. Scalar Arguments ..
    COMPLEX*16      ALPHA, BETA
    DOUBLE PRECISION EPS, ERR
    INTEGER         INCX, INCY, M, N, NMAX, NOUT
    LOGICAL         FATAL, MV
    CHARACTER*1     TRANS
    !
    !   .. Array Arguments ..
    COMPLEX*16      A( NMAX, * ), X( * ), Y( * ), YT( * ), YY( * )
    DOUBLE PRECISION G( * )
    !
    !   .. Local Scalars ..
    COMPLEX*16      C

```

```

DOUBLE PRECISION  ERRI
INTEGER           I, INCXL, INCYL, IY, J, JX, KX, KY, ML, NL
LOGICAL           CTRAN, TRAN
! .. Intrinsic Functions ..
INTRINSIC         ABS, DBLE, DCONJG, DIMAG, MAX, SQRT
! .. Statement Functions ..
DOUBLE PRECISION ABS1
! .. Statement Function Definitions ..
ABS1( C ) = ABS( DBLE( C ) ) + ABS( DIMAG( C ) )
! .. Executable Statements ..
TRAN = TRANS.EQ.'T'
CTRAN = TRANS.EQ.'C'
IF( TRAN.OR.CTRAN )THEN
  ML = N
  NL = M
ELSE
  ML = M
  NL = N
END IF
IF( INCX.LT.0 )THEN
  KX = NL
  INCXL = -1
ELSE
  KX = 1
  INCXL = 1
END IF
IF( INCY.LT.0 )THEN
  KY = ML
  INCYL = -1
ELSE
  KY = 1
  INCYL = 1
END IF
!
! Compute expected result in YT using data in A, X and Y.
! Compute gauges in G.
!
IY = KY
DO 40 I = 1, ML
  YT( IY ) = ZERO
  G( IY ) = RZERO
  JX = KX
  IF( TRAN )THEN
    DO 10 J = 1, NL
      YT( IY ) = YT( IY ) + A( J, I ) * X( JX )
      G( IY ) = G( IY ) + ABS1( A( J, I ) ) * ABS1( X( JX ) )
      JX = JX + INCXL
    10 CONTINUE
  ELSE IF( CTRAN )THEN
    DO 20 J = 1, NL
      YT( IY ) = YT( IY ) + DCONJG( A( J, I ) ) * X( JX )
      G( IY ) = G( IY ) + ABS1( A( J, I ) ) * ABS1( X( JX ) )
      JX = JX + INCXL
    20 CONTINUE
  ELSE
    DO 30 J = 1, NL
      YT( IY ) = YT( IY ) + A( I, J ) * X( JX )
      G( IY ) = G( IY ) + ABS1( A( I, J ) ) * ABS1( X( JX ) )
      JX = JX + INCXL
    30 CONTINUE
  END IF
  YT( IY ) = ALPHA * YT( IY ) + BETA * Y( IY )
  G( IY ) = ABS1( ALPHA ) * G( IY ) + ABS1( BETA ) * ABS1( Y( IY ) )
  IY = IY + INCYL
40 CONTINUE
!
! Compute the error ratio for this result.
!
ERR = ZERO
DO 50 I = 1, ML
  ERRI = ABS( YT( I ) - YY( 1 + ( I - 1 ) * ABS( INCY ) ) ) / EPS
  IF( G( I ) .NE. RZERO ) &
    ERRI = ERRI / G( I )
  ERR = MAX( ERR, ERRI )
  IF( ERR * SQRT( EPS ) .GE. RONE ) &
    GO TO 60
50 CONTINUE
! If the loop completes, all results are at least half accurate.
GO TO 80
!
! Report fatal error.
!
60 FATAL = .TRUE.
WRITE( NOUT, FMT = 9999 )
DO 70 I = 1, ML
  IF( MV )THEN
    WRITE( NOUT, FMT = 9998 ) I, YT( I ), &
      YY( 1 + ( I - 1 ) * ABS( INCY ) )
  ELSE
    WRITE( NOUT, FMT = 9998 ) I, &
      YY( 1 + ( I - 1 ) * ABS( INCY ) ), YT( I )
  END IF
70 CONTINUE
!
80 CONTINUE
RETURN
!
9999 FORMAT( ' ***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HAL', &
  'F ACCURATE *****', /, ' EXPECTED RE', &
  'SULT', ' COMPUTED RESULT' )
9998 FORMAT( 1X, I7, 2( ' (', G15.6, ', ', G15.6, ' )' ) )
!
! End of ZMVCH.
!
END SUBROUTINE
LOGICAL FUNCTION LZE( RI, RJ, LR )
!
! Tests if two arrays are identical.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.

```

```

!
! .. Scalar Arguments ..
! INTEGER          LR
! .. Array Arguments ..
! COMPLEX*16       RI( * ), RJ( * )
! .. Local Scalars ..
! INTEGER          I
! .. Executable Statements ..
! DO 10 I = 1, LR
!   IF( RI( I ) ).NE.RJ( I ) %
!     GO TO 20
! 10 CONTINUE
!   LZE = .TRUE.
!   GO TO 30
! 20 CONTINUE
!   LZE = .FALSE.
! 30 RETURN
!
! End of LZE.
!
! END FUNCTION
! LOGICAL FUNCTION LZERES( TYPE, UPLO, M, N, AA, AS, LDA )
!
! Tests if selected elements in two arrays are equal.
!
! TYPE is 'GE', 'HE' or 'HP'.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ..
! INTEGER          LDA, M, N
! CHARACTER*1      UPLO
! CHARACTER*2      TYPE
! .. Array Arguments ..
! COMPLEX*16       AA( LDA, * ), AS( LDA, * )
! .. Local Scalars ..
! INTEGER          I, IBEG, IEND, J
! LOGICAL          UPPER
! .. Executable Statements ..
! UPPER = UPLO.EQ.'U'
! IF( TYPE.EQ.'GE' ) THEN
!   DO 20 J = 1, N
!     DO 10 I = M + 1, LDA
!       IF( AA( I, J ) .NE. AS( I, J ) ) %
!         GO TO 70
!     10 CONTINUE
!   20 CONTINUE
! ELSE IF( TYPE.EQ.'HE' ) THEN
!   DO 50 J = 1, N
!     IF( UPPER ) THEN
!       IBEG = 1
!       IEND = J
!     ELSE
!       IBEG = J
!       IEND = N
!     END IF
!     DO 30 I = 1, IBEG - 1
!       IF( AA( I, J ) .NE. AS( I, J ) ) %
!         GO TO 70
!     30 CONTINUE
!     DO 40 I = IEND + 1, LDA
!       IF( AA( I, J ) .NE. AS( I, J ) ) %
!         GO TO 70
!     40 CONTINUE
!   50 CONTINUE
! END IF
!
! LZERES = .TRUE.
! GO TO 80
! 70 CONTINUE
! LZERES = .FALSE.
! 80 RETURN
!
! End of LZERES.
!
! END FUNCTION
! COMPLEX*16 FUNCTION ZBEG( RESET )
!
! Generates complex numbers as pairs of random numbers uniformly
! distributed between -0.5 and 0.5.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ..
! LOGICAL          RESET
! .. Local Scalars ..
! INTEGER          I, IC, J, MI, MJ
! .. Save statement ..
! SAVE             I, IC, J, MI, MJ
! .. Intrinsic Functions ..
! INTRINSIC        DCMLPX
! .. Executable Statements ..
! IF( RESET ) THEN
!   Initialize local variables.
!   MI = 891
!   MJ = 457
!   I = 7
!   J = 7
!   IC = 0
!   RESET = .FALSE.
! END IF
!
! The sequence of values of I or J is bounded between 1 and 999.
! If initial I or J = 1,2,3,6,7 or 9, the period will be 50.
! If initial I or J = 4 or 8, the period will be 25.
! If initial I or J = 5, the period will be 10.

```

```

! IC is used to break up the period by skipping 1 value of I or J
! in 6.
!
! IC = IC + 1
10 I = I*MI
J = J*MJ
I = I - 1000*( I/1000 )
J = J - 1000*( J/1000 )
IF( IC.GE.5 )THEN
    IC = 0
    GO TO 10
END IF
ZBEG = DCMLPX( ( I - 500 )/1001.0D0, ( J - 500 )/1001.0D0 )
RETURN
!
! End of ZBEG.
!
! END FUNCTION
! DOUBLE PRECISION FUNCTION DDIFF( X, Y )
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
!
! .. Scalar Arguments ..
! DOUBLE PRECISION X, Y
! .. Executable Statements ..
! DDIFF = X - Y
! RETURN
!
! End of DDIFF.
!
! END FUNCTION
! SUBROUTINE CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
! Tests whether XERBLA has detected an error when it should.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ..
! INTEGER INFOT, NOUT
! LOGICAL LERR, OK
! CHARACTER*6 SRNAMT
! .. Executable Statements ..
! IF( .NOT.LERR )THEN
!     WRITE( NOUT, FMT = 9999 )INFOT, SRNAMT
!     OK = .FALSE.
! END IF
! LERR = .FALSE.
! RETURN
!
! 9999 FORMAT( ' ***** ILLEGAL VALUE OF PARAMETER NUMBER ', I2, ' NOT D',&
!             'ETECTED BY ', A6, ' *****' )
!
! End of CHKXER.
!
! END SUBROUTINE
! SUBROUTINE XERBLA( SRNAME, INFO )
!
! This is a special version of XERBLA to be used only as part of
! the test program for testing error exits from the Level 2 BLAS
! routines.
!
! XERBLA is an error handler for the Level 2 BLAS routines.
!
! It is called by the Level 2 BLAS routines if an input parameter is
! invalid.
!
! Auxiliary routine for test program for Level 2 Blas.
!
! -- Written on 10-August-1987.
! Richard Hanson, Sandia National Labs.
! Jeremy Du Croz, NAG Central Office.
!
! .. Scalar Arguments ...
! INTEGER INFO
! CHARACTER*6 SRNAME
! .. Scalars in Common ..
! INTEGER INFOT, NOUT
! LOGICAL LERR, OK
! CHARACTER*6 SRNAMT
! .. Common blocks ..
! COMMON /INFOC/INFOT, NOUT, OK, LERR
! COMMON /SRNAMC/SRNAME
! .. Executable Statements ..
! LERR = .TRUE.
! IF( INFO.NE.INFOT )THEN
!     IF( INFOT.NE.0 )THEN
!         WRITE( NOUT, FMT = 9999 )INFO, INFOT
!     ELSE
!         WRITE( NOUT, FMT = 9997 )INFO
!     END IF
!     OK = .FALSE.
! END IF
! IF( SRNAME.NE.SRNAME )THEN
!     WRITE( NOUT, FMT = 9998 )SRNAME, SRNAMT
!     OK = .FALSE.
! END IF
! RETURN
!
! 9999 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, ' INSTEAD',&
!             ' OF ', I2, ' *****' )
! 9998 FORMAT( ' ***** XERBLA WAS CALLED WITH SRNAME = ', A6, ' INSTE',&
!             'AD OF ', A6, ' *****' )
! 9997 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6,&
!             ' *****' )
!
! End of XERBLA
!

```

```

END SUBROUTINE
!> \brief \b ZGEMV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER INCX,INCY,LDA,M,N
! CHARACTER TRANS
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*),Y(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZGEMV performs one of the matrix-vector operations
!>
!>   y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or
!>
!>   y := alpha*A**H*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are vectors and A is an
!> m by n matrix.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
!>
!>   TRANS = 'T' or 't'   y := alpha*A**T*x + beta*y.
!>
!>   TRANS = 'C' or 'c'   y := alpha*A**H*x + beta*y.
!> \endverbatim
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix A.
!> M must be at least zero.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is COMPLEX*16
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] A
!> \verbatim
!> A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!> Before entry, the leading m by n part of the array A must
!> contain the matrix of coefficients.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. LDA must be at least
!> max( 1, m ).
!> \endverbatim
!> \param[in] X
!> \verbatim
!> X is COMPLEX*16 array of DIMENSION at least
!>   ( 1 + ( n - 1 ) *abs( INCX ) ) when TRANS = 'N' or 'n'
!>   and at least
!>   ( 1 + ( m - 1 ) *abs( INCX ) ) otherwise.
!> Before entry, the incremented array X must contain the
!> vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!> \param[in] BETA
!> \verbatim
!> BETA is COMPLEX*16
!> On entry, BETA specifies the scalar beta. When BETA is
!> supplied as zero then Y need not be set on input.
!> \endverbatim
!> \param[in,out] Y

```

```

!> \verbatim
!>      Y is COMPLEX*16 array of DIMENSION at least
!>      ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n'
!>      and at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ) otherwise.
!>      Before entry with BETA non-zero, the incremented array Y
!>      must contain the vector y. On exit, Y is overwritten by the
!>      updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!>      SUBROUTINE ZGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ALPHA,BETA
!> INTEGER INCX,INCY,LDA,M,N
!> CHARACTER TRANS
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 A(LDA,*),X(*),Y(*)
!> ..
!> =====
!>
!> .. Parameters ..
!> COMPLEX*16 ONE
!> PARAMETER (ONE= (1.0D+0,0.0D+0))
!> COMPLEX*16 ZERO
!> PARAMETER (ZERO= (0.0D+0,0.0D+0))
!> ..
!> .. Local Scalars ..
!> COMPLEX*16 TEMP
!> INTEGER I,INFO,IX,IY,J,JX,JY,KX,KY,LENX,LENY
!> LOGICAL NOCONJ
!> ..
!> .. External Functions ..
!> LOGICAL LSAME
!> EXTERNAL LSAME
!> ..
!> .. External Subroutines ..
!> EXTERNAL XERBLA
!> ..
!> .. Intrinsic Functions ..
!> INTRINSIC DCONJG,MAX
!> ..
!>
!> Test the input parameters.
!>
!> INFO = 0
!> IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND. &
!>   .NOT.LSAME(TRANS,'C')) THEN
!>   INFO = 1
!> ELSE IF (M.LT.0) THEN
!>   INFO = 2
!> ELSE IF (N.LT.0) THEN
!>   INFO = 3
!> ELSE IF (LDA.LT.MAX(1,M)) THEN
!>   INFO = 6
!> ELSE IF (INCX.EQ.0) THEN
!>   INFO = 8
!> ELSE IF (INCY.EQ.0) THEN
!>   INFO = 11
!> END IF
!> IF (INFO.NE.0) THEN
!>   CALL XERBLA('ZGEMV ',INFO)
!>   RETURN
!> END IF
!>
!> Quick return if possible.
!>
!> IF ((M.EQ.0) .OR. (N.EQ.0) .OR. &
!>   (ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN
!>
!> NOCONJ = LSAME(TRANS,'T')
!>
!>

```

```

! Set LENX and LENY, the lengths of the vectors x and y, and set
! up the start points in X and Y.
!
IF (LSAME(TRANS,'N')) THEN
    LENX = N
    LENY = M
ELSE
    LENX = M
    LENY = N
END IF
IF (INCX.GT.0) THEN
    KX = 1
ELSE
    KX = 1 - (LENX-1)*INCX
END IF
IF (INCY.GT.0) THEN
    KY = 1
ELSE
    KY = 1 - (LENY-1)*INCY
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
! First form y := beta*y.
!
IF (BETA.NE.ONE) THEN
    IF (INCY.EQ.1) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 10 I = 1,LENY
                Y(I) = ZERO
            CONTINUE
        ELSE
            DO 20 I = 1,LENY
                Y(I) = BETA*Y(I)
            CONTINUE
        END IF
    ELSE
        IY = KY
        IF (BETA.EQ.ZERO) THEN
            DO 30 I = 1,LENY
                Y(IY) = ZERO
            CONTINUE
        ELSE
            DO 40 I = 1,LENY
                Y(IY) = BETA*Y(IY)
            CONTINUE
        END IF
    END IF
END IF
IF (ALPHA.EQ.ZERO) RETURN
IF (LSAME(TRANS,'N')) THEN
    Form y := alpha*A*x + y.
    JX = KX
    IF (INCY.EQ.1) THEN
        DO 60 J = 1,N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*X(JX)
                DO 50 I = 1,M
                    Y(I) = Y(I) + TEMP*A(I,J)
                CONTINUE
            END IF
            JX = JX + INCX
        CONTINUE
    ELSE
        DO 80 J = 1,N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*X(JX)
                IY = KY
                DO 70 I = 1,M
                    Y(IY) = Y(IY) + TEMP*A(I,J)
                CONTINUE
            END IF
            JX = JX + INCX
        CONTINUE
    END IF
ELSE
    Form y := alpha*A**T*x + y or y := alpha*A**H*x + y.
    JY = KY
    IF (INCX.EQ.1) THEN
        DO 110 J = 1,N
            TEMP = ZERO
            IF (NOCONJ) THEN
                DO 90 I = 1,M
                    TEMP = TEMP + A(I,J)*X(I)
                CONTINUE
            ELSE
                DO 100 I = 1,M
                    TEMP = TEMP + DCONJG(A(I,J))*X(I)
                CONTINUE
            END IF
            Y(JY) = Y(JY) + ALPHA*TEMP
            JY = JY + INCY
        CONTINUE
    ELSE
        DO 140 J = 1,N
            TEMP = ZERO
            IX = KX
            IF (NOCONJ) THEN
                DO 120 I = 1,M
                    TEMP = TEMP + A(I,J)*X(IX)
                CONTINUE
            ELSE
                DO 130 I = 1,M
                    TEMP = TEMP + DCONJG(A(I,J))*X(IX)
                CONTINUE
            END IF
            IX = IX + INCX
        CONTINUE
    END IF

```

```

130          CONTINUE
            END IF
            Y(JY) = Y(JY) + ALPHA*TEMP
            JY = JY + INCY
140          CONTINUE
        END IF
    END IF
!
!   RETURN
!
!   End of ZGEMV .
!
!   END SUBROUTINE
!*> \brief \b ZGBMV
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZGBMV(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
!   .. Scalar Arguments ..
!   COMPLEX*16 ALPHA,BETA
!   INTEGER INCX,INCY,KL,KU,LDA,M,N
!   CHARACTER TRANS
!
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),X(*),Y(*)
!   ..
!
!> \par Purpose:
!   =====
!>
!> \verbatim
!> ZGBMV performs one of the matrix-vector operations
!>
!>   y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or
!>
!>   y := alpha*A**H*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are vectors and A is an
!> m by n band matrix, with kl sub-diagonals and ku super-diagonals.
!> \endverbatim
!
!   Arguments:
!   =====
!
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
!>
!>   TRANS = 'T' or 't'   y := alpha*A**T*x + beta*y.
!>
!>   TRANS = 'C' or 'c'   y := alpha*A**H*x + beta*y.
!> \endverbatim
!
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of the matrix A.
!> M must be at least zero.
!> \endverbatim
!
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of the matrix A.
!> N must be at least zero.
!> \endverbatim
!
!> \param[in] KL
!> \verbatim
!> KL is INTEGER
!> On entry, KL specifies the number of sub-diagonals of the
!> matrix A. KL must satisfy 0 .le. KL.
!> \endverbatim
!
!> \param[in] KU
!> \verbatim
!> KU is INTEGER
!> On entry, KU specifies the number of super-diagonals of the
!> matrix A. KU must satisfy 0 .le. KU.
!> \endverbatim
!
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is COMPLEX*16
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!
!> \param[in] A
!> \verbatim
!> A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!> Before entry, the leading ( kl + ku + 1 ) by n part of the
!> array A must contain the matrix of coefficients, supplied
!> column by column, with the leading diagonal of the matrix in
!> row ( ku + 1 ) of the array, the first super-diagonal
!> starting at position 2 in row ku, the first sub-diagonal
!> starting at position 1 in row ( ku + 2 ), and so on.
!> Elements in the array A that do not correspond to elements
!> in the band matrix (such as the top left ku by ku triangle)
!> are not referenced.
!> The following program segment will transfer a band matrix
!> from conventional full matrix storage to band storage:

```

```

!>
!>          DO 20, J = 1, N
!>              K = KU + 1 - J
!>              DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
!>                  A( K + I, J ) = matrix( I, J )
!>          10    CONTINUE
!>          20    CONTINUE
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. LDA must be at least
!> ( kl + ku + 1 ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!> X is COMPLEX*16 array of DIMENSION at least
!> ( 1 + ( n - 1 ) * abs( INCX ) ) when TRANS = 'N' or 'n'
!> and at least
!> ( 1 + ( m - 1 ) * abs( INCX ) ) otherwise.
!> Before entry, the incremented array X must contain the
!> vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!> BETA is COMPLEX*16
!> On entry, BETA specifies the scalar beta. When BETA is
!> supplied as zero then Y need not be set on input.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!> Y is COMPLEX*16 array of DIMENSION at least
!> ( 1 + ( m - 1 ) * abs( INCY ) ) when TRANS = 'N' or 'n'
!> and at least
!> ( 1 + ( n - 1 ) * abs( INCY ) ) otherwise.
!> Before entry, the incremented array Y must contain the
!> vector y. On exit, Y is overwritten by the updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!> INCY is INTEGER
!> On entry, INCY specifies the increment for the elements of
!> Y. INCY must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZGBMV(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ALPHA,BETA
!> INTEGER INCX,INCY,KL,KU,LDA,M,N
!> CHARACTER TRANS
!> ..
!> .. Array Arguments ..
!> COMPLEX*16 A(LDA,*),X(*),Y(*)
!> ..
!> =====
!>
!> .. Parameters ..
!> COMPLEX*16 ONE
!> PARAMETER (ONE= (1.0D+0,0.0D+0))
!> COMPLEX*16 ZERO
!> PARAMETER (ZERO= (0.0D+0,0.0D+0))
!> ..
!> .. Local Scalars ..
!> COMPLEX*16 TEMP
!> INTEGER I,INFO,IX,IY,J,JX,JY,K,KUP1,KX,KY,LENX,LENY
!> LOGICAL NOCONJ

```

```

! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX,MIN
! ..
!
! Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(TRANS,'N')) .AND. .NOT.LSAME(TRANS,'T') .AND.&
.NOT.LSAME(TRANS,'C')) THEN
    INFO = 1
ELSE IF (M.LT.0) THEN
    INFO = 2
ELSE IF (N.LT.0) THEN
    INFO = 3
ELSE IF (KL.LT.0) THEN
    INFO = 4
ELSE IF (KU.LT.0) THEN
    INFO = 5
ELSE IF (LDA.LT. (KL+KU+1)) THEN
    INFO = 8
ELSE IF (INCX.EQ.0) THEN
    INFO = 10
ELSE IF (INCY.EQ.0) THEN
    INFO = 13
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZGBMV ',INFO)
    RETURN
END IF

! Quick return if possible.
!
IF ((M.EQ.0) .OR. (N.EQ.0) .OR.&
    ((ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN

! NOCONJ = LSAME(TRANS,'T')
!
! Set LENX and LENY, the lengths of the vectors x and y, and set
! up the start points in X and Y.
!
IF (LSAME(TRANS,'N')) THEN
    LENX = N
    LENY = M
ELSE
    LENX = M
    LENY = N
END IF
IF (INCX.GT.0) THEN
    KX = 1
ELSE
    KX = 1 - (LENX-1)*INCX
END IF
IF (INCY.GT.0) THEN
    KY = 1
ELSE
    KY = 1 - (LENY-1)*INCY
END IF

!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the band part of A.
!
! First form y := beta*y.
!
IF (BETA.NE.ONE) THEN
    IF (INCY.EQ.1) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 10 I = 1,LENY
                Y(I) = ZERO
            CONTINUE
10        ELSE
            DO 20 I = 1,LENY
                Y(I) = BETA*Y(I)
            CONTINUE
20        END IF
        ELSE
            IY = KY
            IF (BETA.EQ.ZERO) THEN
                DO 30 I = 1,LENY
                    Y(IY) = ZERO
                    IY = IY + INCY
                CONTINUE
30            ELSE
                DO 40 I = 1,LENY
                    Y(IY) = BETA*Y(IY)
                    IY = IY + INCY
                CONTINUE
40            END IF
        END IF
    END IF
    IF (ALPHA.EQ.ZERO) RETURN
    KUP1 = KU + 1
    IF (LSAME(TRANS,'N')) THEN
        Form y := alpha*A*x + y.
        JX = KX
        IF (INCY.EQ.1) THEN
            DO 60 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = ALPHA*X(JX)
                    K = KUP1 - J
                    DO 50 I = MAX(1,J-KU),MIN(M,J+KL)
                        Y(I) = Y(I) + TEMP*A(K+I,J)
                    CONTINUE
50                END IF
                JX = JX + INCX
            CONTINUE

```

```

60      CONTINUE
      ELSE
        DO 80 J = 1,N
          IF (X(JX).NE.ZERO) THEN
            TEMP = ALPHA*X(JX)
            IY = KY
            K = KUP1 - J
            DO 70 I = MAX(1,J-KU),MIN(M,J+KL)
              Y(IY) = Y(IY) + TEMP*A(K+I,J)
              IY = IY + INCY
            CONTINUE
          END IF
          JX = JX + INCX
          IF (J.GT.KU) KY = KY + INCY
        CONTINUE
      END IF
    ELSE
      !
      !      Form y := alpha*A**T*x + y or y := alpha*A**H*x + y.
      !
      JY = KY
      IF (INCX.EQ.1) THEN
        DO 110 J = 1,N
          TEMP = ZERO
          K = KUP1 - J
          IF (NOCONJ) THEN
            DO 90 I = MAX(1,J-KU),MIN(M,J+KL)
              TEMP = TEMP + A(K+I,J)*X(I)
            CONTINUE
          ELSE
            DO 100 I = MAX(1,J-KU),MIN(M,J+KL)
              TEMP = TEMP + DCONJG(A(K+I,J))*X(I)
            CONTINUE
          END IF
          Y(JY) = Y(JY) + ALPHA*TEMP
          JY = JY + INCY
        CONTINUE
      ELSE
        DO 140 J = 1,N
          TEMP = ZERO
          IX = KX
          K = KUP1 - J
          IF (NOCONJ) THEN
            DO 120 I = MAX(1,J-KU),MIN(M,J+KL)
              TEMP = TEMP + A(K+I,J)*X(IX)
              IX = IX + INCX
            CONTINUE
          ELSE
            DO 130 I = MAX(1,J-KU),MIN(M,J+KL)
              TEMP = TEMP + DCONJG(A(K+I,J))*X(IX)
              IX = IX + INCX
            CONTINUE
          END IF
          Y(JY) = Y(JY) + ALPHA*TEMP
          JY = JY + INCY
          IF (J.GT.KU) KX = KX + INCX
        CONTINUE
      END IF
    END IF
  !
  !      RETURN
  !
  !      End of ZGBMV .
  !
  END SUBROUTINE

!
!      *> \brief \b ZHEMV
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE ZHEMV(UPLO,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 ALPHA,BETA
!      INTEGER INCX,INCY,LDA,N
!      CHARACTER UPLO
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 A(LDA,*),X(*),Y(*)
!      ..
!
!> \par Purpose:
!> =====
!> \verbatim
!> ZHEMV performs the matrix-vector operation
!>
!>   y := alpha*A*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are n element vectors and
!> A is an n by n hermitian matrix.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array A is to be referenced as
!> follows:
!>
!>   UPLO = 'U' or 'u' Only the upper triangular part of A
!>                     is to be referenced.
!>
!>

```

```

!>          UPLO = 'L' or 'l'    Only the lower triangular part of A
!>                                is to be referenced.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with  UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular part of the hermitian matrix and the strictly
!>     lower triangular part of A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular part of the hermitian matrix and the strictly
!>     upper triangular part of A is not referenced.
!>     Note that the imaginary parts of the diagonal elements need
!>     not be set and are assumed to be zero.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then Y need not be set on input.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>     Y is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y. On exit, Y is overwritten by the updated
!>     vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \endverbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZHEMV(UPLO,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..

```

```

COMPLEX*16 ALPHA,BETA
INTEGER INCX, INCY, LDA, N
CHARACTER UPLO
!
! .. Array Arguments ..
COMPLEX*16 A(LDA,*),X(*),Y(*)
!
! ..
! =====
!
! .. Parameters ..
COMPLEX*16 ONE
PARAMETER (ONE= (1.0D+0,0.0D+0))
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
COMPLEX*16 TEMP1,TEMP2
INTEGER I, INFO, IX, IY, J, JX, JY, KX, KY
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC DBLE, DCONJG, MAX
!
! ..
!
Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
    INFO = 1
ELSE IF (N.LT.0) THEN
    INFO = 2
ELSE IF (LDA.LT.MAX(1,N)) THEN
    INFO = 5
ELSE IF (INCX.EQ.0) THEN
    INFO = 7
ELSE IF (INCY.EQ.0) THEN
    INFO = 10
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZHEMV ',INFO)
    RETURN
END IF
!
! Quick return if possible.
!
IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!
! Set up the start points in X and Y.
!
IF (INCX.GT.0) THEN
    KX = 1
ELSE
    KX = 1 - (N-1)*INCX
END IF
IF (INCY.GT.0) THEN
    KY = 1
ELSE
    KY = 1 - (N-1)*INCY
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
! First form y := beta*y.
!
IF (BETA.NE.ONE) THEN
    IF (INCY.EQ.1) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 10 I = 1,N
                Y(I) = ZERO
            END IF
        ELSE
            DO 20 I = 1,N
                Y(I) = BETA*Y(I)
            END IF
        END IF
    ELSE
        IY = KY
        IF (BETA.EQ.ZERO) THEN
            DO 30 I = 1,N
                Y(IY) = ZERO
                IY = IY + INCY
            END IF
        ELSE
            DO 40 I = 1,N
                Y(IY) = BETA*Y(IY)
                IY = IY + INCY
            END IF
        END IF
    END IF
    IF (ALPHA.EQ.ZERO) RETURN
    IF (LSAME(UPLO,'U')) THEN
!
! Form y when A is stored in upper triangle.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 60 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                DO 50 I = 1, J - 1
                    Y(I) = Y(I) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + DCONJG(A(I,J))*X(I)
                CONTINUE
                Y(J) = Y(J) + TEMP1*DBLE(A(J,J)) + ALPHA*TEMP2
            CONTINUE
60          CONTINUE

```

```

        ELSE
            JX = KX
            JY = KY
            DO 80 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                IX = KX
                IY = KY
                DO 70 I = 1,J - 1
                    Y(IY) = Y(IY) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + DCONJG(A(I,J))*X(IX)
                    IX = IX + INCX
                    IY = IY + INCY
                70 CONTINUE
                Y(JY) = Y(JY) + TEMP1*DBLE(A(J,J)) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
            80 CONTINUE
        END IF
    ELSE
        !
        !       Form y when A is stored in lower triangle.
        !
        IF ((INCX.EQ.1).AND. (INCY.EQ.1)) THEN
            DO 100 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                Y(J) = Y(J) + TEMP1*DBLE(A(J,J))
                DO 90 I = J + 1,N
                    Y(I) = Y(I) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + DCONJG(A(I,J))*X(I)
                90 CONTINUE
                Y(J) = Y(J) + ALPHA*TEMP2
            100 CONTINUE
        ELSE
            JX = KX
            JY = KY
            DO 120 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                Y(JY) = Y(JY) + TEMP1*DBLE(A(J,J))
                IX = JX
                IY = JY
                DO 110 I = J + 1,N
                    IX = IX + INCX
                    IY = IY + INCY
                    Y(IY) = Y(IY) + TEMP1*A(I,J)
                    TEMP2 = TEMP2 + DCONJG(A(I,J))*X(IX)
                110 CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
            120 CONTINUE
        END IF
    END IF
    !
    RETURN
    !
    !       End of ZHEMV .
    !
    END SUBROUTINE
    *> \brief \b ZHBMV

    ===== DOCUMENTATION =====
    !
    !       Online html documentation available at
    !       http://www.netlib.org/lapack/explore-html/
    !
    !       Definition:
    !       =====
    !
    !       SUBROUTINE ZHBMV(UPLO,N,K,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
    !
    !       .. Scalar Arguments ..
    !       COMPLEX*16 ALPHA,BETA
    !       INTEGER INCX,INCY,K,LDA,N
    !       CHARACTER UPLO
    !
    !       ..
    !       .. Array Arguments ..
    !       COMPLEX*16 A(LDA,*),X(*),Y(*)
    !       ..
    !
    !> \par Purpose:
    !> =====
    !> \verbatim
    !> ZHBMV performs the matrix-vector operation
    !>
    !>   y := alpha*A*x + beta*y,
    !>
    !> where alpha and beta are scalars, x and y are n element vectors and
    !> A is an n by n hermitian band matrix, with k super-diagonals.
    !> \endverbatim
    !
    !       Arguments:
    !       =====
    !> \param[in] UPLO
    !> \verbatim
    !> UPLO is CHARACTER*1
    !> On entry, UPLO specifies whether the upper or lower
    !> triangular part of the band matrix A is being supplied as
    !> follows:
    !>
    !>     UPLO = 'U' or 'u'   The upper triangular part of A is
    !>                          being supplied.
    !>
    !>     UPLO = 'L' or 'l'   The lower triangular part of A is
    !>                          being supplied.
    !> \endverbatim
    !> \param[in] N

```

```

!> \verbatim
!>      N is INTEGER
!>      On entry, N specifies the order of the matrix A.
!>      N must be at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>      K is INTEGER
!>      On entry, K specifies the number of super-diagonals of the
!>      matrix A. K must satisfy 0 ≤ K.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is COMPLEX*16
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>      A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>      Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!>      by n part of the array A must contain the upper triangular
!>      band part of the hermitian matrix, supplied column by
!>      column, with the leading diagonal of the matrix in row
!>      ( k + 1 ) of the array, the first super-diagonal starting at
!>      position 2 in row k, and so on. The top left k by k triangle
!>      of the array A is not referenced.
!>      The following program segment will transfer the upper
!>      triangular part of a hermitian band matrix from conventional
!>      full matrix storage to band storage:
!>
!>      DO 20, J = 1, N
!>        M = K + 1 - J
!>        DO 10, I = MAX( 1, J - K ), J
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!>
!>      Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!>      by n part of the array A must contain the lower triangular
!>      band part of the hermitian matrix, supplied column by
!>      column, with the leading diagonal of the matrix in row 1 of
!>      the array, the first sub-diagonal starting at position 1 in
!>      row 2, and so on. The bottom right k by k triangle of the
!>      array A is not referenced.
!>      The following program segment will transfer the lower
!>      triangular part of a hermitian band matrix from conventional
!>      full matrix storage to band storage:
!>
!>      DO 20, J = 1, N
!>        M = 1 - J
!>        DO 10, I = J, MIN( N, J + K )
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!>
!>      Note that the imaginary parts of the diagonal elements need
!>      not be set and are assumed to be zero.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      ( k + 1 ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is COMPLEX*16 array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the
!>      vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is COMPLEX*16
!>      On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] Y
!> \verbatim
!>      Y is COMPLEX*16 array of DIMENSION at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ).
!>      Before entry, the incremented array Y must contain the
!>      vector y. On exit, Y is overwritten by the updated vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!>

```

```

!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
! =====
SUBROUTINE ZHBMV (UPLO,N,K,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
COMPLEX*16 ALPHA,BETA
INTEGER INCX,INCY,K,LDA,N
CHARACTER UPLO
!
! .. Array Arguments ..
COMPLEX*16 A(LDA,*),X(*),Y(*)
!
! ..
! =====
!
! .. Parameters ..
COMPLEX*16 ONE
PARAMETER (ONE= (1.0D+0,0.0D+0))
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
COMPLEX*16 TEMP1,TEMP2
INTEGER I,INFO,IX,IY,J,JX,KPLUS1,KX,KY,L
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC DBLE,DCONJG,MAX,MIN
!
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (K.LT.0) THEN
!   INFO = 3
! ELSE IF (LDA.LT. (K+1)) THEN
!   INFO = 6
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 8
! ELSE IF (INCY.EQ.0) THEN
!   INFO = 11
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZHBMV ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!
! Set up the start points in X and Y.
!
! IF (INCX.GT.0) THEN
!   KX = 1
! ELSE
!   KX = 1 - (N-1)*INCX
! END IF
! IF (INCY.GT.0) THEN
!   KY = 1
! ELSE
!   KY = 1 - (N-1)*INCY
! END IF
!
! Start the operations. In this version the elements of the array A
! are accessed sequentially with one pass through A.
!
! First form y := beta*y.
!
! IF (BETA.NE.ONE) THEN
!   IF (INCY.EQ.1) THEN
!     IF (BETA.EQ.ZERO) THEN
!       DO 10 I = 1,N
!         Y(I) = ZERO
!10      CONTINUE
!     ELSE
!       DO 20 I = 1,N
!         Y(I) = BETA*Y(I)
!20      CONTINUE
!     END IF
!   ELSE
!

```

```

        IY = KY
        IF (BETA.EQ.ZERO) THEN
            DO 30 I = 1,N
                Y(IY) = ZERO
                IY = IY + INCY
30         CONTINUE
        ELSE
            DO 40 I = 1,N
                Y(IY) = BETA*Y(IY)
                IY = IY + INCY
40         CONTINUE
        END IF
    END IF
    IF (ALPHA.EQ.ZERO) RETURN
    IF (LSAME(UPLO,'U')) THEN
!
!       Form y when upper triangle of A is stored.
!
        KPLUS1 = K + 1
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 60 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                L = KPLUS1 - J
                DO 50 I = MAX(1,J-K),J - 1
                    Y(I) = Y(I) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + DCONJG(A(L+I,J))*X(I)
50             CONTINUE
                Y(J) = Y(J) + TEMP1*DBLE(A(KPLUS1,J)) + ALPHA*TEMP2
60             CONTINUE
        ELSE
            JX = KX
            JY = KY
            DO 80 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                IX = KX
                IY = KY
                L = KPLUS1 - J
                DO 70 I = MAX(1,J-K),J - 1
                    Y(IY) = Y(IY) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + DCONJG(A(L+I,J))*X(IX)
                    IX = IX + INCX
                    IY = IY + INCY
70             CONTINUE
                Y(JY) = Y(JY) + TEMP1*DBLE(A(KPLUS1,J)) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
                IF (J.GT.K) THEN
                    KX = KX + INCX
                    KY = KY + INCY
                END IF
80             CONTINUE
        END IF
    ELSE
!
!       Form y when lower triangle of A is stored.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
            DO 100 J = 1,N
                TEMP1 = ALPHA*X(J)
                TEMP2 = ZERO
                Y(J) = Y(J) + TEMP1*DBLE(A(1,J))
                L = 1 - J
                DO 90 I = J + 1,MIN(N,J+K)
                    Y(I) = Y(I) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + DCONJG(A(L+I,J))*X(I)
90             CONTINUE
                Y(J) = Y(J) + ALPHA*TEMP2
100            CONTINUE
        ELSE
            JX = KX
            JY = KY
            DO 120 J = 1,N
                TEMP1 = ALPHA*X(JX)
                TEMP2 = ZERO
                Y(JY) = Y(JY) + TEMP1*DBLE(A(1,J))
                L = 1 - J
                IX = JX
                IY = JY
                DO 110 I = J + 1,MIN(N,J+K)
                    Y(IY) = Y(IY) + TEMP1*A(L+I,J)
                    TEMP2 = TEMP2 + DCONJG(A(L+I,J))*X(IX)
110            CONTINUE
                Y(JY) = Y(JY) + ALPHA*TEMP2
                JX = JX + INCX
                JY = JY + INCY
120            CONTINUE
        END IF
    END IF
    RETURN
!
!       End of ZHBMV .
!
    END SUBROUTINE
    *> \brief \b ZHPMV
!
!       ===== DOCUMENTATION =====
!
!       Online html documentation available at
!       http://www.netlib.org/lapack/explore-html/
!
!       Definition:
!       =====
!
!       SUBROUTINE ZHPMV(UPLO,N,ALPHA,AP,X,INCX,BETA,Y,INCY)
!
!       .. Scalar Arguments ..
!       COMPLEX*16 ALPHA,BETA
!       INTEGER INCX,INCY,N

```

```

!      CHARACTER UPLO
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 AP(*),X(*),Y(*)
!      ..
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> ZHPMV performs the matrix-vector operation
!>
!>   y := alpha*A*x + beta*y,
!>
!> where alpha and beta are scalars, x and y are n element vectors and
!> A is an n by n hermitian matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the upper or lower
!>     triangular part of the matrix A is supplied in the packed
!>     array AP as follows:
!>
!>         UPLO = 'U' or 'u'   The upper triangular part of A is
!>                             supplied in AP.
!>
!>         UPLO = 'L' or 'l'   The lower triangular part of A is
!>                             supplied in AP.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] AP
!> \verbatim
!>     AP is COMPLEX*16 array of DIMENSION at least
!>     ( ( n*( n + 1 ) )/2 ).
!>     Before entry with UPLO = 'U' or 'u', the array AP must
!>     contain the upper triangular part of the hermitian matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>     and a( 2, 2 ) respectively, and so on.
!>     Before entry with UPLO = 'L' or 'l', the array AP must
!>     contain the lower triangular part of the hermitian matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>     and a( 3, 1 ) respectively, and so on.
!>     Note that the imaginary parts of the diagonal elements need
!>     not be set and are assumed to be zero.
!> \endverbatim
!> \param[in] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then Y need not be set on input.
!> \endverbatim
!> \param[in,out] Y
!> \verbatim
!>     Y is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y. On exit, Y is overwritten by the updated
!>     vector y.
!> \endverbatim
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!

```

```

!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
! =====
SUBROUTINE ZHPMV (UPLO,N,ALPHA,AP,X,INCX,BETA,Y,INCY)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
COMPLEX*16 ALPHA,BETA
INTEGER INCX,INCY,N
CHARACTER UPLO
!
! .. Array Arguments ..
COMPLEX*16 AP(*),X(*),Y(*)
!
! ..
! =====
!
! .. Parameters ..
COMPLEX*16 ONE
PARAMETER (ONE= (1.0D+0,0.0D+0))
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
COMPLEX*16 TEMP1,TEMP2
INTEGER I,INFO,IX,IY,J,JX,JY,K,KK,KX,KY
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC DBLE,DCONJG
!
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 6
! ELSE IF (INCY.EQ.0) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZHPMV ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).AND. (BETA.EQ.ONE))) RETURN
!
! Set up the start points in X and Y.
!
! IF (INCX.GT.0) THEN
!   KX = 1
! ELSE
!   KX = 1 - (N-1)*INCX
! END IF
! IF (INCY.GT.0) THEN
!   KY = 1
! ELSE
!   KY = 1 - (N-1)*INCY
! END IF
!
! Start the operations. In this version the elements of the array AP
! are accessed sequentially with one pass through AP.
!
! First form y := beta*y.
!
! IF (BETA.NE.ONE) THEN
!   IF (INCY.EQ.1) THEN
!     IF (BETA.EQ.ZERO) THEN
!       DO 10 I = 1,N
!         Y(I) = ZERO
!       CONTINUE
!     ELSE
!       DO 20 I = 1,N
!         Y(I) = BETA*Y(I)
!       CONTINUE
!     END IF
!   ELSE
!     IY = KY
!     IF (BETA.EQ.ZERO) THEN
!       DO 30 I = 1,N
!         Y(IY) = ZERO

```

```

        IY = IY + INCY
30      CONTINUE
      ELSE
        DO 40 I = 1,N
          Y(IY) = BETA*Y(IY)
          IY = IY + INCY
40      CONTINUE
      END IF
    END IF
  END IF
  IF (ALPHA.EQ.ZERO) RETURN
  KK = 1
  IF (LSAME(UPLO,'U')) THEN
!
!   Form y when AP contains the upper triangle.
!
    IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
      DO 60 J = 1,N
        TEMP1 = ALPHA*X(J)
        TEMP2 = ZERO
        K = KK
        DO 50 I = 1,J - 1
          Y(I) = Y(I) + TEMP1*AP(K)
          TEMP2 = TEMP2 + DCONJG(AP(K))*X(I)
          K = K + 1
50      CONTINUE
        Y(J) = Y(J) + TEMP1*DBLE(AP(KK+J-1)) + ALPHA*TEMP2
        KK = KK + J
60      CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80 J = 1,N
          TEMP1 = ALPHA*X(JX)
          TEMP2 = ZERO
          IX = KX
          IY = KY
          DO 70 K = KK, KK + J - 2
            Y(IY) = Y(IY) + TEMP1*AP(K)
            TEMP2 = TEMP2 + DCONJG(AP(K))*X(IX)
            IX = IX + INCX
            IY = IY + INCY
70          CONTINUE
          Y(JY) = Y(JY) + TEMP1*DBLE(AP(KK+J-1)) + ALPHA*TEMP2
          JX = JX + INCX
          JY = JY + INCY
          KK = KK + J
80          CONTINUE
        END IF
      ELSE
!
!   Form y when AP contains the lower triangle.
!
        IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
          DO 100 J = 1,N
            TEMP1 = ALPHA*X(J)
            TEMP2 = ZERO
            Y(J) = Y(J) + TEMP1*DBLE(AP(KK))
            K = KK + 1
            DO 90 I = J + 1,N
              Y(I) = Y(I) + TEMP1*AP(K)
              TEMP2 = TEMP2 + DCONJG(AP(K))*X(I)
              K = K + 1
90          CONTINUE
            Y(J) = Y(J) + ALPHA*TEMP2
            KK = KK + (N-J+1)
100         CONTINUE
          ELSE
            JX = KX
            JY = KY
            DO 120 J = 1,N
              TEMP1 = ALPHA*X(JX)
              TEMP2 = ZERO
              Y(JY) = Y(JY) + TEMP1*DBLE(AP(KK))
              IX = JX
              IY = JY
              DO 110 K = KK + 1, KK + N - J
                IX = IX + INCX
                IY = IY + INCY
                Y(IY) = Y(IY) + TEMP1*AP(K)
                TEMP2 = TEMP2 + DCONJG(AP(K))*X(IX)
110             CONTINUE
              Y(JY) = Y(JY) + ALPHA*TEMP2
              JX = JX + INCX
              JY = JY + INCY
              KK = KK + (N-J+1)
120             CONTINUE
            END IF
          END IF
        RETURN
!
!   End of ZHPMV .
!
      END SUBROUTINE
      *> \brief \b ZTRMV
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZTRMV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,LDA,N
!   CHARACTER DIAG,TRANS,UPLO
!
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),X(*)
!   ..

```

```

!
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZTRMV performs one of the matrix-vector operations
!>
!>   x := A*x,   or   x := A**T*x,   or   x := A**H*x,
!>
!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular matrix.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the operation to be performed as
!>     follows:
!>
!>         TRANS = 'N' or 'n'   x := A*x.
!>
!>         TRANS = 'T' or 't'   x := A**T*x.
!>
!>         TRANS = 'C' or 'c'   x := A**H*x.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular matrix and the strictly lower triangular part of
!>     A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular matrix and the strictly upper triangular part of
!>     A is not referenced.
!>     Note that when DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced either, but are assumed to be unity.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!> \param[in] X
!> \verbatim
!>     X is (input/output) COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x. On exit, X is overwritten with the
!>     transformed vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====

```

```

!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
! =====
      SUBROUTINE ZTRMV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
! .. Scalar Arguments ..
      INTEGER INCX,LDA,N
      CHARACTER DIAG,TRANS,UPLO
!
! .. Array Arguments ..
      COMPLEX*16 A(LDA,*),X(*)
!
! ..
! =====
!
! .. Parameters ..
      COMPLEX*16 ZERO
      PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
      COMPLEX*16 TEMP
      INTEGER I,INFO,IX,J,JX,KX
      LOGICAL NOCONJ,NOUNIT
!
! ..
! .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
!
! .. External Subroutines ..
      EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
      INTRINSIC DCONJG,MAX
!
! ..
!
! Test the input parameters.
!
      INFO = 0
      IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
          INFO = 1
      ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.
             .NOT.LSAME(TRANS,'C')) THEN
          INFO = 2
      ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
          INFO = 3
      ELSE IF (N.LT.0) THEN
          INFO = 4
      ELSE IF (LDA.LT.MAX(1,N)) THEN
          INFO = 6
      ELSE IF (INCX.EQ.0) THEN
          INFO = 8
      END IF
      IF (INFO.NE.0) THEN
          CALL XERBLA('ZTRMV ',INFO)
          RETURN
      END IF
!
! Quick return if possible.
!
      IF (N.EQ.0) RETURN
!
      NOCONJ = LSAME(TRANS,'T')
      NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
      IF (INCX.LE.0) THEN
          KX = 1 - (N-1)*INCX
      ELSE IF (INCX.NE.1) THEN
          KX = 1
      END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
      IF (LSAME(TRANS,'N')) THEN
!
! Form x := A*x.
!
          IF (LSAME(UPLO,'U')) THEN
              IF (INCX.EQ.1) THEN
                  DO 20 J = 1,N
                      IF (X(J).NE.ZERO) THEN
                          TEMP = X(J)
                          DO 10 I = 1,J - 1
                              X(I) = X(I) + TEMP*A(I,J)
                          CONTINUE
                          IF (NOUNIT) X(J) = X(J)*A(J,J)
                      END IF
                  CONTINUE
              ELSE
                  JX = KX
                  DO 40 J = 1,N
                      IF (X(JX).NE.ZERO) THEN
                          TEMP = X(JX)
                          IX = KX
                          DO 30 I = 1,J - 1

```

```

        X(IX) = X(IX) + TEMP*A(I,J)
        IX = IX + INCX
30      CONTINUE
        IF (NOUNIT) X(JX) = X(JX)*A(J,J)
        END IF
        JX = JX + INCX
40      CONTINUE
    END IF
ELSE
    IF (INCX.EQ.1) THEN
        DO 60 J = N,1,-1
            IF (X(J).NE.ZERO) THEN
                TEMP = X(J)
                DO 50 I = N,J + 1,-1
                    X(I) = X(I) + TEMP*A(I,J)
50                CONTINUE
                IF (NOUNIT) X(J) = X(J)*A(J,J)
                END IF
60            CONTINUE
        ELSE
            KX = KX + (N-1)*INCX
            JX = KX
            DO 80 J = N,1,-1
                IF (X(JX).NE.ZERO) THEN
                    TEMP = X(JX)
                    IX = KX
                    DO 70 I = N,J + 1,-1
                        X(IX) = X(IX) + TEMP*A(I,J)
70                    CONTINUE
                        IF (NOUNIT) X(JX) = X(JX)*A(J,J)
                    END IF
                    JX = JX - INCX
80                CONTINUE
            END IF
        END IF
    ELSE
        Form x := A**T*x or x := A**H*x.
    !
    !
    IF (LSAME(UPLO,'U')) THEN
        IF (INCX.EQ.1) THEN
            DO 110 J = N,1,-1
                TEMP = X(J)
                IF (NOCONJ) THEN
                    IF (NOUNIT) TEMP = TEMP*A(J,J)
                    DO 90 I = J - 1,1,-1
                        TEMP = TEMP + A(I,J)*X(I)
90                    CONTINUE
                ELSE
                    IF (NOUNIT) TEMP = TEMP*DCONJG(A(J,J))
                    DO 100 I = J - 1,1,-1
                        TEMP = TEMP + DCONJG(A(I,J))*X(I)
100                    CONTINUE
                    END IF
                    X(J) = TEMP
110                CONTINUE
            ELSE
                JX = KX + (N-1)*INCX
                DO 140 J = N,1,-1
                    TEMP = X(JX)
                    IX = JX
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(J,J)
                        DO 120 I = J - 1,1,-1
                            IX = IX - INCX
                            TEMP = TEMP + A(I,J)*X(IX)
120                        CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(J,J))
                        DO 130 I = J - 1,1,-1
                            IX = IX - INCX
                            TEMP = TEMP + DCONJG(A(I,J))*X(IX)
130                        CONTINUE
                    END IF
                    X(JX) = TEMP
                    JX = JX - INCX
140                CONTINUE
            END IF
        ELSE
            IF (INCX.EQ.1) THEN
                DO 170 J = 1,N
                    TEMP = X(J)
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(J,J)
                        DO 150 I = J + 1,N
                            TEMP = TEMP + A(I,J)*X(I)
150                        CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(J,J))
                        DO 160 I = J + 1,N
                            TEMP = TEMP + DCONJG(A(I,J))*X(I)
160                        CONTINUE
                    END IF
                    X(J) = TEMP
170                CONTINUE
            ELSE
                JX = KX
                DO 200 J = 1,N
                    TEMP = X(JX)
                    IX = JX
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(J,J)
                        DO 180 I = J + 1,N
                            IX = IX + INCX
                            TEMP = TEMP + A(I,J)*X(IX)
180                        CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(J,J))
                        DO 190 I = J + 1,N
                            IX = IX + INCX
                            TEMP = TEMP + DCONJG(A(I,J))*X(IX)
190                        CONTINUE
                    END IF
                END IF
            END IF
        END IF
    END IF

```

```

                X(JX) = TEMP
                JX = JX + INCX
200             CONTINUE
            END IF
        END IF
    END IF
!
!   RETURN
!
!   End of ZTRMV .
!
!   END SUBROUTINE
!   *> \brief \b ZTBMV
!
!   ===== DOCUMENTATION =====
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZTBMV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
!   .. Scalar Arguments ..
!   INTEGER INCX,K,LDA,N
!   CHARACTER DIAG,TRANS,UPLO
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),X(*)
!   ..
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!> ZTBMV performs one of the matrix-vector operations
!>
!>   x := A*x,   or   x := A**T*x,   or   x := A**H*x,
!>
!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular band matrix, with ( k + 1 ) diagonals.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'   A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the operation to be performed as
!>     follows:
!>
!>         TRANS = 'N' or 'n'   x := A*x.
!>
!>         TRANS = 'T' or 't'   x := A**T*x.
!>
!>         TRANS = 'C' or 'c'   x := A**H*x.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'   A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'   A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with UPLO = 'U' or 'u', K specifies the number of
!>     super-diagonals of the matrix A.
!>     On entry with UPLO = 'L' or 'l', K specifies the number of
!>     sub-diagonals of the matrix A.
!>     K must satisfy 0 .le. K.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!>     by n part of the array A must contain the upper triangular
!>     band part of the matrix of coefficients, supplied column by
!>     column, with the leading diagonal of the matrix in row
!>     ( k + 1 ) of the array, the first super-diagonal starting at
!>     position 2 in row k, and so on. The top left k by k triangle
!>     of the array A is not referenced.
!>     The following program segment will transfer an upper
!>     triangular band matrix from conventional full matrix storage

```

```

!>      to band storage:
!>
!>      DO 20, J = 1, N
!>        M = K + 1 - J
!>        DO 10, I = MAX( 1, J - K ), J
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!>
!>      Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!>      by n part of the array A must contain the lower triangular
!>      band part of the matrix of coefficients, supplied column by
!>      column, with the leading diagonal of the matrix in row 1 of
!>      the array, the first sub-diagonal starting at position 1 in
!>      row 2, and so on. The bottom right k by k triangle of the
!>      array A is not referenced.
!>      The following program segment will transfer a lower
!>      triangular band matrix from conventional full matrix storage
!>      to band storage:
!>
!>      DO 20, J = 1, N
!>        M = 1 - J
!>        DO 10, I = J, MIN( N, J + K )
!>          A( M + I, J ) = matrix( I, J )
!>        10 CONTINUE
!>      20 CONTINUE
!>
!>      Note that when DIAG = 'U' or 'u' the elements of the array A
!>      corresponding to the diagonal elements of the matrix are not
!>      referenced, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      ( k + 1 ).
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is (input/output) COMPLEX*16 array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the n
!>      element vector x. On exit, X is overwritten with the
!>      transformed vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!> \verbatim
!>
!>      Level 2 Blas routine.
!>      The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!>      -- Written on 22-October-1986.
!>      Jack Dongarra, Argonne National Lab.
!>      Jeremy Du Croz, Nag Central Office.
!>      Sven Hammarling, Nag Central Office.
!>      Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!>      SUBROUTINE ZTMV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!>
!>      -- Reference BLAS level2 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
!>      November 2011
!>
!>      .. Scalar Arguments ..
!>      INTEGER INCX,K,LDA,N
!>      CHARACTER DIAG,TRANS,UPLO
!>
!>      ..
!>      .. Array Arguments ..
!>      COMPLEX*16 A(LDA,*),X(*)
!>      ..
!>
!>      =====
!>
!>      .. Parameters ..
!>      COMPLEX*16 ZERO
!>      PARAMETER (ZERO= (0.0D+0,0.0D+0))
!>
!>      ..
!>      .. Local Scalars ..
!>      COMPLEX*16 TEMP
!>      INTEGER I,INFO,IX,J,JX,KPLUS1,KX,L
!>      LOGICAL NOCONJ,NOUNIT
!>
!>      ..
!>      .. External Functions ..
!>      LOGICAL LSAME
!>      EXTERNAL LSAME
!>
!>      ..

```

```

! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX,MIN
! ..
!
! Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
    INFO = 1
ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.&
    .NOT.LSAME(TRANS,'C')) THEN
    INFO = 2
ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
    INFO = 3
ELSE IF (N.LT.0) THEN
    INFO = 4
ELSE IF (K.LT.0) THEN
    INFO = 5
ELSE IF (LDA.LT. (K+1)) THEN
    INFO = 7
ELSE IF (INCX.EQ.0) THEN
    INFO = 9
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZTBMV ',INFO)
    RETURN
END IF

! Quick return if possible.
!
IF (N.EQ.0) RETURN

! NOCONJ = LSAME(TRANS,'T')
! NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
IF (INCX.LE.0) THEN
    KX = 1 - (N-1)*INCX
ELSE IF (INCX.NE.1) THEN
    KX = 1
END IF

! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
IF (LSAME(TRANS,'N')) THEN
    Form x := A*x.

    IF (LSAME(UPLO,'U')) THEN
        KPLUS1 = K + 1
        IF (INCX.EQ.1) THEN
            DO 20 J = 1,N
                IF (X(J).NE.ZERO) THEN
                    TEMP = X(J)
                    L = KPLUS1 - J
                    DO 10 I = MAX(1,J-K),J - 1
                        X(I) = X(I) + TEMP*A(L+I,J)
                    CONTINUE
                    IF (NOUNIT) X(J) = X(J)*A(KPLUS1,J)
                END IF
            CONTINUE
        ELSE
            JX = KX
            DO 40 J = 1,N
                IF (X(JX).NE.ZERO) THEN
                    TEMP = X(JX)
                    IX = KX
                    L = KPLUS1 - J
                    DO 30 I = MAX(1,J-K),J - 1
                        X(IX) = X(IX) + TEMP*A(L+I,J)
                    IX = IX + INCX
                CONTINUE
                    IF (NOUNIT) X(JX) = X(JX)*A(KPLUS1,J)
                END IF
                JX = JX + INCX
                IF (J.GT.K) KX = KX + INCX
            CONTINUE
        END IF
    ELSE
        IF (INCX.EQ.1) THEN
            DO 60 J = N,1,-1
                IF (X(J).NE.ZERO) THEN
                    TEMP = X(J)
                    L = 1 - J
                    DO 50 I = MIN(N,J+K),J + 1,-1
                        X(I) = X(I) + TEMP*A(L+I,J)
                    CONTINUE
                    IF (NOUNIT) X(J) = X(J)*A(1,J)
                END IF
            CONTINUE
        ELSE
            KX = KX + (N-1)*INCX
            JX = KX
            DO 80 J = N,1,-1
                IF (X(JX).NE.ZERO) THEN
                    TEMP = X(JX)
                    IX = KX
                    L = 1 - J
                    DO 70 I = MIN(N,J+K),J + 1,-1
                        X(IX) = X(IX) + TEMP*A(L+I,J)
                    IX = IX - INCX
                CONTINUE
                    IF (NOUNIT) X(JX) = X(JX)*A(1,J)
                END IF
                JX = JX - INCX
                IF ((N-J).GE.K) KX = KX - INCX
            CONTINUE
        END IF
    END IF

```

```

        END IF
    ELSE
!
!       Form x := A**T*x or x := A**H*x.
!
        IF (LSAME(UPLO,'U')) THEN
            KPLUS1 = K + 1
            IF (INCX.EQ.1) THEN
                DO 110 J = N,1,-1
                    TEMP = X(J)
                    L = KPLUS1 - J
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(KPLUS1,J)
                        DO 90 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + A(L+I,J)*X(I)
90                     CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(KPLUS1,J))
                        DO 100 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + DCONJG(A(L+I,J))*X(I)
100                     CONTINUE
                    END IF
                    X(J) = TEMP
110                 CONTINUE
            ELSE
                KX = KX + (N-1)*INCX
                JX = KX
                DO 140 J = N,1,-1
                    TEMP = X(JX)
                    KX = KX - INCX
                    IX = KX
                    L = KPLUS1 - J
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(KPLUS1,J)
                        DO 120 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + A(L+I,J)*X(IX)
120                     CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(KPLUS1,J))
                        DO 130 I = J - 1,MAX(1,J-K),-1
                            TEMP = TEMP + DCONJG(A(L+I,J))*X(IX)
130                     CONTINUE
                    END IF
                    X(JX) = TEMP
                    JX = JX - INCX
140                 CONTINUE
            END IF
        ELSE
            IF (INCX.EQ.1) THEN
                DO 170 J = 1,N
                    TEMP = X(J)
                    L = 1 - J
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(1,J)
                        DO 150 I = J + 1,MIN(N,J+K)
                            TEMP = TEMP + A(L+I,J)*X(I)
150                     CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(1,J))
                        DO 160 I = J + 1,MIN(N,J+K)
                            TEMP = TEMP + DCONJG(A(L+I,J))*X(I)
160                     CONTINUE
                    END IF
                    X(J) = TEMP
170                 CONTINUE
            ELSE
                JX = KX
                DO 200 J = 1,N
                    TEMP = X(JX)
                    KX = KX + INCX
                    IX = KX
                    L = 1 - J
                    IF (NOCONJ) THEN
                        IF (NOUNIT) TEMP = TEMP*A(1,J)
                        DO 180 I = J + 1,MIN(N,J+K)
                            TEMP = TEMP + A(L+I,J)*X(IX)
180                     CONTINUE
                    ELSE
                        IF (NOUNIT) TEMP = TEMP*DCONJG(A(1,J))
                        DO 190 I = J + 1,MIN(N,J+K)
                            TEMP = TEMP + DCONJG(A(L+I,J))*X(IX)
190                     CONTINUE
                    END IF
                    X(JX) = TEMP
                    JX = JX + INCX
200                 CONTINUE
            END IF
        END IF
    END IF
!
    RETURN
!
!       End of ZTBMV .
!
    END SUBROUTINE
!
!       *> \brief \b ZTPMV
!
!       ===== DOCUMENTATION =====
!
!       Online html documentation available at
!       http://www.netlib.org/lapack/explore-html/
!
!       Definition:
!       =====
!
!       SUBROUTINE ZTPMV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
!       .. Scalar Arguments ..
!       INTEGER INCX,N
!       CHARACTER DIAG,TRANS,UPLO

```

```

! ..
! .. Array Arguments ..
! COMPLEX*16 AP(*),X(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZTPMV performs one of the matrix-vector operations
!>
!> x := A*x, or x := A**T*x, or x := A**H*x,
!>
!> where x is an n element vector and A is an n by n unit, or non-unit,
!> upper or lower triangular matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!> TRANS = 'N' or 'n' x := A*x.
!>
!> TRANS = 'T' or 't' x := A**T*x.
!>
!> TRANS = 'C' or 'c' x := A**H*x.
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] AP
!> \verbatim
!> AP is COMPLEX*16 array of DIMENSION at least
!> ( ( n*(n+1) )/2 ).
!> Before entry with UPLO = 'U' or 'u', the array AP must
!> contain the upper triangular matrix packed sequentially,
!> column by column, so that AP( 1 ) contains a( 1, 1 ),
!> AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
!> respectively, and so on.
!> Before entry with UPLO = 'L' or 'l', the array AP must
!> contain the lower triangular matrix packed sequentially,
!> column by column, so that AP( 1 ) contains a( 1, 1 ),
!> AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 )
!> respectively, and so on.
!> Note that when DIAG = 'U' or 'u', the diagonal elements of
!> A are not referenced, but are assumed to be unity.
!> \endverbatim
!> \param[in] X
!> \verbatim
!> X is (input/output) COMPLEX*16 array of dimension at least
!> ( 1 + ( n - 1 )*abs( INCX ) ).
!> Before entry, the incremented array X must contain the n
!> element vector x. On exit, X is overwritten with the
!> transformed vector x.
!> \endverbatim
!> \param[in] INCX
!> \verbatim
!> INCX is INTEGER
!> On entry, INCX specifies the increment for the elements of
!> X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
!> =====
!>

```

```

!> \verbatim
!>
!> Level 2 Blas routine.
!> The vector and matrix arguments are not referenced when N = 0, or M = 0
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!
=====
SUBROUTINE ZTPMV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
INTEGER INCX,N
CHARACTER DIAG,TRANS,UPLO
!
! .. Array Arguments ..
COMPLEX*16 AP(*),X(*)
!
! ..
=====
!
! .. Parameters ..
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
COMPLEX*16 TEMP
INTEGER I,INFO,IX,J,JX,K,KK,KX
LOGICAL NOCONJ,NOUNIT
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC DCONJG
!
! ..
!
! Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
  INFO = 1
ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND.&
  .NOT.LSAME(TRANS,'C')) THEN
  INFO = 2
ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
  INFO = 3
ELSE IF (N.LT.0) THEN
  INFO = 4
ELSE IF (INCX.EQ.0) THEN
  INFO = 7
END IF
IF (INFO.NE.0) THEN
  CALL XERBLA('ZTPMV ',INFO)
  RETURN
END IF
!
! Quick return if possible.
!
IF (N.EQ.0) RETURN
!
NOCONJ = LSAME(TRANS,'T')
NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be ( N - 1 ) * INCX too small for descending loops.
!
IF (INCX.LE.0) THEN
  KX = 1 - (N-1)*INCX
ELSE IF (INCX.NE.1) THEN
  KX = 1
END IF
!
! Start the operations. In this version the elements of AP are
! accessed sequentially with one pass through AP.
!
IF (LSAME(TRANS,'N')) THEN
!
  Form x:= A*x.
!
  IF (LSAME(UPLO,'U')) THEN
    KK = 1
    IF (INCX.EQ.1) THEN
      DO 20 J = 1,N
        IF (X(J).NE.ZERO) THEN
          TEMP = X(J)
          K = KK
          DO 10 I = 1,J - 1
            X(I) = X(I) + TEMP*AP(K)
            K = K + 1
10          CONTINUE
          IF (NOUNIT) X(J) = X(J)*AP(KK+J-1)
        END IF
        KK = KK + J
20      CONTINUE
    ELSE
      JX = KX
      DO 40 J = 1,N
        IF (X(JX).NE.ZERO) THEN
          TEMP = X(JX)
          IX = KX

```

```

DO 30 K = KK, KK + J - 2
  X(IX) = X(IX) + TEMP*AP(K)
  IX = IX + INCX
30  CONTINUE
  IF (NOUNIT) X(JX) = X(JX)*AP(KK+J-1)
  END IF
  JX = JX + INCX
  KK = KK + J
40  CONTINUE
  END IF
ELSE
  KK = (N*(N+1))/2
  IF (INCX.EQ.1) THEN
    DO 60 J = N, 1, -1
      IF (X(J).NE.ZERO) THEN
        TEMP = X(J)
        K = KK
        DO 50 I = N, J + 1, -1
          X(I) = X(I) + TEMP*AP(K)
          K = K - 1
50      CONTINUE
        IF (NOUNIT) X(J) = X(J)*AP(KK-N+J)
        END IF
        KK = KK - (N-J+1)
60      CONTINUE
      ELSE
        KX = KX + (N-1)*INCX
        JX = KX
        DO 80 J = N, 1, -1
          IF (X(JX).NE.ZERO) THEN
            TEMP = X(JX)
            IX = KX
            DO 70 K = KK, KK - (N - (J+1)), -1
              X(IX) = X(IX) + TEMP*AP(K)
              IX = IX - INCX
70          CONTINUE
            IF (NOUNIT) X(JX) = X(JX)*AP(KK-N+J)
            END IF
            JX = JX - INCX
            KK = KK - (N-J+1)
80          CONTINUE
          END IF
        END IF
      ELSE
!
!      Form x := A**T*x or x := A**H*x.
!
      IF (LSAME(UPLO, 'U')) THEN
        KK = (N*(N+1))/2
        IF (INCX.EQ.1) THEN
          DO 110 J = N, 1, -1
            TEMP = X(J)
            K = KK - 1
            IF (NOCNJ) THEN
              IF (NOUNIT) TEMP = TEMP*AP(KK)
              DO 90 I = J - 1, 1, -1
                TEMP = TEMP + AP(K)*X(I)
                K = K - 1
90          CONTINUE
            ELSE
              IF (NOUNIT) TEMP = TEMP*DCONJG(AP(KK))
              DO 100 I = J - 1, 1, -1
                TEMP = TEMP + DCONJG(AP(K))*X(I)
                K = K - 1
100         CONTINUE
            END IF
            X(J) = TEMP
            KK = KK - J
110         CONTINUE
          ELSE
            JX = KX + (N-1)*INCX
            DO 140 J = N, 1, -1
              TEMP = X(JX)
              IX = JX
              IF (NOCNJ) THEN
                IF (NOUNIT) TEMP = TEMP*AP(KK)
                DO 120 K = KK - 1, KK - J + 1, -1
                  IX = IX - INCX
                  TEMP = TEMP + AP(K)*X(IX)
120             CONTINUE
                ELSE
                  IF (NOUNIT) TEMP = TEMP*DCONJG(AP(KK))
                  DO 130 K = KK - 1, KK - J + 1, -1
                    IX = IX - INCX
                    TEMP = TEMP + DCONJG(AP(K))*X(IX)
130             CONTINUE
                  END IF
                  X(JX) = TEMP
                  JX = JX - INCX
                  KK = KK - J
140             CONTINUE
              END IF
            ELSE
              KK = 1
              IF (INCX.EQ.1) THEN
                DO 170 J = 1, N
                  TEMP = X(J)
                  K = KK + 1
                  IF (NOCNJ) THEN
                    IF (NOUNIT) TEMP = TEMP*AP(KK)
                    DO 150 I = J + 1, N
                      TEMP = TEMP + AP(K)*X(I)
                      K = K + 1
150             CONTINUE
                  ELSE
                    IF (NOUNIT) TEMP = TEMP*DCONJG(AP(KK))
                    DO 160 I = J + 1, N
                      TEMP = TEMP + DCONJG(AP(K))*X(I)
                      K = K + 1
160             CONTINUE
                  END IF
                  X(J) = TEMP
                  KK = KK + (N-J+1)
170             CONTINUE

```

```

        ELSE
            JX = KX
            DO 200 J = 1,N
                TEMP = X(JX)
                IX = JX
                IF (NOCONJ) THEN
                    IF (NOUNIT) TEMP = TEMP*AP(KK)
                    DO 180 K = KK + 1, KK + N - J
                        IX = IX + INCX
                        TEMP = TEMP + AP(K)*X(IX)
180                 CONTINUE
                ELSE
                    IF (NOUNIT) TEMP = TEMP*DCONJG(AP(KK))
                    DO 190 K = KK + 1, KK + N - J
                        IX = IX + INCX
                        TEMP = TEMP + DCONJG(AP(K))*X(IX)
190                 CONTINUE
                END IF
                X(JX) = TEMP
                JX = JX + INCX
                KK = KK + (N-J+1)
200            CONTINUE
        END IF
    END IF
END IF
RETURN
!
! End of ZTPMV .
!
END SUBROUTINE
!> \brief \b ZTRSV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZTRSV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,LDA,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> ZTRSV solves one of the systems of equations
!>
!>  $Ax = b$ , or  $A^*Tx = b$ , or  $A^*Hx = b$ ,
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular matrix.
!>
!> No test for singularity or near-singularity is included in this
!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the equations to be solved as
!> follows:
!>
!> TRANS = 'N' or 'n'  $Ax = b$ .
!>
!> TRANS = 'T' or 't'  $A^*Tx = b$ .
!>
!> TRANS = 'C' or 'c'  $A^*Hx = b$ .
!> \endverbatim
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!>

```

```

!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with  UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular matrix and the strictly lower triangular part of
!>     A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular matrix and the strictly upper triangular part of
!>     A is not referenced.
!>     Note that when  DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 )*abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element right-hand side vector b. On exit, X is overwritten
!>     with the solution vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
!   Authors:
!   =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
!   =====
!> \verbatim
!>     Level 2 Blas routine.
!>
!>     -- Written on 22-October-1986.
!>     Jack Dongarra, Argonne National Lab.
!>     Jeremy Du Croz, Nag Central Office.
!>     Sven Hammarling, Nag Central Office.
!>     Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!   =====
!   SUBROUTINE ZTRSV(UPLO,TRANS,DIAG,N,A,LDA,X,INCX)
!
!   -- Reference BLAS level2 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
!   .. Scalar Arguments ..
!   INTEGER INCX,LDA,N
!   CHARACTER DIAG,TRANS,UPLO
!
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),X(*)
!   ..
!
!   =====
!
!   .. Parameters ..
!   COMPLEX*16 ZERO
!   PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
!   ..
!   .. Local Scalars ..
!   COMPLEX*16 TEMP
!   INTEGER I,INFO,IX,J,JX,KX
!   LOGICAL NOCONJ,NOUNIT
!
!   ..
!   .. External Functions ..
!   LOGICAL LSAME
!   EXTERNAL LSAME
!
!   ..
!   .. External Subroutines ..
!   EXTERNAL XERBLA
!
!   ..
!   .. Intrinsic Functions ..
!   INTRINSIC DCONJG,MAX
!
!   ..
!
!   Test the input parameters.
!
!   INFO = 0
!   IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!       INFO = 1
!   ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND. &
!       .NOT.LSAME(TRANS,'C')) THEN
!       INFO = 2
!   ELSE IF (.NOT.LSAME(DIAG,'U') .AND. .NOT.LSAME(DIAG,'N')) THEN
!       INFO = 3
!   ELSE IF (N.LT.0) THEN

```

```

        INFO = 4
    ELSE IF (LDA.LT.MAX(1,N)) THEN
        INFO = 6
    ELSE IF (INCX.EQ.0) THEN
        INFO = 8
    END IF
    IF (INFO.NE.0) THEN
        CALL XERBLA('ZTRSV ',INFO)
        RETURN
    END IF
!
! Quick return if possible.
!
    IF (N.EQ.0) RETURN
!
    NOCONJ = LSAME(TRANS,'T')
    NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
    IF (INCX.LE.0) THEN
        KX = 1 - (N-1)*INCX
    ELSE IF (INCX.NE.1) THEN
        KX = 1
    END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
    IF (LSAME(TRANS,'N')) THEN
!
! Form x := inv( A ) * x.
!
        IF (LSAME(UPLO,'U')) THEN
            IF (INCX.EQ.1) THEN
                DO 20 J = N,1,-1
                    IF (X(J).NE.ZERO) THEN
                        IF (NOUNIT) X(J) = X(J)/A(J,J)
                        TEMP = X(J)
                        DO 10 I = J - 1,1,-1
                            X(I) = X(I) - TEMP*A(I,J)
10                     CONTINUE
                        END IF
20                     CONTINUE
                    ELSE
                        JX = KX + (N-1)*INCX
                        DO 40 J = N,1,-1
                            IF (X(JX).NE.ZERO) THEN
                                IF (NOUNIT) X(JX) = X(JX)/A(J,J)
                                TEMP = X(JX)
                                IX = JX
                                DO 30 I = J - 1,1,-1
                                    IX = IX - INCX
                                    X(IX) = X(IX) - TEMP*A(I,J)
30                             CONTINUE
                                END IF
                                JX = JX - INCX
40                             CONTINUE
                            END IF
                        ELSE
                            IF (INCX.EQ.1) THEN
                                DO 60 J = 1,N
                                    IF (X(J).NE.ZERO) THEN
                                        IF (NOUNIT) X(J) = X(J)/A(J,J)
                                        TEMP = X(J)
                                        DO 50 I = J + 1,N
                                            X(I) = X(I) - TEMP*A(I,J)
50                                     CONTINUE
                                    END IF
                                    CONTINUE
60                                 ELSE
                                    JX = KX
                                    DO 80 J = 1,N
                                        IF (X(JX).NE.ZERO) THEN
                                            IF (NOUNIT) X(JX) = X(JX)/A(J,J)
                                            TEMP = X(JX)
                                            IX = JX
                                            DO 70 I = J + 1,N
                                                IX = IX + INCX
                                                X(IX) = X(IX) - TEMP*A(I,J)
70                                             CONTINUE
                                            END IF
                                            JX = JX + INCX
80                                             CONTINUE
                                        END IF
                                    END IF
                                ELSE
110                                 CONTINUE
                                ELSE
                                    JX = KX
                                    DO 140 J = 1,N
                                        IX = KX
                                        TEMP = X(JX)
                                        IF (NOCONJ) THEN

```

```

DO 120 I = 1, J - 1
    TEMP = TEMP - A(I, J) * X(IX)
    IX = IX + INCX
120    CONTINUE
    IF (NOUNIT) TEMP = TEMP/A(J, J)
ELSE
    DO 130 I = 1, J - 1
        TEMP = TEMP - DCONJG(A(I, J)) * X(IX)
        IX = IX + INCX
130    CONTINUE
    IF (NOUNIT) TEMP = TEMP/DCONJG(A(J, J))
END IF
X(JX) = TEMP
JX = JX + INCX
140    CONTINUE
END IF
ELSE
    IF (INCX.EQ.1) THEN
        DO 170 J = N, 1, -1
            TEMP = X(J)
            IF (NOCONJ) THEN
                DO 150 I = N, J + 1, -1
                    TEMP = TEMP - A(I, J) * X(I)
150                CONTINUE
                IF (NOUNIT) TEMP = TEMP/A(J, J)
            ELSE
                DO 160 I = N, J + 1, -1
                    TEMP = TEMP - DCONJG(A(I, J)) * X(I)
160                CONTINUE
                IF (NOUNIT) TEMP = TEMP/DCONJG(A(J, J))
            END IF
            X(J) = TEMP
170        CONTINUE
    ELSE
        KX = KX + (N-1) * INCX
        JX = KX
        DO 200 J = N, 1, -1
            IX = KX
            TEMP = X(JX)
            IF (NOCONJ) THEN
                DO 180 I = N, J + 1, -1
                    TEMP = TEMP - A(I, J) * X(IX)
                    IX = IX - INCX
180                CONTINUE
                IF (NOUNIT) TEMP = TEMP/A(J, J)
            ELSE
                DO 190 I = N, J + 1, -1
                    TEMP = TEMP - DCONJG(A(I, J)) * X(IX)
                    IX = IX - INCX
190                CONTINUE
                IF (NOUNIT) TEMP = TEMP/DCONJG(A(J, J))
            END IF
            X(JX) = TEMP
            JX = JX - INCX
200        CONTINUE
    END IF
END IF
END IF
!
! RETURN
!
! End of ZTRSV .
!
! END SUBROUTINE
!
! *> \brief \b ZTBSV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZTBSV(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX, K, LDA, N
! CHARACTER DIAG, TRANS, UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA, *), X(*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!>
!> ZTBSV solves one of the systems of equations
!>
!>   A*x = b,   or   A**T*x = b,   or   A**H*x = b,
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular band matrix, with ( k + 1 )
!> diagonals.
!>
!> No test for singularity or near-singularity is included in this
!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!>   UPLO is CHARACTER*1
!>   On entry, UPLO specifies whether the matrix is an upper or
!>   lower triangular matrix as follows:
!>
!>       UPLO = 'U' or 'u'   A is an upper triangular matrix.
!>
!>       UPLO = 'L' or 'l'   A is a lower triangular matrix.
!>

```

```

!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the equations to be solved as
!>     follows:
!>
!>         TRANS = 'N' or 'n'    A*x = b.
!>
!>         TRANS = 'T' or 't'    A**T*x = b.
!>
!>         TRANS = 'C' or 'c'    A**H*x = b.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit
!>     triangular as follows:
!>
!>         DIAG = 'U' or 'u'    A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'    A is not assumed to be unit
!>         triangular.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with UPLO = 'U' or 'u', K specifies the number of
!>     super-diagonals of the matrix A.
!>     On entry with UPLO = 'L' or 'l', K specifies the number of
!>     sub-diagonals of the matrix A.
!>     K must satisfy 0 .le. K.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading ( k + 1 )
!>     by n part of the array A must contain the upper triangular
!>     band part of the matrix of coefficients, supplied column by
!>     column, with the leading diagonal of the matrix in row
!>     ( k + 1 ) of the array, the first super-diagonal starting at
!>     position 2 in row k, and so on. The top left k by k triangle
!>     of the array A is not referenced.
!>     The following program segment will transfer an upper
!>     triangular band matrix from conventional full matrix storage
!>     to band storage:
!>
!>         DO 20, J = 1, N
!>             M = K + 1 - J
!>             DO 10, I = MAX( 1, J - K ), J
!>                 A( M + I, J ) = matrix( I, J )
!>             10 CONTINUE
!>             20 CONTINUE
!>
!>     Before entry with UPLO = 'L' or 'l', the leading ( k + 1 )
!>     by n part of the array A must contain the lower triangular
!>     band part of the matrix of coefficients, supplied column by
!>     column, with the leading diagonal of the matrix in row 1 of
!>     the array, the first sub-diagonal starting at position 1 in
!>     row 2, and so on. The bottom right k by k triangle of the
!>     array A is not referenced.
!>     The following program segment will transfer a lower
!>     triangular band matrix from conventional full matrix storage
!>     to band storage:
!>
!>         DO 20, J = 1, N
!>             M = 1 - J
!>             DO 10, I = J, MIN( N, J + K )
!>                 A( M + I, J ) = matrix( I, J )
!>             10 CONTINUE
!>             20 CONTINUE
!>
!>     Note that when DIAG = 'U' or 'u' the elements of the array A
!>     corresponding to the diagonal elements of the matrix are not
!>     referenced, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     ( k + 1 ).
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element right-hand side vector b. On exit, X is overwritten
!>     with the solution vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====

```

```

!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
! =====
! SUBROUTINE ZTBSV(UPLO,TRANS,DIAG,N,K,A,LDA,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,K,LDA,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*)
! ..
!
! =====
! .. Parameters ..
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,IX,J,JX,KPLUS1,KX,L
! LOGICAL NOCONJ,NOUNIT
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX,MIN
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U')) .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (.NOT.LSAME(TRANS,'N')) .AND. .NOT.LSAME(TRANS,'T')) .AND. &
!   .NOT.LSAME(TRANS,'C')) THEN
!   INFO = 2
! ELSE IF (.NOT.LSAME(DIAG,'U')) .AND. .NOT.LSAME(DIAG,'N')) THEN
!   INFO = 3
! ELSE IF (N.LT.0) THEN
!   INFO = 4
! ELSE IF (K.LT.0) THEN
!   INFO = 5
! ELSE IF (LDA.LT. (K+1)) THEN
!   INFO = 7
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZTBSV ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF (N.EQ.0) RETURN
!
! NOCONJ = LSAME(TRANS,'T')
! NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be ( N - 1 ) * INCX too small for descending loops.
!
! IF (INCX.LE.0) THEN
!   KX = 1 - (N-1)*INCX
! ELSE IF (INCX.NE.1) THEN
!   KX = 1
! END IF
!
! Start the operations. In this version the elements of A are
! accessed by sequentially with one pass through A.
!
! IF (LSAME(TRANS,'N')) THEN
!
!   Form x := inv( A ) * x.
!
!   IF (LSAME(UPLO,'U')) THEN
!     KPLUS1 = K + 1
!     IF (INCX.EQ.1) THEN
!       DO 20 J = N,1,-1
!         IF (X(J).NE.ZERO) THEN

```

```

                L = KPLUS1 - J
                IF (NOUNIT) X(J) = X(J)/A(KPLUS1,J)
                TEMP = X(J)
                DO 10 I = J - 1,MAX(1,J-K),-1
                    X(I) = X(I) - TEMP*A(L+I,J)
10                CONTINUE
                END IF
20                CONTINUE
            ELSE
                KX = KX + (N-1)*INCX
                JX = KX
                DO 40 J = N,1,-1
                    KX = KX - INCX
                    IF (X(JX).NE.ZERO) THEN
                        IX = KX
                        L = KPLUS1 - J
                        IF (NOUNIT) X(JX) = X(JX)/A(KPLUS1,J)
                        TEMP = X(JX)
                        DO 30 I = J - 1,MAX(1,J-K),-1
                            X(IX) = X(IX) - TEMP*A(L+I,J)
30                        IX = IX - INCX
                        CONTINUE
                    END IF
                    JX = JX - INCX
40                CONTINUE
            END IF
        ELSE
            IF (INCX.EQ.1) THEN
                DO 60 J = 1,N
                    IF (X(J).NE.ZERO) THEN
                        L = 1 - J
                        IF (NOUNIT) X(J) = X(J)/A(1,J)
                        TEMP = X(J)
                        DO 50 I = J + 1,MIN(N,J+K)
                            X(I) = X(I) - TEMP*A(L+I,J)
50                        CONTINUE
                    END IF
                    CONTINUE
60                CONTINUE
            ELSE
                JX = KX
                DO 80 J = 1,N
                    KX = KX + INCX
                    IF (X(JX).NE.ZERO) THEN
                        IX = KX
                        L = 1 - J
                        IF (NOUNIT) X(JX) = X(JX)/A(1,J)
                        TEMP = X(JX)
                        DO 70 I = J + 1,MIN(N,J+K)
                            X(IX) = X(IX) - TEMP*A(L+I,J)
70                        IX = IX + INCX
                        CONTINUE
                    END IF
                    JX = JX + INCX
80                CONTINUE
            END IF
        END IF
    ELSE
        !
        !      Form x := inv( A**T ) * x  or  x := inv( A**H ) * x.
        !
        IF (LSAME(UPLO,'U')) THEN
            KPLUS1 = K + 1
            IF (INCX.EQ.1) THEN
                DO 110 J = 1,N
                    TEMP = X(J)
                    L = KPLUS1 - J
                    IF (NOCNJJ) THEN
                        DO 90 I = MAX(1,J-K),J - 1
                            TEMP = TEMP - A(L+I,J)*X(I)
90                        CONTINUE
                        IF (NOUNIT) TEMP = TEMP/A(KPLUS1,J)
                    ELSE
                        DO 100 I = MAX(1,J-K),J - 1
                            TEMP = TEMP - DCONJG(A(L+I,J))*X(I)
100                        CONTINUE
                        IF (NOUNIT) TEMP = TEMP/DCONJG(A(KPLUS1,J))
                    END IF
                    X(J) = TEMP
110                CONTINUE
            ELSE
                JX = KX
                DO 140 J = 1,N
                    TEMP = X(JX)
                    IX = KX
                    L = KPLUS1 - J
                    IF (NOCNJJ) THEN
                        DO 120 I = MAX(1,J-K),J - 1
                            TEMP = TEMP - A(L+I,J)*X(IX)
120                        IX = IX + INCX
                        CONTINUE
                        IF (NOUNIT) TEMP = TEMP/A(KPLUS1,J)
                    ELSE
                        DO 130 I = MAX(1,J-K),J - 1
                            TEMP = TEMP - DCONJG(A(L+I,J))*X(IX)
130                        IX = IX + INCX
                        CONTINUE
                        IF (NOUNIT) TEMP = TEMP/DCONJG(A(KPLUS1,J))
                    END IF
                    X(JX) = TEMP
                    JX = JX + INCX
                    IF (J.GT.K) KX = KX + INCX
140                CONTINUE
            END IF
        ELSE
            IF (INCX.EQ.1) THEN
                DO 170 J = N,1,-1
                    TEMP = X(J)
                    L = 1 - J
                    IF (NOCNJJ) THEN
                        DO 150 I = MIN(N,J+K),J + 1,-1
                            TEMP = TEMP - A(L+I,J)*X(I)
150                        CONTINUE
                        IF (NOUNIT) TEMP = TEMP/A(1,J)
                    ELSE

```

```

DO 160 I = MIN(N,J+K),J + 1,-1
    TEMP = TEMP - DCONJG(A(L+I,J))*X(I)
160    CONTINUE
    IF (NOUNIT) TEMP = TEMP/DCONJG(A(1,J))
    END IF
    X(J) = TEMP
170    CONTINUE
    ELSE
        KX = KX + (N-1)*INCX
        JX = KX
        DO 200 J = N,1,-1
            TEMP = X(JX)
            IX = KX
            L = 1 - J
            IF (NOCNJJ) THEN
                DO 180 I = MIN(N,J+K),J + 1,-1
                    TEMP = TEMP - A(L+I,J)*X(IX)
                    IX = IX - INCX
180                CONTINUE
                IF (NOUNIT) TEMP = TEMP/A(1,J)
            ELSE
                DO 190 I = MIN(N,J+K),J + 1,-1
                    TEMP = TEMP - DCONJG(A(L+I,J))*X(IX)
                    IX = IX - INCX
190                CONTINUE
                IF (NOUNIT) TEMP = TEMP/DCONJG(A(1,J))
            END IF
            X(JX) = TEMP
            JX = JX - INCX
            IF ((N-J).GE.K) KX = KX - INCX
200        CONTINUE
    END IF
END IF
END IF
!
RETURN
!
End of ZTBSV .
!
END SUBROUTINE
!
*> \brief \b ZTPSV
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZTPSV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! .. Scalar Arguments ..
! INTEGER INCX,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 AP(*),X(*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZTPSV solves one of the systems of equations
!>
!>  $A*x = b$ , or  $A**T*x = b$ , or  $A**H*x = b$ ,
!>
!> where b and x are n element vectors and A is an n by n unit, or
!> non-unit, upper or lower triangular matrix, supplied in packed form.
!>
!> No test for singularity or near-singularity is included in this
!> routine. Such tests must be performed before calling this routine.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the equations to be solved as
!> follows:
!>
!> TRANS = 'N' or 'n'  $A*x = b$ .
!>
!> TRANS = 'T' or 't'  $A**T*x = b$ .
!>
!> TRANS = 'C' or 'c'  $A**H*x = b$ .
!> \endverbatim
!
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit
!> triangular as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.

```

```

!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] AP
!> \verbatim
!>     AP is COMPLEX*16 array of DIMENSION at least
!>       ( ( n*( n + 1 ) )/2 ).
!>     Before entry with  UPLO = 'U' or 'u', the array AP must
!>     contain the upper triangular matrix packed sequentially,
!>     column by column, so that AP( 1 ) contains a( 1, 1 ),
!>     AP( 2 ) and AP( 3 ) contain a( 1, 2 ) and a( 2, 2 )
!>     respectively, and so on.
!>     Before entry with UPLO = 'L' or 'l', the array AP must
!>     contain the lower triangular matrix packed sequentially,
!>     column by column, so that AP( 1 ) contains a( 1, 1 ),
!>     AP( 2 ) and AP( 3 ) contain a( 2, 1 ) and a( 3, 1 )
!>     respectively, and so on.
!>     Note that when  DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in,out] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>       ( 1 + ( n - 1 ) *abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element right-hand side vector b. On exit, X is overwritten
!>     with the solution vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!
! =====
!
! SUBROUTINE ZTPSV(UPLO,TRANS,DIAG,N,AP,X,INCX)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! INTEGER INCX,N
! CHARACTER DIAG,TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 AP(*),X(*)
! ..
! =====
!
! .. Parameters ..
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,IX,J,JX,K,KK,KX
! LOGICAL NOCONJ,NOUNIT
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (.NOT.LSAME(TRANS,'N') .AND. .NOT.LSAME(TRANS,'T') .AND. &
!   .NOT.LSAME(TRANS,'C')) THEN

```

```

        INFO = 2
    ELSE IF (.NOT.LSAME(DIAG,'U')) .AND. .NOT.LSAME(DIAG,'N')) THEN
        INFO = 3
    ELSE IF (N.LT.0) THEN
        INFO = 4
    ELSE IF (INCX.EQ.0) THEN
        INFO = 7
    END IF
    IF (INFO.NE.0) THEN
        CALL XERBLA('ZTPSV ',INFO)
        RETURN
    END IF
!
! Quick return if possible.
!
    IF (N.EQ.0) RETURN
!
    NOCONJ = LSAME(TRANS,'T')
    NOUNIT = LSAME(DIAG,'N')
!
! Set up the start point in X if the increment is not unity. This
! will be (N-1)*INCX too small for descending loops.
!
    IF (INCX.LE.0) THEN
        KX = 1 - (N-1)*INCX
    ELSE IF (INCX.NE.1) THEN
        KK = 1
    END IF
!
! Start the operations. In this version the elements of AP are
! accessed sequentially with one pass through AP.
!
    IF (LSAME(TRANS,'N')) THEN
!
        Form x := inv( A ) * x.
!
        IF (LSAME(UPLO,'U')) THEN
            KK = (N*(N+1))/2
            IF (INCX.EQ.1) THEN
                DO 20 J = N,1,-1
                    IF (X(J).NE.ZERO) THEN
                        IF (NOUNIT) X(J) = X(J)/AP(KK)
                        TEMP = X(J)
                        K = KK - 1
                        DO 10 I = J - 1,1,-1
                            X(I) = X(I) - TEMP*AP(K)
                            K = K - 1
10                        CONTINUE
                        END IF
                        KK = KK - J
20                    CONTINUE
                ELSE
                    JX = KX + (N-1)*INCX
                    DO 40 J = N,1,-1
                        IF (X(JX).NE.ZERO) THEN
                            IF (NOUNIT) X(JX) = X(JX)/AP(KK)
                            TEMP = X(JX)
                            IX = JX
                            DO 30 K = KK - 1, KK - J + 1, -1
                                IX = IX - INCX
                                X(IX) = X(IX) - TEMP*AP(K)
30                            CONTINUE
                            END IF
                            JX = JX - INCX
                            KK = KK - J
40                        CONTINUE
                    END IF
                ELSE
                    KK = 1
                    IF (INCX.EQ.1) THEN
                        DO 60 J = 1,N
                            IF (X(J).NE.ZERO) THEN
                                IF (NOUNIT) X(J) = X(J)/AP(KK)
                                TEMP = X(J)
                                K = KK + 1
                                DO 50 I = J + 1,N
                                    X(I) = X(I) - TEMP*AP(K)
                                    K = K + 1
50                                CONTINUE
                                END IF
                                KK = KK + (N-J+1)
60                            CONTINUE
                        ELSE
                            JX = KX
                            DO 80 J = 1,N
                                IF (X(JX).NE.ZERO) THEN
                                    IF (NOUNIT) X(JX) = X(JX)/AP(KK)
                                    TEMP = X(JX)
                                    IX = JX
                                    DO 70 K = KK + 1, KK + N - J
                                        IX = IX + INCX
                                        X(IX) = X(IX) - TEMP*AP(K)
70                                    CONTINUE
                                END IF
                                JX = JX + INCX
                                KK = KK + (N-J+1)
80                            CONTINUE
                        END IF
                    END IF
                ELSE
!
                    Form x := inv( A**T ) * x or x := inv( A**H ) * x.
!
                    IF (LSAME(UPLO,'U')) THEN
                        KK = 1
                        IF (INCX.EQ.1) THEN
                            DO 110 J = 1,N
                                TEMP = X(J)
                                K = KK
                                IF (NOCONJ) THEN
                                    DO 90 I = 1, J - 1
                                        TEMP = TEMP - AP(K)*X(I)
                                        K = K + 1
90                                    CONTINUE

```

```

        IF (NOUNIT) TEMP = TEMP/AP(KK+J-1)
      ELSE
        DO 100 I = 1,J - 1
          TEMP = TEMP - DCONJG(AP(K)) * X(I)
          K = K + 1
100      CONTINUE
        IF (NOUNIT) TEMP = TEMP/DCONJG(AP(KK+J-1))
      END IF
      X(J) = TEMP
      KK = KK + J
110    CONTINUE
  ELSE
    JX = KX
    DO 140 J = 1,N
      TEMP = X(JX)
      IX = KX
      IF (NOCONJ) THEN
        DO 120 K = KK, KK + J - 2
          TEMP = TEMP - AP(K) * X(IX)
          IX = IX + INCX
120      CONTINUE
        IF (NOUNIT) TEMP = TEMP/AP(KK+J-1)
      ELSE
        DO 130 K = KK, KK + J - 2
          TEMP = TEMP - DCONJG(AP(K)) * X(IX)
          IX = IX + INCX
130      CONTINUE
        IF (NOUNIT) TEMP = TEMP/DCONJG(AP(KK+J-1))
      END IF
      X(JX) = TEMP
      JX = JX + INCX
      KK = KK + J
140    CONTINUE
  END IF
ELSE
  KK = (N* (N+1))/2
  IF (INCX.EQ.1) THEN
    DO 170 J = N,1,-1
      TEMP = X(J)
      K = KK
      IF (NOCONJ) THEN
        DO 150 I = N,J + 1,-1
          TEMP = TEMP - AP(K) * X(I)
          K = K - 1
150      CONTINUE
        IF (NOUNIT) TEMP = TEMP/AP(KK-N+J)
      ELSE
        DO 160 I = N,J + 1,-1
          TEMP = TEMP - DCONJG(AP(K)) * X(I)
          K = K - 1
160      CONTINUE
        IF (NOUNIT) TEMP = TEMP/DCONJG(AP(KK-N+J))
      END IF
      X(J) = TEMP
      KK = KK - (N-J+1)
170    CONTINUE
  ELSE
    KX = KX + (N-1)*INCX
    JX = KX
    DO 200 J = N,1,-1
      TEMP = X(JX)
      IX = KX
      IF (NOCONJ) THEN
        DO 180 K = KK, KK - (N- (J+1)), -1
          TEMP = TEMP - AP(K) * X(IX)
          IX = IX - INCX
180      CONTINUE
        IF (NOUNIT) TEMP = TEMP/AP(KK-N+J)
      ELSE
        DO 190 K = KK, KK - (N- (J+1)), -1
          TEMP = TEMP - DCONJG(AP(K)) * X(IX)
          IX = IX - INCX
190      CONTINUE
        IF (NOUNIT) TEMP = TEMP/DCONJG(AP(KK-N+J))
      END IF
      X(JX) = TEMP
      JX = JX - INCX
      KK = KK - (N-J+1)
200    CONTINUE
  END IF
END IF
END IF
RETURN
!
!   End of ZTPSV .
!
!   END SUBROUTINE
!   *> \brief \b ZGERC
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZGERC(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!
!   .. Scalar Arguments ..
!   COMPLEX*16 ALPHA
!   INTEGER INCX,INCY,LDA,M,N
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),X(*),Y(*)
!   ..
!
!> \par Purpose:
!> =====
!> \verbatim
!>

```

```

!> ZGERC performs the rank 1 operation
!>
!>   A := alpha*x*y**H + A,
!>
!> where alpha is a scalar, x is an m element vector, y is an n element
!> vector and A is an m by n matrix.
!> \endverbatim
!>
!> Arguments:
!> =====
!>
!> \param[in] M
!> \verbatim
!>   M is INTEGER
!>   On entry, M specifies the number of rows of the matrix A.
!>   M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>   N is INTEGER
!>   On entry, N specifies the number of columns of the matrix A.
!>   N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>   ALPHA is COMPLEX*16
!>   On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>   X is COMPLEX*16 array of dimension at least
!>   ( 1 + ( m - 1 ) *abs( INCX ) ).
!>   Before entry, the incremented array X must contain the m
!>   element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>   INCX is INTEGER
!>   On entry, INCX specifies the increment for the elements of
!>   X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!>   Y is COMPLEX*16 array of dimension at least
!>   ( 1 + ( n - 1 ) *abs( INCY ) ).
!>   Before entry, the incremented array Y must contain the n
!>   element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>   INCY is INTEGER
!>   On entry, INCY specifies the increment for the elements of
!>   Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!>   A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>   Before entry, the leading m by n part of the array A must
!>   contain the matrix of coefficients. On exit, A is
!>   overwritten by the updated matrix.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>   LDA is INTEGER
!>   On entry, LDA specifies the first dimension of A as declared
!>   in the calling (sub) program. LDA must be at least
!>   max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>   Level 2 Blas routine.
!>
!>   -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZGERC(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ALPHA
!> INTEGER INCX, INCY, LDA, M, N

```

```

! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*),Y(*)
! ..
!
! =====
! .. Parameters ..
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,IX,J,JY,KX
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (M.LT.0) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 5
! ELSE IF (INCY.EQ.0) THEN
!   INFO = 7
! ELSE IF (LDA.LT.MAX(1,M)) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZGERC ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((M.EQ.0) .OR. (N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
! IF (INCY.GT.0) THEN
!   JY = 1
! ELSE
!   JY = 1 - (N-1)*INCY
! END IF
! IF (INCX.EQ.1) THEN
!   DO 20 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*DCONJG(Y(JY))
!       DO 10 I = 1,M
!         A(I,J) = A(I,J) + X(I)*TEMP
!10      CONTINUE
!       END IF
!       JY = JY + INCY
!20    CONTINUE
! ELSE
!   IF (INCX.GT.0) THEN
!     KX = 1
!   ELSE
!     KX = 1 - (M-1)*INCX
!   END IF
!   DO 40 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*DCONJG(Y(JY))
!       IX = KX
!       DO 30 I = 1,M
!         A(I,J) = A(I,J) + X(IX)*TEMP
!30      IX = IX + INCX
!       CONTINUE
!     END IF
!     JY = JY + INCY
!40    CONTINUE
! END IF
!
! RETURN
!
! End of ZGERC .
!
! END SUBROUTINE
! *> \brief \b ZGERU
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZGERU(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA
! INTEGER INCX,INCY,LDA,M,N
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*),Y(*)
! ..
!
! !> \par Purpose:
! ! =====
! !> \verbatim
! !>
! !> ZGERU performs the rank 1 operation

```

```

!>
!>   A := alpha*x*y**T + A,
!>
!> where alpha is a scalar, x is an m element vector, y is an n element
!> vector and A is an m by n matrix.
!> \endverbatim
!>
!> Arguments:
!> =====
!> \param[in] M
!> \verbatim
!>     M is INTEGER
!>     On entry, M specifies the number of rows of the matrix A.
!>     M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the number of columns of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( m - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the m
!>     element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!>     Y is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry, the leading m by n part of the array A must
!>     contain the matrix of coefficients. On exit, A is
!>     overwritten by the updated matrix.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!> \verbatim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>   Jack Dongarra, Argonne National Lab.
!>   Jeremy Du Croz, Nag Central Office.
!>   Sven Hammarling, Nag Central Office.
!>   Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!>
!> =====
!> SUBROUTINE ZGERU(M,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!> .. Scalar Arguments ..
!> COMPLEX*16 ALPHA
!> INTEGER INCX,INCY,LDA,M,N
!> ..

```

```

! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*),Y(*)
! ..
! =====
! .. Parameters ..
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,IX,J,JY,KX
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (M.LT.0) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 5
! ELSE IF (INCY.EQ.0) THEN
!   INFO = 7
! ELSE IF (LDA.LT.MAX(1,M)) THEN
!   INFO = 9
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZGERU ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((M.EQ.0) .OR. (N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through A.
!
! IF (INCY.GT.0) THEN
!   JY = 1
! ELSE
!   JY = 1 - (N-1)*INCY
! END IF
! IF (INCX.EQ.1) THEN
!   DO 20 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*Y(JY)
!       DO 10 I = 1,M
!         A(I,J) = A(I,J) + X(I)*TEMP
!10      CONTINUE
!       END IF
!       JY = JY + INCY
!20    CONTINUE
! ELSE
!   IF (INCX.GT.0) THEN
!     KX = 1
!   ELSE
!     KX = 1 - (M-1)*INCX
!   END IF
!   DO 40 J = 1,N
!     IF (Y(JY).NE.ZERO) THEN
!       TEMP = ALPHA*Y(JY)
!       IX = KX
!       DO 30 I = 1,M
!         A(I,J) = A(I,J) + X(IX)*TEMP
!30      CONTINUE
!       IX = IX + INCX
!     END IF
!     JY = JY + INCY
!40    CONTINUE
! END IF
!
! RETURN
!
! End of ZGERU .
!
! END SUBROUTINE
! *> \brief \b ZHER
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZHER(UPLO,N,ALPHA,X,INCX,A,LDA)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX,LDA,N
! CHARACTER UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),X(*)
! ..
!
!> \par Purpose:
!> =====
!> \verbatim
!>
!> ZHER performs the hermitian rank 1 operation

```

```

!>
!>   A := alpha*x*x**H + A,
!>
!> where alpha is a real scalar, x is an n element vector and A is an
!> n by n hermitian matrix.
!> \endverbatim
!>
!> Arguments:
!> =====
!>
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the upper or lower
!>     triangular part of the array A is to be referenced as
!>     follows:
!>
!>         UPLO = 'U' or 'u'   Only the upper triangular part of A
!>         is to be referenced.
!>
!>         UPLO = 'L' or 'l'   Only the lower triangular part of A
!>         is to be referenced.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix A.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is DOUBLE PRECISION.
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) *abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>     Before entry with  UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array A must contain the upper
!>     triangular part of the hermitian matrix and the strictly
!>     lower triangular part of A is not referenced. On exit, the
!>     upper triangular part of the array A is overwritten by the
!>     upper triangular part of the updated matrix.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array A must contain the lower
!>     triangular part of the hermitian matrix and the strictly
!>     upper triangular part of A is not referenced. On exit, the
!>     lower triangular part of the array A is overwritten by the
!>     lower triangular part of the updated matrix.
!>     Note that the imaginary parts of the diagonal elements need
!>     not be set, they are assumed to be zero, and on exit they
!>     are set to zero.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. LDA must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level2
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>     Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>    Jack Dongarra, Argonne National Lab.
!>    Jeremy Du Croz, Nag Central Office.
!>    Sven Hammarling, Nag Central Office.
!>    Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZHER(UPLO,N,ALPHA,X,INCX,A,LDA)
!>
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!> November 2011
!>
!>

```

```

! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA
INTEGER INCX, LDA, N
CHARACTER UPLO
! ..
! .. Array Arguments ..
COMPLEX*16 A(LDA,*),X(*)
! ..
!
=====
! .. Parameters ..
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
! .. Local Scalars ..
COMPLEX*16 TEMP
INTEGER I, INFO, IX, J, JX, KX
! ..
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
INTRINSIC DBLE, DCONJG, MAX
! ..
!
Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
    INFO = 1
ELSE IF (N.LT.0) THEN
    INFO = 2
ELSE IF (INCX.EQ.0) THEN
    INFO = 5
ELSE IF (LDA.LT.MAX(1,N)) THEN
    INFO = 7
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZHER ',INFO)
    RETURN
END IF
!
Quick return if possible.
!
IF ((N.EQ.0) .OR. (ALPHA.EQ.DBLE(ZERO))) RETURN
!
Set the start point in X if the increment is not unity.
!
IF (INCX.LE.0) THEN
    KX = 1 - (N-1)*INCX
ELSE IF (INCX.NE.1) THEN
    KX = 1
END IF
!
Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
IF (LSAME(UPLO,'U')) THEN
!
    Form A when A is stored in upper triangle.
!
    IF (INCX.EQ.1) THEN
        DO 20 J = 1,N
            IF (X(J).NE.ZERO) THEN
                TEMP = ALPHA*DCONJG(X(J))
                DO 10 I = 1, J - 1
                    A(I,J) = A(I,J) + X(I)*TEMP
10                CONTINUE
                A(J,J) = DBLE(A(J,J)) + DBLE(X(J)*TEMP)
            ELSE
                A(J,J) = DBLE(A(J,J))
            END IF
20        CONTINUE
    ELSE
        JX = KX
        DO 40 J = 1,N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*DCONJG(X(JX))
                IX = KX
                DO 30 I = 1, J - 1
                    A(I,J) = A(I,J) + X(IX)*TEMP
                    IX = IX + INCX
30                CONTINUE
                A(J,J) = DBLE(A(J,J)) + DBLE(X(JX)*TEMP)
            ELSE
                A(J,J) = DBLE(A(J,J))
            END IF
            JX = JX + INCX
40        CONTINUE
    END IF
ELSE
!
    Form A when A is stored in lower triangle.
!
    IF (INCX.EQ.1) THEN
        DO 60 J = 1,N
            IF (X(J).NE.ZERO) THEN
                TEMP = ALPHA*DCONJG(X(J))
                A(J,J) = DBLE(A(J,J)) + DBLE(TEMP*X(J))
                DO 50 I = J + 1,N
                    A(I,J) = A(I,J) + X(I)*TEMP
50                CONTINUE
            ELSE
                A(J,J) = DBLE(A(J,J))
            END IF
60        CONTINUE
    ELSE
        JX = KX

```

```

      DO 80 J = 1,N
        IF (X(JX).NE.ZERO) THEN
          TEMP = ALPHA*DCONJG(X(JX))
          A(J,J) = DBLE(A(J,J)) + DBLE(TEMP*X(JX))
          IX = JX
          DO 70 I = J + 1,N
            IX = IX + INCX
            A(I,J) = A(I,J) + X(IX)*TEMP
70          CONTINUE
        ELSE
          A(J,J) = DBLE(A(J,J))
        END IF
        JX = JX + INCX
80      CONTINUE
    END IF
  END IF
!
!   RETURN
!
!   End of ZHER .
!
!   END SUBROUTINE
!   *> \brief \b ZHPR
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZHPR(UPLO,N,ALPHA,X,INCX,AP)
!
!   .. Scalar Arguments ..
!   DOUBLE PRECISION ALPHA
!   INTEGER INCX,N
!   CHARACTER UPLO
!   ..
!   .. Array Arguments ..
!   COMPLEX*16 AP(*),X(*)
!   ..
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!> ZHPR   performs the hermitian rank 1 operation
!>
!>   A := alpha*x*x**H + A,
!>
!> where alpha is a real scalar, x is an n element vector and A is an
!> n by n hermitian matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!>   UPLO is CHARACTER*1
!>   On entry, UPLO specifies whether the upper or lower
!>   triangular part of the matrix A is supplied in the packed
!>   array AP as follows:
!>
!>       UPLO = 'U' or 'u'   The upper triangular part of A is
!>                           supplied in AP.
!>
!>       UPLO = 'L' or 'l'   The lower triangular part of A is
!>                           supplied in AP.
!> \endverbatim
!
!> \param[in] N
!> \verbatim
!>   N is INTEGER
!>   On entry, N specifies the order of the matrix A.
!>   N must be at least zero.
!> \endverbatim
!
!> \param[in] ALPHA
!> \verbatim
!>   ALPHA is DOUBLE PRECISION.
!>   On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!
!> \param[in] X
!> \verbatim
!>   X is COMPLEX*16 array of dimension at least
!>   ( 1 + ( n - 1 ) *abs( INCX ) ).
!>   Before entry, the incremented array X must contain the n
!>   element vector x.
!> \endverbatim
!
!> \param[in] INCX
!> \verbatim
!>   INCX is INTEGER
!>   On entry, INCX specifies the increment for the elements of
!>   X. INCX must not be zero.
!> \endverbatim
!
!> \param[in,out] AP
!> \verbatim
!>   AP is COMPLEX*16 array of DIMENSION at least
!>   ( ( n*( n + 1 ) )/2 ).
!>   Before entry with  UPLO = 'U' or 'u', the array AP must
!>   contain the upper triangular part of the hermitian matrix
!>   packed sequentially, column by column, so that AP( 1 )
!>   contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>   and a( 2, 2 ) respectively, and so on. On exit, the array
!>   AP is overwritten by the upper triangular part of the
!>   updated matrix.
!>   Before entry with UPLO = 'L' or 'l', the array AP must

```

```

!>      contain the lower triangular part of the hermitian matrix
!>      packed sequentially, column by column, so that AP( 1 )
!>      contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>      and a( 3, 1 ) respectively, and so on. On exit, the array
!>      AP is overwritten by the lower triangular part of the
!>      updated matrix.
!>      Note that the imaginary parts of the diagonal elements need
!>      not be set, they are assumed to be zero, and on exit they
!>      are set to zero.
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!> Jack Dongarra, Argonne National Lab.
!> Jeremy Du Croz, Nag Central Office.
!> Sven Hammarling, Nag Central Office.
!> Richard Hanson, Sandia National Labs.
!> \endverbatim
!
! =====
!
! SUBROUTINE ZHPR(UPLO,N,ALPHA,X,INCX,AP)
!
! -- Reference BLAS level2 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA
! INTEGER INCX,N
! CHARACTER UPLO
!
! ..
! .. Array Arguments ..
! COMPLEX*16 AP(*),X(*)
! ..
!
! =====
!
! .. Parameters ..
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,IX,J,JX,K,KK,KX
! ..
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DBLE,DCONJG
! ..
!
! Test the input parameters.
!
! INFO = 0
! IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
!   INFO = 1
! ELSE IF (N.LT.0) THEN
!   INFO = 2
! ELSE IF (INCX.EQ.0) THEN
!   INFO = 5
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZHPR ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. (ALPHA.EQ.DBLE(ZERO))) RETURN
!
! Set the start point in X if the increment is not unity.
!
! IF (INCX.LE.0) THEN
!   KX = 1 - (N-1)*INCX
! ELSE IF (INCX.NE.1) THEN
!   KX = 1
! END IF
!
! Start the operations. In this version the elements of the array AP
! are accessed sequentially with one pass through AP.
!
! KK = 1
! IF (LSAME(UPLO,'U')) THEN
!
!   Form A when upper triangle is stored in AP.
!
!   IF (INCX.EQ.1) THEN
!     DO 20 J = 1,N
!       IF (X(J).NE.ZERO) THEN
!         TEMP = ALPHA*DCONJG(X(J))

```

```

        K = KK
        DO 10 I = 1, J - 1
            AP(K) = AP(K) + X(I)*TEMP
            K = K + 1
10      CONTINUE
        AP(KK+J-1) = DBLE(AP(KK+J-1)) + DBLE(X(J)*TEMP)
        ELSE
            AP(KK+J-1) = DBLE(AP(KK+J-1))
        END IF
        KK = KK + J
20      CONTINUE
    ELSE
        JX = KX
        DO 40 J = 1, N
            IF (X(JX).NE.ZERO) THEN
                TEMP = ALPHA*DCONJG(X(JX))
                IX = KX
                DO 30 K = KK, KK + J - 2
                    AP(K) = AP(K) + X(IX)*TEMP
                    IX = IX + INCX
30              CONTINUE
                AP(KK+J-1) = DBLE(AP(KK+J-1)) + DBLE(X(JX)*TEMP)
            ELSE
                AP(KK+J-1) = DBLE(AP(KK+J-1))
            END IF
            JX = JX + INCX
            KK = KK + J
40          CONTINUE
        END IF
    ELSE
        !
        !      Form A when lower triangle is stored in AP.
        !
        IF (INCX.EQ.1) THEN
            DO 60 J = 1, N
                IF (X(J).NE.ZERO) THEN
                    TEMP = ALPHA*DCONJG(X(J))
                    AP(KK) = DBLE(AP(KK)) + DBLE(TEMP*X(J))
                    K = KK + 1
                    DO 50 I = J + 1, N
                        AP(K) = AP(K) + X(I)*TEMP
                        K = K + 1
50              CONTINUE
                ELSE
                    AP(KK) = DBLE(AP(KK))
                END IF
                KK = KK + N - J + 1
60              CONTINUE
            ELSE
                JX = KX
                DO 80 J = 1, N
                    IF (X(JX).NE.ZERO) THEN
                        TEMP = ALPHA*DCONJG(X(JX))
                        AP(KK) = DBLE(AP(KK)) + DBLE(TEMP*X(JX))
                        IX = JX
                        DO 70 K = KK + 1, KK + N - J
                            IX = IX + INCX
                            AP(K) = AP(K) + X(IX)*TEMP
70                        CONTINUE
                        ELSE
                            AP(KK) = DBLE(AP(KK))
                        END IF
                        JX = JX + INCX
                        KK = KK + N - J + 1
80                      CONTINUE
                    END IF
                END IF
            RETURN
        !
        !      End of ZHPR .
        !
        END SUBROUTINE
        *> \brief \b ZHER2
        !
        ! ===== DOCUMENTATION =====
        !
        ! Online html documentation available at
        ! http://www.netlib.org/lapack/explore-html/
        !
        ! Definition:
        ! =====
        !
        ! SUBROUTINE ZHER2(UPLO,N,ALPHA,X,INCX,Y,INCY,A,LDA)
        !
        ! .. Scalar Arguments ..
        ! COMPLEX*16 ALPHA
        ! INTEGER INCX, INCY, LDA, N
        ! CHARACTER UPLO
        ! ..
        ! .. Array Arguments ..
        ! COMPLEX*16 A(LDA,*), X(*), Y(*)
        ! ..
        !
        !> \par Purpose:
        ! =====
        !>
        !> \verbatim
        !>
        !> ZHER2 performs the hermitian rank 2 operation
        !>
        !> A := alpha*x*y**H + conjg(alpha)*y*x**H + A,
        !>
        !> where alpha is a scalar, x and y are n element vectors and A is an n
        !> by n hermitian matrix.
        !> \endverbatim
        !
        ! Arguments:
        ! =====
        !> \param[in] UPLO
        !> \verbatim
        !> UPLO is CHARACTER*1

```

```

!>      On entry, UPLO specifies whether the upper or lower
!>      triangular part of the array A is to be referenced as
!>      follows:
!>
!>      UPLO = 'U' or 'u'   Only the upper triangular part of A
!>                          is to be referenced.
!>
!>      UPLO = 'L' or 'l'   Only the lower triangular part of A
!>                          is to be referenced.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>      N is INTEGER
!>      On entry, N specifies the order of the matrix A.
!>      N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is COMPLEX*16
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] X
!> \verbatim
!>      X is COMPLEX*16 array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCX ) ).
!>      Before entry, the incremented array X must contain the n
!>      element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>      INCX is INTEGER
!>      On entry, INCX specifies the increment for the elements of
!>      X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!>      Y is COMPLEX*16 array of dimension at least
!>      ( 1 + ( n - 1 ) * abs( INCY ) ).
!>      Before entry, the incremented array Y must contain the n
!>      element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>      INCY is INTEGER
!>      On entry, INCY specifies the increment for the elements of
!>      Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] A
!> \verbatim
!>      A is COMPLEX*16 array of DIMENSION ( LDA, n ).
!>      Before entry with  UPLO = 'U' or 'u', the leading n by n
!>      upper triangular part of the array A must contain the upper
!>      triangular part of the hermitian matrix and the strictly
!>      lower triangular part of A is not referenced. On exit, the
!>      upper triangular part of the array A is overwritten by the
!>      upper triangular part of the updated matrix.
!>      Before entry with UPLO = 'L' or 'l', the leading n by n
!>      lower triangular part of the array A must contain the lower
!>      triangular part of the hermitian matrix and the strictly
!>      upper triangular part of A is not referenced. On exit, the
!>      lower triangular part of the array A is overwritten by the
!>      lower triangular part of the updated matrix.
!>      Note that the imaginary parts of the diagonal elements need
!>      not be set, they are assumed to be zero, and on exit they
!>      are set to zero.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. LDA must be at least
!>      max( 1, n ).
!> \endverbatim
!>
!> !
!> ! Authors:
!> ! =====
!> !
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!> !
!> \date November 2011
!> !
!> \ingroup complex16_blas_level2
!> !
!> \par Further Details:
!> ! =====
!> !
!> \verbatim
!>      Level 2 Blas routine.
!>
!> -- Written on 22-October-1986.
!>    Jack Dongarra, Argonne National Lab.
!>    Jeremy Du Croz, Nag Central Office.
!>    Sven Hammarling, Nag Central Office.
!>    Richard Hanson, Sandia National Labs.
!> \endverbatim
!>
!> !
!> ! =====
!> SUBROUTINE ZHER2(UPLO,N,ALPHA,X,INCX,Y,INCY,A,LDA)
!> !
!> -- Reference BLAS level2 routine (version 3.4.0) --
!> -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!> -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---

```

```

! November 2011
!
! .. Scalar Arguments ..
COMPLEX*16 ALPHA
INTEGER INCX, INCY, LDA, N
CHARACTER UPLO
!
! .. Array Arguments ..
COMPLEX*16 A(LDA,*), X(*), Y(*)
!
!
=====
!
! .. Parameters ..
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! .. Local Scalars ..
COMPLEX*16 TEMP1, TEMP2
INTEGER I, INFO, IX, IY, J, JX, JY, KX, KY
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! .. External Subroutines ..
EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
INTRINSIC DBLE, DCONJG, MAX
!
! ..
!
Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO, 'U') .AND. .NOT.LSAME(UPLO, 'L')) THEN
    INFO = 1
ELSE IF (N.LT.0) THEN
    INFO = 2
ELSE IF (INCX.EQ.0) THEN
    INFO = 5
ELSE IF (INCY.EQ.0) THEN
    INFO = 7
ELSE IF (LDA.LT.MAX(1,N)) THEN
    INFO = 9
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZHER2 ', INFO)
    RETURN
END IF
!
! Quick return if possible.
!
IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN
!
! Set up the start points in X and Y if the increments are not both
! unity.
!
IF ((INCX.NE.1) .OR. (INCY.NE.1)) THEN
    IF (INCX.GT.0) THEN
        KX = 1
    ELSE
        KX = 1 - (N-1)*INCX
    END IF
    IF (INCY.GT.0) THEN
        KY = 1
    ELSE
        KY = 1 - (N-1)*INCY
    END IF
    JX = KX
    JY = KY
END IF
!
! Start the operations. In this version the elements of A are
! accessed sequentially with one pass through the triangular part
! of A.
!
IF (LSAME(UPLO, 'U')) THEN
!
! Form A when A is stored in the upper triangle.
!
    IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
        DO 20 J = 1, N
            IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
                TEMP1 = ALPHA*DCONJG(Y(J))
                TEMP2 = DCONJG(ALPHA*X(J))
                DO 10 I = 1, J - 1
                    A(I,J) = A(I,J) + X(I)*TEMP1 + Y(I)*TEMP2
10                CONTINUE
                A(J,J) = DBLE(A(J,J)) +&
                    DBLE(X(J)*TEMP1+Y(J)*TEMP2)
            ELSE
                A(J,J) = DBLE(A(J,J))
            END IF
20        CONTINUE
    ELSE
        DO 40 J = 1, N
            IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
                TEMP1 = ALPHA*DCONJG(Y(JY))
                TEMP2 = DCONJG(ALPHA*X(JX))
                IX = KX
                IY = KY
                DO 30 I = 1, J - 1
                    A(I,J) = A(I,J) + X(IX)*TEMP1 + Y(IY)*TEMP2
30                CONTINUE
                A(J,J) = DBLE(A(J,J)) +&
                    DBLE(X(JX)*TEMP1+Y(JY)*TEMP2)
            ELSE
                A(J,J) = DBLE(A(J,J))
            END IF
            JX = JX + INCX
            JY = JY + INCY
        END IF
    END IF

```

```

40      CONTINUE
      END IF
      ELSE
!
!       Form A when A is stored in the lower triangle.
!
      IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
        DO 60 J = 1,N
          IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
            TEMP1 = ALPHA*DCONJG(Y(J))
            TEMP2 = DCONJG(ALPHA*X(J))
            A(J,J) = DBLE(A(J,J)) +%
              DBLE(X(J)*TEMP1+Y(J)*TEMP2)
            DO 50 I = J + 1,N
              A(I,J) = A(I,J) + X(I)*TEMP1 + Y(I)*TEMP2
50          CONTINUE
            ELSE
              A(J,J) = DBLE(A(J,J))
            END IF
          END IF
60        CONTINUE
      ELSE
        DO 80 J = 1,N
          IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
            TEMP1 = ALPHA*DCONJG(Y(JY))
            TEMP2 = DCONJG(ALPHA*X(JX))
            A(J,J) = DBLE(A(J,J)) +%
              DBLE(X(JX)*TEMP1+Y(JY)*TEMP2)
            IX = JX
            IY = JY
            DO 70 I = J + 1,N
              IX = IX + INCX
              IY = IY + INCY
              A(I,I) = A(I,I) + X(IX)*TEMP1 + Y(IY)*TEMP2
70          CONTINUE
            ELSE
              A(J,J) = DBLE(A(J,J))
            END IF
            JX = JX + INCX
            JY = JY + INCY
          END IF
80        CONTINUE
      END IF
      END IF
!
      RETURN
!
!       End of ZHER2 .
!
      END SUBROUTINE
!
!       *> \brief \b ZHPR2
!
!       ===== DOCUMENTATION =====
!
!       Online html documentation available at
!       http://www.netlib.org/lapack/explore-html/
!
!       Definition:
!       =====
!
!       SUBROUTINE ZHPR2(UPLO,N,ALPHA,X,INCX,Y,INCY,AP)
!
!       .. Scalar Arguments ..
!       COMPLEX*16 ALPHA
!       INTEGER INCX,INCY,N
!       CHARACTER UPLO
!       ..
!       .. Array Arguments ..
!       COMPLEX*16 AP(*),X(*),Y(*)
!       ..
!
!
!> \par Purpose:
!> =====
!> \verbatim
!> ZHPR2 performs the hermitian rank 2 operation
!>
!>   A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
!>
!> where alpha is a scalar, x and y are n element vectors and A is an
!> n by n hermitian matrix, supplied in packed form.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the matrix A is supplied in the packed
!> array AP as follows:
!>
!>   UPLO = 'U' or 'u' The upper triangular part of A is
!>   supplied in AP.
!>
!>   UPLO = 'L' or 'l' The lower triangular part of A is
!>   supplied in AP.
!> \endverbatim
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix A.
!> N must be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is COMPLEX*16
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] X

```

```

!> \verbatim
!>     X is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCX ) ).
!>     Before entry, the incremented array X must contain the n
!>     element vector x.
!> \endverbatim
!>
!> \param[in] INCX
!> \verbatim
!>     INCX is INTEGER
!>     On entry, INCX specifies the increment for the elements of
!>     X. INCX must not be zero.
!> \endverbatim
!>
!> \param[in] Y
!> \verbatim
!>     Y is COMPLEX*16 array of dimension at least
!>     ( 1 + ( n - 1 ) * abs( INCY ) ).
!>     Before entry, the incremented array Y must contain the n
!>     element vector y.
!> \endverbatim
!>
!> \param[in] INCY
!> \verbatim
!>     INCY is INTEGER
!>     On entry, INCY specifies the increment for the elements of
!>     Y. INCY must not be zero.
!> \endverbatim
!>
!> \param[in,out] AP
!> \verbatim
!>     AP is COMPLEX*16 array of DIMENSION at least
!>     ( ( n*( n + 1 ) ) / 2 ).
!>     Before entry with  UPLO = 'U' or 'u', the array AP must
!>     contain the upper triangular part of the hermitian matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 1, 2 )
!>     and a( 2, 2 ) respectively, and so on. On exit, the array
!>     AP is overwritten by the upper triangular part of the
!>     updated matrix.
!>     Before entry with  UPLO = 'L' or 'l', the array AP must
!>     contain the lower triangular part of the hermitian matrix
!>     packed sequentially, column by column, so that AP( 1 )
!>     contains a( 1, 1 ), AP( 2 ) and AP( 3 ) contain a( 2, 1 )
!>     and a( 3, 1 ) respectively, and so on. On exit, the array
!>     AP is overwritten by the lower triangular part of the
!>     updated matrix.
!>     Note that the imaginary parts of the diagonal elements need
!>     not be set, they are assumed to be zero, and on exit they
!>     are set to zero.
!> \endverbatim
!
!   Authors:
!   =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level2
!
!> \par Further Details:
!   =====
!> \verbatim
!>
!>     Level 2 Blas routine.
!>
!>     -- Written on 22-October-1986.
!>     Jack Dongarra, Argonne National Lab.
!>     Jeremy Du Croz, Nag Central Office.
!>     Sven Hammarling, Nag Central Office.
!>     Richard Hanson, Sandia National Labs.
!> \endverbatim
!
!   =====
!   SUBROUTINE ZHPR2(UPLO,N,ALPHA,X,INCX,Y,INCY,AP)
!
!   -- Reference BLAS level2 routine (version 3.4.0) --
!   -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!   -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
!   .. Scalar Arguments ..
!   COMPLEX*16 ALPHA
!   INTEGER INCX, INCY, N
!   CHARACTER UPLO
!
!   .. Array Arguments ..
!   COMPLEX*16 AP(*), X(*), Y(*)
!
!   ..
!
!   =====
!
!   .. Parameters ..
!   COMPLEX*16 ZERO
!   PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
!   .. Local Scalars ..
!   COMPLEX*16 TEMP1, TEMP2
!   INTEGER I, INFO, IX, IY, J, JX, JY, K, KK, KX, KY
!
!   .. External Functions ..
!   LOGICAL LSAME
!   EXTERNAL LSAME
!
!   ..
!   .. External Subroutines ..
!   EXTERNAL XERBLA
!
!   ..
!   .. Intrinsic Functions ..
!   INTRINSIC DBLE, DCONJG

```

```

! ..
!
! Test the input parameters.
!
INFO = 0
IF (.NOT.LSAME(UPLO,'U') .AND. .NOT.LSAME(UPLO,'L')) THEN
  INFO = 1
ELSE IF (N.LT.0) THEN
  INFO = 2
ELSE IF (INCX.EQ.0) THEN
  INFO = 5
ELSE IF (INCY.EQ.0) THEN
  INFO = 7
END IF
IF (INFO.NE.0) THEN
  CALL XERBLA('ZHPR2 ',INFO)
  RETURN
END IF

! Quick return if possible.
!
IF ((N.EQ.0) .OR. (ALPHA.EQ.ZERO)) RETURN

! Set up the start points in X and Y if the increments are not both
! unity.
!
IF ((INCX.NE.1) .OR. (INCY.NE.1)) THEN
  IF (INCX.GT.0) THEN
    KX = 1
  ELSE
    KX = 1 - (N-1)*INCX
  END IF
  IF (INCY.GT.0) THEN
    KY = 1
  ELSE
    KY = 1 - (N-1)*INCY
  END IF
  JX = KX
  JY = KY
END IF

! Start the operations. In this version the elements of the array AP
! are accessed sequentially with one pass through AP.
!
KK = 1
IF (LSAME(UPLO,'U')) THEN
  Form A when upper triangle is stored in AP.
  IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
    DO 20 J = 1,N
      IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
        TEMP1 = ALPHA*DCONJG(Y(J))
        TEMP2 = DCONJG(ALPHA*X(J))
        K = KK
        DO 10 I = 1, J - 1
          AP(K) = AP(K) + X(I)*TEMP1 + Y(I)*TEMP2
          K = K + 1
10        CONTINUE
        AP(KK+J-1) = DBLE(AP(KK+J-1)) +&
          DBLE(X(J)*TEMP1+Y(J)*TEMP2)
        ELSE
          AP(KK+J-1) = DBLE(AP(KK+J-1))
        END IF
        KK = KK + J
20      CONTINUE
    ELSE
      DO 40 J = 1,N
        IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
          TEMP1 = ALPHA*DCONJG(Y(JY))
          TEMP2 = DCONJG(ALPHA*X(JX))
          IX = KX
          IY = KY
          DO 30 K = KK, KK + J - 2
            AP(K) = AP(K) + X(IX)*TEMP1 + Y(IY)*TEMP2
            IX = IX + INCX
            IY = IY + INCY
30          CONTINUE
          AP(KK+J-1) = DBLE(AP(KK+J-1)) +&
            DBLE(X(JX)*TEMP1+Y(JY)*TEMP2)
          ELSE
            AP(KK+J-1) = DBLE(AP(KK+J-1))
          END IF
          JX = JX + INCX
          JY = JY + INCY
          KK = KK + J
40        CONTINUE
      END IF
    ELSE
      Form A when lower triangle is stored in AP.
      IF ((INCX.EQ.1) .AND. (INCY.EQ.1)) THEN
        DO 60 J = 1,N
          IF ((X(J).NE.ZERO) .OR. (Y(J).NE.ZERO)) THEN
            TEMP1 = ALPHA*DCONJG(Y(J))
            TEMP2 = DCONJG(ALPHA*X(J))
            AP(KK) = DBLE(AP(KK)) +&
              DBLE(X(J)*TEMP1+Y(J)*TEMP2)
            K = KK + 1
            DO 50 I = J + 1, N
              AP(K) = AP(K) + X(I)*TEMP1 + Y(I)*TEMP2
              K = K + 1
50            CONTINUE
            ELSE
              AP(KK) = DBLE(AP(KK))
            END IF
            KK = KK + N - J + 1
60          CONTINUE
        ELSE
          DO 80 J = 1,N
            IF ((X(JX).NE.ZERO) .OR. (Y(JY).NE.ZERO)) THEN
              TEMP1 = ALPHA*DCONJG(Y(JY))
              TEMP2 = DCONJG(ALPHA*X(JX))

```

```

        AP(KK) = DBLE(AP(KK)) +&
            DBLE(X(JX)*TEMP1+Y(JY)*TEMP2)
        IX = JX
        IY = JY
        DO 70 K = KK + 1, KK + N - J
            IX = IX + INCX
            IY = IY + INCY
            AP(K) = AP(K) + X(IX)*TEMP1 + Y(IY)*TEMP2
70      CONTINUE
        ELSE
            AP(KK) = DBLE(AP(KK))
        END IF
        JX = JX + INCX
        JY = JY + INCY
        KK = KK + N - J + 1
80      CONTINUE
    END IF
END IF
RETURN
!
!   End of ZHPR2 .
!
END SUBROUTINE

end program

```

APPENDIX A.7 – ZBLAT3.f90

```

program testzblat3
implicit none

! Variables
CHARACTER :: kin

CALL ZBLAT3
print *, 'ZBLAT3 Done.'

CONTAINS
!> \brief \b LSAME
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! LOGICAL FUNCTION LSAME(CA,CB)
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> LSAME returns .TRUE. if CA is the same letter as CB regardless of
!> case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] CA
!> \verbatim
!> CA is CHARACTER*1
!> \endverbatim
!> \param[in] CB
!> \verbatim
!> CB is CHARACTER*1
!> CA and CB specify the single characters to be compared.
!> \endverbatim
!
! Authors:
! =====
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup aux_blas
!
! =====
! LOGICAL FUNCTION LSAME(CA,CB)
!
! -- Reference BLAS level1 routine (version 3.1) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! CHARACTER CA,CB
! ..
! =====
!
! .. Intrinsic Functions ..
! INTRINSIC ICHAR
! ..
! .. Local Scalars ..
! INTEGER INTA,INTB,ZCODE
! ..
!
! Test if the characters are equal
!
! LSAME = CA .EQ. CB
! IF (LSAME) RETURN
!
! Now test for equivalence if both characters are alphabetic.
!
! ZCODE = ICHAR('Z')
!
! Use 'Z' rather than 'A' so that ASCII can be detected on Prime
! machines, on which ICHAR returns a value with bit 8 set.
! ICHAR('A') on Prime machines returns 193 which is the same as
! ICHAR('A') on an EBCDIC machine.
!
! INTA = ICHAR(CA)
! INTB = ICHAR(CB)
!
! IF (ZCODE.EQ.90 .OR. ZCODE.EQ.122) THEN
!
! ASCII is assumed - ZCODE is the ASCII code of either lower or
! upper case 'Z'.
!
! IF (INTA.GE.97 .AND. INTA.LE.122) INTA = INTA - 32
! IF (INTB.GE.97 .AND. INTB.LE.122) INTB = INTB - 32
!
! ELSE IF (ZCODE.EQ.233 .OR. ZCODE.EQ.169) THEN
!
! EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or

```

```

!         upper case 'Z'.
!
!         IF (INTA.GE.129 .AND. INTA.LE.137 .OR.&
!             INTA.GE.145 .AND. INTA.LE.153 .OR.&
!             INTA.GE.162 .AND. INTA.LE.169) INTA = INTA + 64
!         IF (INTB.GE.129 .AND. INTB.LE.137 .OR.&
!             INTB.GE.145 .AND. INTB.LE.153 .OR.&
!             INTB.GE.162 .AND. INTB.LE.169) INTB = INTB + 64
!
!         ELSE IF (ZCODE.EQ.218 .OR. ZCODE.EQ.250) THEN
!
!             ASCII is assumed, on Prime machines - ZCODE is the ASCII code
!             plus 128 of either lower or upper case 'Z'.
!
!             IF (INTA.GE.225 .AND. INTA.LE.250) INTA = INTA - 32
!             IF (INTB.GE.225 .AND. INTB.LE.250) INTB = INTB - 32
!         END IF
!         LSAME = INTA .EQ. INTB
!
!         RETURN
!
!         End of LSAME
!
!     END FUNCTION
!> \brief \b ZBLAT3
!
!     ===== DOCUMENTATION =====
!
!     Online html documentation available at
!     http://www.netlib.org/lapack/explore-html/
!
!     Definition:
!     =====
!
!     PROGRAM ZBLAT3
!
!> \par Purpose:
!     =====
!>
!> \verbatim
!>
!> Test program for the COMPLEX*16          Level 3 Blas.
!>
!> The program must be driven by a short data file. The first 14 records
!> of the file are read using list-directed input, the last 9 records
!> are read using the format ( A6, L2 ). An annotated example of a data
!> file can be obtained by deleting the first 3 characters from the
!> following 23 lines:
!> 'zblat3.out'      NAME OF SUMMARY OUTPUT FILE
!> 6                 UNIT NUMBER OF SUMMARY FILE
!> 'ZBLAT3.SNAP'     NAME OF SNAPSHOT OUTPUT FILE
!> -1                UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
!> F                 LOGICAL FLAG, T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
!> F                 LOGICAL FLAG, T TO STOP ON FAILURES.
!> T                 LOGICAL FLAG, T TO TEST ERROR EXITS.
!> 16.0              THRESHOLD VALUE OF TEST RATIO
!> 6                 NUMBER OF VALUES OF N
!> 0 1 2 3 5 9       VALUES OF N
!> 3                 NUMBER OF VALUES OF ALPHA
!> (0.0,0.0) (1.0,0.0) (0.7,-0.9)  VALUES OF ALPHA
!> 3                 NUMBER OF VALUES OF BETA
!> (0.0,0.0) (1.0,0.0) (1.3,-1.1)  VALUES OF BETA
!> ZGEMM T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHEMM T PUT F FOR NO TEST. SAME COLUMNS.
!> ZSYMM T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTRMM T PUT F FOR NO TEST. SAME COLUMNS.
!> ZTRSM T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHERK T PUT F FOR NO TEST. SAME COLUMNS.
!> ZSYRK T PUT F FOR NO TEST. SAME COLUMNS.
!> ZHER2K T PUT F FOR NO TEST. SAME COLUMNS.
!> ZSYR2K T PUT F FOR NO TEST. SAME COLUMNS.
!>
!>
!> Further Details
!> =====
!>
!> See:
!>
!> Dongarra J. J., Du Croz J. J., Duff I. S. and Hammarling S.
!> A Set of Level 3 Basic Linear Algebra Subprograms.
!>
!> Technical Memorandum No.88 (Revision 1), Mathematics and
!> Computer Science Division, Argonne National Laboratory, 9700
!> South Cass Avenue, Argonne, Illinois 60439, US.
!>
!> -- Written on 8-February-1989.
!> Jack Dongarra, Argonne National Laboratory.
!> Iain Duff, AERE Harwell.
!> Jeremy Du Croz, Numerical Algorithms Group Ltd.
!> Sven Hammarling, Numerical Algorithms Group Ltd.
!>
!> 10-9-00:  Change STATUS='NEW' to 'UNKNOWN' so that the testers
!>           can be run multiple times without deleting generated
!>           output files (susan)
!> \endverbatim
!
!     Authors:
!     =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date April 2012
!
!> \ingroup complex16_blas_testing
!
!     =====
!
!     SUBROUTINE ZBLAT3
!
!     -- Reference BLAS test routine (version 3.4.1) --
!     -- Reference BLAS is a software package provided by Univ. of Tennessee, --

```

```

! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! April 2012
!
! =====
!
! .. Parameters ..
INTEGER      NIN
PARAMETER    ( NIN = 5 )
INTEGER      NSUBS
PARAMETER    ( NSUBS = 9 )
COMPLEX*16   ZERO, ONE
PARAMETER    ( ZERO = ( 0.0D0, 0.0D0 ), &
               ONE = ( 1.0D0, 0.0D0 ) )
DOUBLE PRECISION RZERO
PARAMETER    ( RZERO = 0.0D0 )
INTEGER      NMAX
PARAMETER    ( NMAX = 65 )
INTEGER      NIDMAX, NALMAX, NBEMAX
PARAMETER    ( NIDMAX = 9, NALMAX = 7, NBEMAX = 7 )
!
! .. Local Scalars ..
DOUBLE PRECISION EPS, ERR, THRESH
INTEGER      I, ISNUM, J, N, NALF, NBET, NIDIM, NOUT, NTRA
LOGICAL      FATAL, LTESTT, REWI, SAME, SPATAL, TRACE, &
              TSERR
CHARACTER*1  TRANSA, TRANSB
CHARACTER*6  SNAMET
CHARACTER*32 SNAPS, SUMMRY
!
! .. Local Arrays ..
COMPLEX*16   AA( NMAX*NMAX ), AB( NMAX, 2*NMAX ), &
              ALF( NALMAX ), AS( NMAX*NMAX ), &
              BB( NMAX*NMAX ), BET( NBEMAX ), &
              BS( NMAX*NMAX ), C( NMAX, NMAX ), &
              CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ), &
              W( 2*NMAX )
DOUBLE PRECISION G( NMAX )
INTEGER      IDIM( NIDMAX )
LOGICAL      LTEST( NSUBS )
CHARACTER*6  SNAME( NSUBS )
!
! .. External Functions ..
DOUBLE PRECISION DDIFF
LOGICAL      LZE
EXTERNAL      DDIFF, LZE
!
! .. External Subroutines ..
EXTERNAL      ZCHK1, ZCHK2, ZCHK3, ZCHK4, ZCHK5, ZCHKE, ZMMCH
!
! .. Intrinsic Functions ..
INTRINSIC     MAX, MIN
!
! .. Scalars in Common ..
INTEGER      INFOT, NOUTC
LOGICAL      LERR, OK
CHARACTER*6  SRNAMT
!
! .. Common blocks ..
COMMON       /INFOC/INFOT, NOUTC, OK, LERR
COMMON       /SRNAMC/SRNAMT
!
! .. Data statements ..
DATA         SNAME/'ZGEMM ', 'ZHEMM ', 'ZSYMM ', 'ZTRMM ', &
              'ZTRSM ', 'ZHERK ', 'ZSYRK ', 'ZHER2K ', &
              'ZSYR2K'/
!
! .. Executable Statements ..
!
! Read name and unit number for summary output file and open file.
!
READ( NIN, FMT = * )SUMMRY
READ( NIN, FMT = * )NOUT
OPEN( NOUT, FILE = SUMMRY, STATUS = 'UNKNOWN' )
NOUTC = NOUT
!
! Read name and unit number for snapshot output file and open file.
!
READ( NIN, FMT = * )SNAPS
READ( NIN, FMT = * )NTRA
TRACE = NTRA.GE.0
IF( TRACE )THEN
    OPEN( NTRA, FILE = SNAPS, STATUS = 'UNKNOWN' )
END IF
!
! Read the flag that directs rewinding of the snapshot file.
READ( NIN, FMT = * )REWI
REWI = REWI.AND.TRACE
!
! Read the flag that directs stopping on any failure.
READ( NIN, FMT = * )SPATAL
!
! Read the flag that indicates whether error exits are to be tested.
READ( NIN, FMT = * )TSERR
!
! Read the threshold value of the test ratio
READ( NIN, FMT = * )THRESH
!
! Read and check the parameter values for the tests.
!
! Values of N
READ( NIN, FMT = * )NIDIM
IF( NIDIM.LT.1.OR.NIDIM.GT.NIDMAX )THEN
    WRITE( NOUT, FMT = 9997 )'N', NIDMAX
    GO TO 220
END IF
READ( NIN, FMT = * ) ( IDIM( I ), I = 1, NIDIM )
DO 10 I = 1, NIDIM
    IF( IDIM( I ).LT.0.OR.IDIM( I ).GT.NMAX )THEN
        WRITE( NOUT, FMT = 9996 )NMAX
        GO TO 220
    END IF
10 CONTINUE
!
! Values of ALPHA
READ( NIN, FMT = * )NALF
IF( NALF.LT.1.OR.NALF.GT.NALMAX )THEN
    WRITE( NOUT, FMT = 9997 )'ALPHA', NALMAX
    GO TO 220
END IF
READ( NIN, FMT = * ) ( ALF( I ), I = 1, NALF )
!
! Values of BETA
READ( NIN, FMT = * )NBET
IF( NBET.LT.1.OR.NBET.GT.NBEMAX )THEN
    WRITE( NOUT, FMT = 9997 )'BETA', NBEMAX
    GO TO 220
END IF
READ( NIN, FMT = * ) ( BET( I ), I = 1, NBET )
!

```

```

! Report values of parameters.
!
WRITE( NOUT, FMT = 9995 )
WRITE( NOUT, FMT = 9994 ) ( IDIM( I ), I = 1, NIDIM )
WRITE( NOUT, FMT = 9993 ) ( ALF( I ), I = 1, NALF )
WRITE( NOUT, FMT = 9992 ) ( BET( I ), I = 1, NBET )
IF( .NOT.TSTERR ) THEN
  WRITE( NOUT, FMT = * )
  WRITE( NOUT, FMT = 9984 )
END IF
WRITE( NOUT, FMT = * )
WRITE( NOUT, FMT = 9999 ) THRESH
WRITE( NOUT, FMT = * )
!
! Read names of subroutines and flags which indicate
! whether they are to be tested.
!
DO 20 I = 1, NSUBS
  LTEST( I ) = .FALSE.
20 CONTINUE
30 READ( NIN, FMT = 9988, END = 60 ) SNAMET, LTESTT
DO 40 I = 1, NSUBS
  IF( SNAMET.EQ.SNAMES( I ) ) &
    GO TO 50
40 CONTINUE
WRITE( NOUT, FMT = 9990 ) SNAMET
STOP
50 LTEST( I ) = LTESTT
GO TO 30
!
60 CONTINUE
CLOSE ( NIN )
!
! Compute EPS (the machine precision).
!
EPS = EPSILON(RZERO)
WRITE( NOUT, FMT = 9998 ) EPS
!
! Check the reliability of ZMMCH using exact data.
!
N = MIN( 32, NMAX )
DO 100 J = 1, N
  DO 90 I = 1, N
    AB( I, J ) = MAX( I - J + 1, 0 )
90 CONTINUE
AB( J, NMAX + 1 ) = J
AB( 1, NMAX + J ) = J
C( J, 1 ) = ZERO
100 CONTINUE
DO 110 J = 1, N
  CC( J ) = J*( ( J + 1 ) * J ) / 2 - ( ( J + 1 ) * J * ( J - 1 ) ) / 3
110 CONTINUE
! CC holds the exact result. On exit from ZMMCH CT holds
! the result computed by ZMMCH.
TRANSA = 'N'
TRANSB = 'N'
CALL ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
  AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
  NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
SAME = LZE( CC, CT, N )
IF( .NOT.SAME.OR.ERR.NE.RZERO ) THEN
  WRITE( NOUT, FMT = 9989 ) TRANSA, TRANSB, SAME, ERR
  STOP
END IF
TRANSB = 'C'
CALL ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
  AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
  NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
SAME = LZE( CC, CT, N )
IF( .NOT.SAME.OR.ERR.NE.RZERO ) THEN
  WRITE( NOUT, FMT = 9989 ) TRANSA, TRANSB, SAME, ERR
  STOP
END IF
DO 120 J = 1, N
  AB( J, NMAX + 1 ) = N - J + 1
  AB( 1, NMAX + J ) = N - J + 1
120 CONTINUE
DO 130 J = 1, N
  CC( N - J + 1 ) = J*( ( J + 1 ) * J ) / 2 - &
    ( ( J + 1 ) * J * ( J - 1 ) ) / 3
130 CONTINUE
TRANSA = 'C'
TRANSB = 'N'
CALL ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
  AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
  NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
SAME = LZE( CC, CT, N )
IF( .NOT.SAME.OR.ERR.NE.RZERO ) THEN
  WRITE( NOUT, FMT = 9989 ) TRANSA, TRANSB, SAME, ERR
  STOP
END IF
TRANSB = 'C'
CALL ZMMCH( TRANSA, TRANSB, N, 1, N, ONE, AB, NMAX, &
  AB( 1, NMAX + 1 ), NMAX, ZERO, C, NMAX, CT, G, CC, &
  NMAX, EPS, ERR, FATAL, NOUT, .TRUE. )
SAME = LZE( CC, CT, N )
IF( .NOT.SAME.OR.ERR.NE.RZERO ) THEN
  WRITE( NOUT, FMT = 9989 ) TRANSA, TRANSB, SAME, ERR
  STOP
END IF
!
! Test each subroutine in turn.
!
DO 200 ISNUM = 1, NSUBS
  WRITE( NOUT, FMT = * )
  IF( .NOT.LTEST( ISNUM ) ) THEN
    Subprogram is not to be tested.
    WRITE( NOUT, FMT = 9987 ) SNAMES( ISNUM )
  ELSE
    SRNAMT = SNAMES( ISNUM )
    Test error exits.
    IF( TSTERR ) THEN
      CALL ZCHKE( ISNUM, SNAMES( ISNUM ), NOUT )
      WRITE( NOUT, FMT = * )
    
```

```

        END IF
        Test computations.
        INFOT = 0
        OK = .TRUE.
        FATAL = .FALSE.
        GO TO ( 140, 150, 160, 170, 180, 190, 200, 210, 220, 230 ) ISNUM
!
140      Test ZGEMM, 01
        CALL ZCHK1( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
         & REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, &
         & NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C, &
         & CC, CS, CT, G )

        GO TO 190
!
150      Test ZHEMM, 02, ZSYMM, 03.
        CALL ZCHK2( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
         & REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, &
         & NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C, &
         & CC, CS, CT, G )

        GO TO 190
!
160      Test ZTRMM, 04, ZTRSM, 05.
        CALL ZCHK3( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
         & REWI, FATAL, NIDIM, IDIM, NALF, ALF, NMAX, AB, &
         & AA, AS, AB( 1, NMAX + 1 ), BB, BS, CT, G, C )

        GO TO 190
!
170      Test ZHERK, 06, ZSYRK, 07.
        CALL ZCHK4( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
         & REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, &
         & NMAX, AB, AA, AS, AB( 1, NMAX + 1 ), BB, BS, C, &
         & CC, CS, CT, G )

        GO TO 190
!
180      Test ZHER2K, 08, ZSYR2K, 09.
        CALL ZCHK5( SNAME( ISNUM ), EPS, THRESH, NOUT, NTRA, TRACE, &
         & REWI, FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, &
         & NMAX, AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )

        GO TO 190
!
190      IF( FATAL.AND.SFATAL ) &
         & GO TO 210
        END IF
200      CONTINUE
        WRITE( NOUT, FMT = 9986 )
        GO TO 230
!
210      CONTINUE
        WRITE( NOUT, FMT = 9985 )
        GO TO 230
!
220      CONTINUE
        WRITE( NOUT, FMT = 9991 )
!
230      CONTINUE
        IF( TRACE ) &
         & CLOSE ( NTRA )
        CLOSE ( NOUT )
!
9999 FORMAT( ' ROUTINES PASS COMPUTATIONAL TESTS IF TEST RATIO IS LES', &
         & 'S THAN', F8.2 )
9998 FORMAT( ' RELATIVE MACHINE PRECISION IS TAKEN TO BE', 1P, D9.1 )
9997 FORMAT( ' NUMBER OF VALUES OF ', A, ' IS LESS THAN 1 OR GREATER ', &
         & 'THAN ', I2 )
9996 FORMAT( ' VALUE OF N IS LESS THAN 0 OR GREATER THAN ', I2 )
9995 FORMAT( ' TESTS OF THE COMPLEX*16 LEVEL 3 BLAS', /' THE F', &
         & 'OLLOWING PARAMETER VALUES WILL BE USED:' )
9994 FORMAT( ' FOR N ', 9I6 )
9993 FORMAT( ' FOR ALPHA ', &
         & 7( '(', F4.1, ',', F4.1, ')', &
         & ' ', : ) )
9992 FORMAT( ' FOR BETA ', &
         & 7( '(', F4.1, ',', F4.1, ')', &
         & ' ', : ) )
9991 FORMAT( ' AMEND DATA FILE OR INCREASE ARRAY SIZES IN PROGRAM', &
         & /' ***** TESTS ABANDONED *****' )
9990 FORMAT( ' SUBPROGRAM NAME ', A6, ' NOT RECOGNIZED', /' ***** T', &
         & 'ESTS ABANDONED *****' )
9989 FORMAT( ' ERROR IN ZMMCH - IN-LINE DOT PRODUCTS ARE BEING EVALU', &
         & 'ATED WRONGLY.', /' ZMMCH WAS CALLED WITH TRANSA = ', A1, &
         & ' AND TRANSB = ', A1, /' AND RETURNED SAME = ', L1, ' AND ', &
         & 'ERR = ', F12.3, '.', /' THIS MAY BE DUE TO FAULTS IN THE ', &
         & 'ARITHMETIC OR THE COMPILER.', /' ***** TESTS ABANDONED ', &
         & '*****' )
9988 FORMAT( A6, L2 )
9987 FORMAT( 'X', A6, ' WAS NOT TESTED' )
9986 FORMAT( /' END OF TESTS' )
9985 FORMAT( /' ***** FATAL ERROR - TESTS ABANDONED *****' )
9984 FORMAT( ' ERROR-EXITS WILL NOT BE TESTED' )
!
!      End of ZBLAT3.
!
      END SUBROUTINE
      SUBROUTINE ZCHK1( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI, &
         & FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX, &
         & A, AA, AS, B, BB, BS, C, CC, CS, CT, G )
!
!      Tests ZGEMM.
!
!      Auxiliary routine for test program for Level 3 Blas.
!
!      -- Written on 8-February-1989.
!      Jack Dongarra, Argonne National Laboratory.
!      Iain Duff, AERE Harwell.
!      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!      Sven Hammarling, Numerical Algorithms Group Ltd.
!
!      .. Parameters ..
      COMPLEX*16      ZERO
      PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ) )
      DOUBLE PRECISION      RZERO
      PARAMETER      ( RZERO = 0.0D0 )
!
!      .. Scalar Arguments ..
      DOUBLE PRECISION      EPS, THRESH
      INTEGER              NALF, NBET, NIDIM, NMAX, NOUT, NTRA
      LOGICAL              FATAL, REWI, TRACE
      CHARACTER*6          SNAME
!
!      .. Array Arguments ..
      COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ), &
         & AS( NMAX*NMAX ), B( NMAX, NMAX ), &

```

```

        BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),&
        C( NMAX, NMAX ), CC( NMAX*NMAX ),&
        CS( NMAX*NMAX ), CT( NMAX )
DOUBLE PRECISION      G( NMAX )
INTEGER               IDIM( NIDIM )
!
! .. Local Scalars ..
COMPLEX*16            ALPHA, ALS, BETA, BLS
DOUBLE PRECISION      ERR, ERRMAX
INTEGER               I, IA, IB, ICA, ICB, IK, IM, IN, K, KS, LAA,&
LBB, LCC, LDA, LDAS, LDB, LDBS, LDC, LDCS, M,&
MA, MB, MS, N, NA, NARGS, NB, NC, NS
LOGICAL               NULL, RESET, SAME, TRANA, TRANB
CHARACTER*1           TRANAS, TRANBS, TRANSA, TRANSB
CHARACTER*3           ICH
!
! .. Local Arrays ..
LOGICAL               ISAME( 13 )
!
! .. External Functions ..
LOGICAL               LZE, LZERES
!
! .. External Subroutines ..
EXTERNAL              ZGEMM, ZMAKE, ZMMCH
!
! .. Intrinsic Functions ..
INTRINSIC             MAX
!
! .. Scalars in Common ..
INTEGER               INFOT, NOUTC
LOGICAL               LERR, OK
!
! .. Common blocks ..
COMMON               /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA                 ICH/'NTC'/
!
! .. Executable Statements ..
!
NARGS = 13
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
DO 110 IM = 1, NIDIM
    M = IDIM( IM )
!
    DO 100 IN = 1, NIDIM
        N = IDIM( IN )
        Set LDC to 1 more than minimum value if room.
        LDC = M
        IF( LDC.LT.NMAX )&
            LDC = LDC + 1
        Skip tests if not enough room.
        IF( LDC.GT.NMAX )&
            GO TO 100
        LCC = LDC*N
        NULL = N.LE.0.OR.M.LE.0
!
        DO 90 IK = 1, NIDIM
            K = IDIM( IK )
!
            DO 80 ICA = 1, 3
                TRANSA = ICH( ICA: ICA )
                TRANA = TRANSA.EQ.'T'.OR.TRANSA.EQ.'C'
!
                IF( TRANA )THEN
                    MA = K
                    NA = M
                ELSE
                    MA = M
                    NA = K
                END IF
                Set LDA to 1 more than minimum value if room.
                LDA = MA
                IF( LDA.LT.NMAX )&
                    LDA = LDA + 1
                Skip tests if not enough room.
                IF( LDA.GT.NMAX )&
                    GO TO 80
                LAA = LDA*NA
!
                Generate the matrix A.
!
                CALL ZMAKE( 'GE', ' ', ' ', MA, NA, A, NMAX, AA, LDA,&
                    RESET, ZERO )
!
                DO 70 ICB = 1, 3
                    TRANSB = ICH( ICB: ICB )
                    TRANB = TRANSB.EQ.'T'.OR.TRANSB.EQ.'C'
!
                    IF( TRANB )THEN
                        MB = N
                        NB = K
                    ELSE
                        MB = K
                        NB = N
                    END IF
                    Set LDB to 1 more than minimum value if room.
                    LDB = MB
                    IF( LDB.LT.NMAX )&
                        LDB = LDB + 1
                    Skip tests if not enough room.
                    IF( LDB.GT.NMAX )&
                        GO TO 70
                    LBB = LDB*NB
!
                    Generate the matrix B.
!
                    CALL ZMAKE( 'GE', ' ', ' ', MB, NB, B, NMAX, BB,&
                        LDB, RESET, ZERO )
!
                    DO 60 IA = 1, NALF
                        ALPHA = ALF( IA )
!
                        DO 50 IB = 1, NBET
                            BETA = BET( IB )
!
                            Generate the matrix C.
!
                            CALL ZMAKE( 'GE', ' ', ' ', M, N, C, NMAX,&

```



```

IF( LDB.GT.NMAX )&
  GO TO 90
LBB = LDB*N
!
!
! Generate the matrix B.
!
CALL ZMAKE( 'GE', ' ', ' ', M, N, B, NMAX, BB, LDB, RESET,&
  ZERO )
!
DO 80 ICS = 1, 2
  SIDE = ICHS( ICS: ICS )
  LEFT = SIDE.EQ.'L'
!
  IF( LEFT )THEN
    NA = M
  ELSE
    NA = N
  END IF
! Set LDA to 1 more than minimum value if room.
  LDA = NA
  IF( LDA.LT.NMAX )&
    LDA = LDA + 1
! Skip tests if not enough room.
  IF( LDA.GT.NMAX )&
    GO TO 80
  LAA = LDA*NA
!
DO 70 ICU = 1, 2
  UPLO = ICHU( ICU: ICU )
!
! Generate the hermitian or symmetric matrix A.
!
CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', NA, NA, A, NMAX,&
  AA, LDA, RESET, ZERO )
!
DO 60 IA = 1, NALF
  ALPHA = ALF( IA )
!
DO 50 IB = 1, NBET
  BETA = BET( IB )
!
! Generate the matrix C.
!
CALL ZMAKE( 'GE', ' ', ' ', M, N, C, NMAX, CC,&
  LDC, RESET, ZERO )
!
NC = NC + 1
!
! Save every datum before calling the
! subroutine.
!
SIDES = SIDE
UPLOS = UPLO
MS = M
NS = N
ALS = ALPHA
DO 10 I = 1, LAA
  AS( I ) = AA( I )
10 CONTINUE
LDAS = LDA
DO 20 I = 1, LBB
  BS( I ) = BB( I )
20 CONTINUE
LDBS = LDB
BLS = BETA
DO 30 I = 1, LCC
  CS( I ) = CC( I )
30 CONTINUE
LDCS = LDC
!
! Call the subroutine.
!
IF( TRACE )&
  WRITE( NTRA, FMT = 9995 )NC, SNAME, SIDE,&
    UPLO, M, N, ALPHA, LDA, LDB, BETA, LDC
IF( REWI )&
  REWIND NTRA
IF( CONJ )THEN
  CALL ZHEMM( SIDE, UPLO, M, N, ALPHA, AA, LDA,&
    BB, LDB, BETA, CC, LDC )
ELSE
  CALL ZSYMM( SIDE, UPLO, M, N, ALPHA, AA, LDA,&
    BB, LDB, BETA, CC, LDC )
END IF
!
! Check if error-exit was taken incorrectly.
!
IF( .NOT.OK )THEN
  WRITE( NOUT, FMT = 9994 )
  FATAL = .TRUE.
  GO TO 110
END IF
!
! See what data changed inside subroutines.
!
ISAME( 1 ) = SIDES.EQ.SIDE
ISAME( 2 ) = UPLOS.EQ.UPLO
ISAME( 3 ) = MS.EQ.M
ISAME( 4 ) = NS.EQ.N
ISAME( 5 ) = ALS.EQ.ALPHA
ISAME( 6 ) = LZE( AS, AA, LAA )
ISAME( 7 ) = LDAS.EQ.LDA
ISAME( 8 ) = LZE( BS, BB, LBB )
ISAME( 9 ) = LDBS.EQ.LDB
ISAME( 10 ) = BLS.EQ.BETA
IF( NULL )THEN
  ISAME( 11 ) = LZE( CS, CC, LCC )
ELSE
  ISAME( 11 ) = LZERES( 'GE', ' ', M, N, CS,&
    CC, LDC )
END IF
ISAME( 12 ) = LDCS.EQ.LDC
!
! If data was incorrectly changed, report and

```

```

!
!
!       return.
!
!       SAME = .TRUE.
!       DO 40 I = 1, NARGS
!           SAME = SAME.AND.ISAME( I )
!           IF( .NOT.ISAME( I ) )&
40      WRITE( NOUT, FMT = 9998 ) I
!       CONTINUE
!       IF( .NOT.SAME ) THEN
!           FATAL = .TRUE.
!           GO TO 110
!       END IF
!
!       IF( .NOT.NULL ) THEN
!
!           Check the result.
!
!           IF( LEFT ) THEN
!               CALL ZMMCH( 'N', 'N', M, N, M, ALPHA, A,&
!                           NMAX, B, NMAX, BETA, C, NMAX,&
!                           CT, G, CC, LDC, EPS, ERR,&
!                           FATAL, NOUT, .TRUE. )
!           ELSE
!               CALL ZMMCH( 'N', 'N', M, N, N, ALPHA, B,&
!                           NMAX, A, NMAX, BETA, C, NMAX,&
!                           CT, G, CC, LDC, EPS, ERR,&
!                           FATAL, NOUT, .TRUE. )
!           END IF
!           ERRMAX = MAX( ERRMAX, ERR )
!           If got really bad answer, report and
!           return.
!           IF( FATAL )&
!               GO TO 110
!       END IF
!
!       50      CONTINUE
!
!       60      CONTINUE
!
!       70      CONTINUE
!
!       80      CONTINUE
!
!       90      CONTINUE
!
!       100 CONTINUE
!
!       Report result.
!
!       IF( ERRMAX.LT.THRESH ) THEN
!           WRITE( NOUT, FMT = 9999 ) SNAME, NC
!       ELSE
!           WRITE( NOUT, FMT = 9997 ) SNAME, NC, ERRMAX
!       END IF
!       GO TO 120
!
!       110 CONTINUE
!       WRITE( NOUT, FMT = 9996 ) SNAME
!       WRITE( NOUT, FMT = 9995 ) NC, SNAME, SIDE, UPLO, M, N, ALPHA, LDA,&
!           LDB, BETA, LDC
!
!       120 CONTINUE
!       RETURN
!
!       9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!           'S)' )
!       9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!           'ANGED INCORRECTLY *****' )
!       9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!           'ALLS)', '/' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!           ' - SUSPECT *****' )
!       9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
!       9995 FORMAT( 1X, I6, ': ', A6, '(', 2( ' ', A1, ' ', ' '), 2( I3, ' ', ' '),&
!           '(', F4.1, ' ', ' ', F4.1, ' '), A, ' ', I3, ' ', B, ' ', I3, ' ', '(', F4.1,&
!           ' ', ' ', F4.1, ' '), C, ' ', I3, ' ' )
!       9994 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
!           '*****' )
!
!       End of ZCHK2.
!
!       END SUBROUTINE
!       SUBROUTINE ZCHK3( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!           FATAL, NIDIM, IDIM, NALF, ALF, NMAX, A, AA, AS,&
!           B, BB, BS, CT, G, C )
!
!       Tests ZTRMM and ZTRSM.
!
!       Auxiliary routine for test program for Level 3 Blas.
!
!       -- Written on 8-February-1989.
!       Jack Dongarra, Argonne National Laboratory.
!       Iain Duff, AERE Harwell.
!       Jeremy Du Croz, Numerical Algorithms Group Ltd.
!       Sven Hammarling, Numerical Algorithms Group Ltd.
!
!       .. Parameters ..
!       COMPLEX*16      ZERO, ONE
!       PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ),&
!           ONE = ( 1.0D0, 0.0D0 ) )
!       DOUBLE PRECISION RZERO
!       PARAMETER      ( RZERO = 0.0D0 )
!
!       .. Scalar Arguments ..
!       DOUBLE PRECISION EPS, THRESH
!       INTEGER         NALF, NIDIM, NMAX, NOUT, NTRA
!       LOGICAL         FATAL, REWI, TRACE
!       CHARACTER*6     SNAME
!
!       .. Array Arguments ..
!       COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
!           AS( NMAX*NMAX ), B( NMAX, NMAX ),&
!           BB( NMAX*NMAX ), BS( NMAX*NMAX ),&
!           C( NMAX, NMAX ), CT( NMAX )
!       DOUBLE PRECISION G( NMAX )
!       INTEGER         IDIM( NIDIM )
!
!       .. Local Scalars ..

```

```

COMPLEX*16      ALPHA, ALS
DOUBLE PRECISION  ERR, ERRMAX
INTEGER          I, IA, ICD, ICS, ICT, ICU, IM, IN, J, LAA, LBB,&
                LDA, LDAS, LDB, LDBS, M, MS, N, NA, NARGS, NC,&
                NS
LOGICAL          LEFT, NULL, RESET, SAME
CHARACTER*1      DIAG, DIAGS, SIDE, SIDES, TRANAS, TRANSA, UPLO,&
                UPLOS
CHARACTER*2      ICHD, ICHS, ICHU
CHARACTER*3      ICHT
!
! .. Local Arrays ..
LOGICAL          ISAME( 13 )
!
! .. External Functions ..
LOGICAL          LZE, LZERES
!
! .. External Subroutines ..
EXTERNAL         ZMAKE, ZMMCH, ZTRMM, ZTRSM
!
! .. Intrinsic Functions ..
INTRINSIC        MAX
!
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
!
! .. Common blocks ..
COMMON           /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA            ICHU/'UL'//, ICHT/'NTC'//, ICHD/'UN'//, ICHS/'LR'//
!
! .. Executable Statements ..
!
NARGS = 11
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
! Set up zero matrix for ZMMCH.
DO 20 J = 1, NMAX
    DO 10 I = 1, NMAX
        C( I, J ) = ZERO
10    CONTINUE
20    CONTINUE
!
DO 140 IM = 1, NIDIM
    M = IDIM( IM )
!
DO 130 IN = 1, NIDIM
    N = IDIM( IN )
!
! Set LDB to 1 more than minimum value if room.
LDB = M
IF( LDB.LT.NMAX )&
    LDB = LDB + 1
!
! Skip tests if not enough room.
IF( LDB.GT.NMAX )&
    GO TO 130
LBB = LDB*N
NULL = M.LE.0.OR.N.LE.0
!
DO 120 ICS = 1, 2
    SIDE = ICHS( ICS: ICS )
    LEFT = SIDE.EQ.'L'
    IF( LEFT )THEN
        NA = M
    ELSE
        NA = N
    END IF
!
! Set LDA to 1 more than minimum value if room.
LDA = NA
IF( LDA.LT.NMAX )&
    LDA = LDA + 1
!
! Skip tests if not enough room.
IF( LDA.GT.NMAX )&
    GO TO 130
LAA = LDA*NA
!
DO 110 ICU = 1, 2
    UPLO = ICHU( ICU: ICU )
!
DO 100 ICT = 1, 3
    TRANSA = ICHT( ICT: ICT )
!
DO 90 ICD = 1, 2
    DIAG = ICHD( ICD: ICD )
!
DO 80 IA = 1, NALF
    ALPHA = ALF( IA )
!
! Generate the matrix A.
CALL ZMAKE( 'TR', UPLO, DIAG, NA, NA, A,&
            NMAX, AA, LDA, RESET, ZERO )
!
! Generate the matrix B.
CALL ZMAKE( 'GE', ' ', ' ', M, N, B, NMAX,&
            BB, LDB, RESET, ZERO )
!
NC = NC + 1
!
! Save every datum before calling the
! subroutine.
SIDES = SIDE
UPLOS = UPLO
TRANAS = TRANSA
DIAGS = DIAG
MS = M
NS = N
ALS = ALPHA
DO 30 I = 1, LAA
    AS( I ) = AA( I )
30    CONTINUE
LDAS = LDA
DO 40 I = 1, LBB
    BS( I ) = BB( I )
40    CONTINUE
LDBS = LDB
!

```

```

      CALL the subroutine.

      IF( SNAME( 4: 5 ).EQ.'MM' ) THEN
        IF( TRACE ) &
          WRITE( NTRA, FMT = 9995 ) NC, SNAME, &
            SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, &
            LDA, LDB
          IF( REWI ) &
            REWIND NTRA
          CALL ZTRMM( SIDE, UPLO, TRANSA, DIAG, M, &
                     N, ALPHA, AA, LDA, BB, LDB )
        ELSE IF( SNAME( 4: 5 ).EQ.'SM' ) THEN
          IF( TRACE ) &
            WRITE( NTRA, FMT = 9995 ) NC, SNAME, &
              SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, &
              LDA, LDB
          IF( REWI ) &
            REWIND NTRA
          CALL ZTRSM( SIDE, UPLO, TRANSA, DIAG, M, &
                     N, ALPHA, AA, LDA, BB, LDB )
        END IF

        Check if error-exit was taken incorrectly.

        IF( .NOT.OK ) THEN
          WRITE( NOUT, FMT = 9994 )
          FATAL = .TRUE.
          GO TO 150
        END IF

        See what data changed inside subroutines.

        ISAME( 1 ) = SIDES.EQ.SIDE
        ISAME( 2 ) = UPLOS.EQ.UPLO
        ISAME( 3 ) = TRANAS.EQ.TRANSA
        ISAME( 4 ) = DIAGS.EQ.DIAG
        ISAME( 5 ) = MS.EQ.M
        ISAME( 6 ) = NS.EQ.N
        ISAME( 7 ) = ALS.EQ.ALPHA
        ISAME( 8 ) = LZE( AS, AA, LAA )
        ISAME( 9 ) = LDAS.EQ.LDA
        IF( NULL ) THEN
          ISAME( 10 ) = LZE( BS, BB, LBB )
        ELSE
          ISAME( 10 ) = LZERES( 'GE', ' ', M, N, BS, &
                                BB, LDB )
        END IF
        ISAME( 11 ) = LDBS.EQ.LDB

        If data was incorrectly changed, report and
        return.

        SAME = .TRUE.
        DO 50 I = 1, NARGS
          SAME = SAME.AND. ISAME( I )
          IF( .NOT.ISAME( I ) ) &
            WRITE( NOUT, FMT = 9998 ) I
        CONTINUE
        IF( .NOT.SAME ) THEN
          FATAL = .TRUE.
          GO TO 150
        END IF

        IF( .NOT.NULL ) THEN
          IF( SNAME( 4: 5 ).EQ.'MM' ) THEN

            Check the result.

            IF( LEFT ) THEN
              CALL ZMMCH( TRANSA, 'N', M, N, M, &
                          ALPHA, A, NMAX, B, NMAX, &
                          ZERO, C, NMAX, CT, G, &
                          BB, LDB, EPS, ERR, &
                          FATAL, NOUT, .TRUE. )
            ELSE
              CALL ZMMCH( 'N', TRANSA, M, N, N, &
                          ALPHA, B, NMAX, A, NMAX, &
                          ZERO, C, NMAX, CT, G, &
                          BB, LDB, EPS, ERR, &
                          FATAL, NOUT, .TRUE. )
            END IF
          ELSE IF( SNAME( 4: 5 ).EQ.'SM' ) THEN

            Compute approximation to original
            matrix.

            DO 70 J = 1, N
              DO 60 I = 1, M
                C( I, J ) = BB( I + ( J - 1 ) * &
                               LDB )
                BB( I + ( J - 1 ) * LDB ) = ALPHA * &
                  B( I, J )
              CONTINUE
            CONTINUE

            IF( LEFT ) THEN
              CALL ZMMCH( TRANSA, 'N', M, N, M, &
                          ONE, A, NMAX, C, NMAX, &
                          ZERO, B, NMAX, CT, G, &
                          BB, LDB, EPS, ERR, &
                          FATAL, NOUT, .FALSE. )
            ELSE
              CALL ZMMCH( 'N', TRANSA, M, N, N, &
                          ONE, C, NMAX, A, NMAX, &
                          ZERO, B, NMAX, CT, G, &
                          BB, LDB, EPS, ERR, &
                          FATAL, NOUT, .FALSE. )
            END IF
          END IF
          ERRMAX = MAX( ERRMAX, ERR )
          If got really bad answer, report and
          return.
          IF( FATAL ) &
            GO TO 150
        END IF
      END IF
    END IF
  END IF
END IF

```

```

      END IF
!
!      80      CONTINUE
!
!      90      CONTINUE
!
!      100     CONTINUE
!
!      110     CONTINUE
!
!      120     CONTINUE
!
!      130     CONTINUE
!
!      140     CONTINUE
!
!      Report result.
!
!      IF( ERRMAX.LT.THRESH )THEN
!        WRITE( NOUT, FMT = 9999 )SNAME, NC
!      ELSE
!        WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
!      END IF
!      GO TO 160
!
!      150     CONTINUE
!      WRITE( NOUT, FMT = 9996 )SNAME
!      WRITE( NOUT, FMT = 9995 )NC, SNAME, SIDE, UPLO, TRANSA, DIAG, M,&
!        N, ALPHA, LDA, LDB
!
!      160     CONTINUE
!      RETURN
!
!      9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
!        'S)' )
!      9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
!        'ANGED INCORRECTLY *****' )
!      9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
!        'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
!        ' - SUSPECT *****' )
!      9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
!      9995 FORMAT( 1X, I6, ': ', A6, '((', 4( ' ', A1, ' ', ' ), 2( I3, ' ', ' ),&
!        '(', F4.1, ' ', ' ', F4.1, ' ), A, ', I3, ' ', B, ', I3, ' ',&
!        ' ', ' )' )
!      9994 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
!        '*****' )
!
!      End of ZCHK3.
!
!      END SUBROUTINE
!      SUBROUTINE ZCHK4( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
!        FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
!        A, AA, AS, B, BB, BS, C, CC, CS, CT, G )
!
!      Tests ZHERK and ZSYRK.
!
!      Auxiliary routine for test program for Level 3 Blas.
!
!      -- Written on 8-February-1989.
!      Jack Dongarra, Argonne National Laboratory.
!      Iain Duff, AERE Harwell.
!      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!      Sven Hammarling, Numerical Algorithms Group Ltd.
!
!      .. Parameters ..
!      COMPLEX*16      ZERO
!      PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ) )
!      DOUBLE PRECISION RONE, RZERO
!      PARAMETER      ( RONE = 1.0D0, RZERO = 0.0D0 )
!
!      .. Scalar Arguments ..
!      DOUBLE PRECISION EPS, THRESH
!      INTEGER          NALF, NBET, NIDIM, NMAX, NOUT, NTRA
!      LOGICAL          FATAL, REWI, TRACE
!      CHARACTER*6      SNAME
!
!      .. Array Arguments ..
!      COMPLEX*16      A( NMAX, NMAX ), AA( NMAX*NMAX ), ALF( NALF ),&
!        AS( NMAX*NMAX ), B( NMAX, NMAX ),&
!        BB( NMAX*NMAX ), BET( NBET ), BS( NMAX*NMAX ),&
!        C( NMAX, NMAX ), CC( NMAX*NMAX ),&
!        CS( NMAX*NMAX ), CT( NMAX )
!      DOUBLE PRECISION
!      IDIM( NIDIM )
!
!      .. Local Scalars ..
!      COMPLEX*16      ALPHA, ALS, BETA, BETS
!      DOUBLE PRECISION ERR, ERRMAX, RALPHA, RALS, RBETA, RBETS
!      INTEGER          I, IA, IB, ICT, ICU, IK, IN, J, JC, JJ, K, KS,&
!        LAA, LCC, LDA, LDAS, LDC, LDCS, LJ, MA, N, NA,&
!        NARGS, NC, NS
!      LOGICAL          CONJ, NULL, RESET, SAME, TRAN, UPPER
!      CHARACTER*1      TRANS, TRANSS, TRANST, UPLO, UPLOS
!      CHARACTER*2      ICHT, ICHU
!
!      .. Local Arrays ..
!      LOGICAL          ISAME( 13 )
!
!      .. External Functions ..
!      LOGICAL          LZE, LZERS
!      EXTERNAL          LZE, LZERS
!
!      .. External Subroutines ..
!      EXTERNAL          ZHERK, ZMAKE, ZMMCH, ZSYRK
!
!      .. Intrinsic Functions ..
!      INTRINSIC          DCMPLX, MAX, DBLE
!
!      .. Scalars in Common ..
!      INTEGER          INFOT, NOUTC
!      LOGICAL          LERR, OK
!
!      .. Common blocks ..
!      COMMON            /INFOC/INFOT, NOUTC, OK, LERR
!
!      .. Data statements ..
!      DATA             ICHT/'NC'//, ICHU/'UL'/
!
!      .. Executable Statements ..
!      CONJ = SNAME( 2: 3 ).EQ.'HE'
!
!      NARGS = 10
!      NC = 0
!      RESET = .TRUE.
!      ERRMAX = RZERO

```

```

!
DO 100 IN = 1, NIDIM
  N = IDIM( IN )
  Set LDC to 1 more than minimum value if room.
  LDC = N
  IF( LDC.LT.NMAX ) &
    LDC = LDC + 1
  Skip tests if not enough room.
  IF( LDC.GT.NMAX ) &
    GO TO 100
  LCC = LDC*N
!
DO 90 IK = 1, NIDIM
  K = IDIM( IK )
!
  DO 80 ICT = 1, 2
    TRANS = ICHT( ICT: ICT )
    TRAN = TRANS.EQ.'C'
    IF( TRAN.AND..NOT.CONJ ) &
      TRANS = 'T'
    IF( TRAN ) THEN
      MA = K
      NA = N
    ELSE
      MA = N
      NA = K
    END IF
    Set LDA to 1 more than minimum value if room.
    LDA = MA
    IF( LDA.LT.NMAX ) &
      LDA = LDA + 1
    Skip tests if not enough room.
    IF( LDA.GT.NMAX ) &
      GO TO 80
    LAA = LDA*NA
!
    Generate the matrix A.
    CALL ZMAKE( 'GE', ' ', ' ', MA, NA, A, NMAX, AA, LDA, &
      RESET, ZERO )
!
    DO 70 ICU = 1, 2
      UPLO = ICHU( ICU: ICU )
      UPPER = UPLO.EQ.'U'
!
      DO 60 IA = 1, NALF
        ALPHA = ALF( IA )
        IF( CONJ ) THEN
          RALPHA = DBLE( ALPHA )
          ALPHA = DCMPLX( RALPHA, RZERO )
        END IF
!
        DO 50 IB = 1, NBET
          BETA = BET( IB )
          IF( CONJ ) THEN
            RBETA = DBLE( BETA )
            BETA = DCMPLX( RBETA, RZERO )
          END IF
          NULL = N.LE.0
          IF( CONJ ) &
            NULL = NULL.OR.( ( K.LE.0.OR.RALPHA.EQ.&
              RZERO ).AND.RBETA.EQ.RONE )
!
          Generate the matrix C.
          CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, C, &
            NMAX, CC, LDC, RESET, ZERO )
!
          NC = NC + 1
!
          Save every datum before calling the subroutine.
          UPLOS = UPLO
          TRANSS = TRANS
          NS = N
          KS = K
          IF( CONJ ) THEN
            RALS = RALPHA
          ELSE
            ALS = ALPHA
          END IF
          DO 10 I = 1, LAA
            AS( I ) = AA( I )
          CONTINUE
          LDAS = LDA
          IF( CONJ ) THEN
            RBETS = RBETA
          ELSE
            BETS = BETA
          END IF
          DO 20 I = 1, LCC
            CS( I ) = CC( I )
          CONTINUE
          LDCS = LDC
!
          Call the subroutine.
          IF( CONJ ) THEN
            IF( TRACE ) &
              WRITE( NTRA, FMT = 9994 ) NC, SNAME, UPLO, &
                TRANS, N, K, RALPHA, LDA, RBETA, LDC
            IF( REWI ) &
              REWIND NTRA
            CALL ZHERK( UPLO, TRANS, N, K, RALPHA, AA, &
              LDA, RBETA, CC, LDC )
          ELSE
            IF( TRACE ) &
              WRITE( NTRA, FMT = 9993 ) NC, SNAME, UPLO, &
                TRANS, N, K, ALPHA, LDA, BETA, LDC
            IF( REWI ) &
              REWIND NTRA
            CALL ZSYRK( UPLO, TRANS, N, K, ALPHA, AA, &
              LDA, BETA, CC, LDC )
          END IF
!
10
20
!
!
!

```

```

!
!
!
Check if error-exit was taken incorrectly.

IF( .NOT.OK )THEN
  WRITE( NOUT, FMT = 9992 )
  FATAL = .TRUE.
  GO TO 120
END IF

!
!
!
See what data changed inside subroutines.

ISAME( 1 ) = UPLOS.EQ.UPLO
ISAME( 2 ) = TRANSS.EQ.TRANS
ISAME( 3 ) = NS.EQ.N
ISAME( 4 ) = KS.EQ.K
IF( CONJ )THEN
  ISAME( 5 ) = RALS.EQ.RALPHA
ELSE
  ISAME( 5 ) = ALS.EQ.ALPHA
END IF
ISAME( 6 ) = LZE( AS, AA, LAA )
ISAME( 7 ) = LDAS.EQ.LDA
IF( CONJ )THEN
  ISAME( 8 ) = RBETS.EQ.RBETA
ELSE
  ISAME( 8 ) = BETS.EQ.BETA
END IF
IF( NULL )THEN
  ISAME( 9 ) = LZE( CS, CC, LCC )
ELSE
  ISAME( 9 ) = LZERES( SNAME( 2: 3 ), UPLO, N,&
    N, CS, CC, LDC )
END IF
ISAME( 10 ) = LDCS.EQ.LDC

!
!
!
If data was incorrectly changed, report and
return.

SAME = .TRUE.
DO 30 I = 1, NARGS
  SAME = SAME.AND.ISAME( I )
  IF( .NOT.ISAME( I ) )&
    WRITE( NOUT, FMT = 9998 ) I
CONTINUE
30 IF( .NOT.SAME )THEN
  FATAL = .TRUE.
  GO TO 120
END IF

!
!
!
IF( .NOT.NULL )THEN

  Check the result column by column.

  IF( CONJ )THEN
    TRANST = 'C'
  ELSE
    TRANST = 'T'
  END IF
  JC = 1
  DO 40 J = 1, N
    IF( UPPER )THEN
      JJ = 1
      LJ = J
    ELSE
      JJ = J
      LJ = N - J + 1
    END IF
    IF( TRAN )THEN
      CALL ZMMCH( TRANST, 'N', LJ, 1, K,&
        ALPHA, A( 1, JJ ), NMAX,&
        A( 1, J ), NMAX, BETA,&
        C( JJ, J ), NMAX, CT, G,&
        CC( JC ), LDC, EPS, ERR,&
        FATAL, NOUT, .TRUE. )
    ELSE
      CALL ZMMCH( 'N', TRANST, LJ, 1, K,&
        ALPHA, A( JJ, 1 ), NMAX,&
        A( J, 1 ), NMAX, BETA,&
        C( JJ, J ), NMAX, CT, G,&
        CC( JC ), LDC, EPS, ERR,&
        FATAL, NOUT, .TRUE. )
    END IF
    IF( UPPER )THEN
      JC = JC + LDC
    ELSE
      JC = JC + LDC + 1
    END IF
    ERRMAX = MAX( ERRMAX, ERR )
    If got really bad answer, report and
    return.
    IF( FATAL )&
      GO TO 110
  40 CONTINUE
  END IF

!
!
!
50 CONTINUE
!
!
!
60 CONTINUE
!
!
!
70 CONTINUE
!
!
!
80 CONTINUE
!
!
!
90 CONTINUE
!
!
!
100 CONTINUE
!
!
!
Report result.

IF( ERRMAX.LT.THRESH )THEN
  WRITE( NOUT, FMT = 9999 )SNAME, NC
ELSE
  WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
END IF

```

```

GO TO 130
!
110 CONTINUE
IF( N.GT.1 )&
WRITE( NOUT, FMT = 9995 )J
!
120 CONTINUE
WRITE( NOUT, FMT = 9996 )SNAME
IF( CONJ )THEN
WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, TRANS, N, K, RALPHA,&
LDA, RBETA, LDC
ELSE
WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, TRANS, N, K, ALPHA,&
LDA, BETA, LDC
END IF
!
130 CONTINUE
RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( ' ', THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '( ', 2( ' ', A1, ' ', ' ', ' ), 2( I3, ' ', ' ),&
F4.1, ' ', A, ' ', I3, ' ', ' ', F4.1, ' ', C, ' ', I3, ' ', ' ',&
' ', ' )
9993 FORMAT( 1X, I6, ': ', A6, '( ', 2( ' ', A1, ' ', ' ', ' ), 2( I3, ' ', ' ),&
' ', F4.1, ' ', ' ', F4.1, ' ', ' ', A, ' ', I3, ' ', ' ', F4.1, ' ', ' ', F4.1,&
' ', C, ' ', I3, ' ', ' ', ' )
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
'*****' )
!
End of ZCHK4.
!
END SUBROUTINE
SUBROUTINE ZCHK5( SNAME, EPS, THRESH, NOUT, NTRA, TRACE, REWI,&
FATAL, NIDIM, IDIM, NALF, ALF, NBET, BET, NMAX,&
AB, AA, AS, BB, BS, C, CC, CS, CT, G, W )
!
! Tests ZHER2K and ZSYR2K.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Parameters ..
COMPLEX*16 ZERO, ONE
PARAMETER
( ZERO = ( 0.0D0, 0.0D0 ),&
ONE = ( 1.0D0, 0.0D0 ) )
DOUBLE PRECISION RONE, RZERO
PARAMETER
( RONE = 1.0D0, RZERO = 0.0D0 )
!
! .. Scalar Arguments ..
DOUBLE PRECISION EPS, THRESH
INTEGER NALF, NBET, NIDIM, NMAX, NOUT, NTRA
LOGICAL FATAL, REWI, TRACE
CHARACTER*6 SNAME
!
! .. Array Arguments ..
COMPLEX*16 AA( NMAX*NMAX ), AB( 2*NMAX*NMAX ),&
ALF( NALF ), AS( NMAX*NMAX ), BB( NMAX*NMAX ),&
BET( NBET ), BS( NMAX*NMAX ), C( NMAX, NMAX ),&
CC( NMAX*NMAX ), CS( NMAX*NMAX ), CT( NMAX ),&
W( 2*NMAX )
DOUBLE PRECISION G( NMAX )
INTEGER IDIM( NIDIM )
!
! .. Local Scalars ..
COMPLEX*16 ALPHA, ALS, BETA, BETS
DOUBLE PRECISION ERR, ERRMAX, RBETA, RBETS
INTEGER I, IA, IB, ICT, ICU, IK, IN, J, JC, JJ, JJAB,&
K, KS, LAA, LBB, LCC, LDA, LDAS, LDB, LDBS,&
LDC, LDCS, LJ, MA, N, NA, NARGS, NC, NS
LOGICAL CONJ, NULL, RESET, SAME, TRAN, UPPER
CHARACTER*1 TRANS, TRANSS, TRANST, UPLO, UPLOS
CHARACTER*2 ICHT, ICHU
!
! .. Local Arrays ..
LOGICAL ISAME( 13 )
!
! .. External Functions ..
LOGICAL LZE, LZERES
EXTERNAL LZE, LZERES
!
! .. External Subroutines ..
EXTERNAL ZHER2K, ZMAKE, ZMMCH, ZSYR2K
!
! .. Intrinsic Functions ..
INTRINSIC DCMPLX, DCONJG, MAX, DBLE
!
! .. Scalars in Common ..
INTEGER INFOT, NOUTC
LOGICAL LERR, OK
!
! .. Common blocks ..
COMMON /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Data statements ..
DATA ICHT/'NC'/, ICHU/'UL'/
!
! .. Executable Statements ..
CONJ = SNAME( 2: 3 ).EQ.'HE'
!
NARGS = 12
NC = 0
RESET = .TRUE.
ERRMAX = RZERO
!
DO 130 IN = 1, NIDIM
N = IDIM( IN )
Set LDC to 1 more than minimum value if room.
LDC = N
IF( LDC.LT.NMAX )&
LDC = LDC + 1
Skip tests if not enough room.
IF( LDC.GT.NMAX )&

```

```

      GO TO 130
      LCC = LDC*N
!
DO 120 IK = 1, NIDIM
      K = IDIM( IK )
!
      DO 110 ICT = 1, 2
        TRANS = ICHT( ICT: ICT )
        TRAN = TRANS.EQ.'C'
        IF( TRAN.AND..NOT.CONJ )&
          TRANS = 'T'
        IF( TRAN )THEN
          MA = K
          NA = N
        ELSE
          MA = N
          NA = K
        END IF
        Set LDA to 1 more than minimum value if room.
        LDA = MA
        IF( LDA.LT.NMAX )&
          LDA = LDA + 1
        Skip tests if not enough room.
        IF( LDA.GT.NMAX )&
          GO TO 110
        LAA = LDA*NA
!
        Generate the matrix A.
!
        IF( TRAN )THEN
          CALL ZMAKE( 'GE', ' ', ' ', MA, NA, AB, 2*NMAX, AA,&
            LDA, RESET, ZERO )
        ELSE
          CALL ZMAKE( 'GE', ' ', ' ', MA, NA, AB, NMAX, AA, LDA,&
            RESET, ZERO )
        END IF
!
        Generate the matrix B.
!
        LDB = LDA
        LBB = LAA
        IF( TRAN )THEN
          CALL ZMAKE( 'GE', ' ', ' ', MA, NA, AB( K + 1 ),&
            2*NMAX, BB, LDB, RESET, ZERO )
        ELSE
          CALL ZMAKE( 'GE', ' ', ' ', MA, NA, AB( K*NMAX + 1 ),&
            NMAX, BB, LDB, RESET, ZERO )
        END IF
!
      DO 100 ICU = 1, 2
        UPLO = ICHU( ICU: ICU )
        UPPER = UPLO.EQ.'U'
!
        DO 90 IA = 1, NALF
          ALPHA = ALF( IA )
!
          DO 80 IB = 1, NBET
            BETA = BET( IB )
            IF( CONJ )THEN
              RBETA = DBLE( BETA )
              BETA = DCMPLX( RBETA, RZERO )
            END IF
            NULL = N.LE.0
            IF( CONJ )&
              NULL = NULL.OR.( ( K.LE.0.OR.ALPHA.EQ.&
                ZERO ).AND.RBETA.EQ.RONE )
!
            Generate the matrix C.
!
            CALL ZMAKE( SNAME( 2: 3 ), UPLO, ' ', N, N, C,&
              NMAX, CC, LDC, RESET, ZERO )
!
            NC = NC + 1
!
            Save every datum before calling the subroutine.
!
            UPLOS = UPLO
            TRANSS = TRANS
            NS = N
            KS = K
            ALS = ALPHA
            DO 10 I = 1, LAA
              AS( I ) = AA( I )
            CONTINUE
            LDAS = LDA
            DO 20 I = 1, LBB
              BS( I ) = BB( I )
            CONTINUE
            LDBS = LDB
            IF( CONJ )THEN
              RBETS = RBETA
            ELSE
              BETS = BETA
            END IF
            DO 30 I = 1, LCC
              CS( I ) = CC( I )
            CONTINUE
            LDCS = LDC
!
            Call the subroutine.
!
            IF( CONJ )THEN
              IF( TRACE )&
                WRITE( NTRA, FMT = 9994 )NC, SNAME, UPLO,&
                  TRANS, N, K, ALPHA, LDA, LDB, RBETA, LDC
              IF( REWI )&
                REWIND NTRA
              CALL ZHER2K( UPLO, TRANS, N, K, ALPHA, AA,&
                LDA, BB, LDB, RBETA, CC, LDC )
            ELSE
              IF( TRACE )&
                WRITE( NTRA, FMT = 9993 )NC, SNAME, UPLO,&
                  TRANS, N, K, ALPHA, LDA, LDB, BETA, LDC
              IF( REWI )&

```

```

        REWIND NTRA
        CALL ZSYR2K( UPLO, TRANS, N, K, ALPHA, AA,&
                     LDA, BB, LDB, BETA, CC, LDC )
    END IF

    !
    !
    !
    Check if error-exit was taken incorrectly.

    IF( .NOT.OK )THEN
        WRITE( NOUT, FMT = 9992 )
        FATAL = .TRUE.
        GO TO 150
    END IF

    !
    !
    !
    See what data changed inside subroutines.

    ISAME( 1 ) = UPLOS.EQ.UPLO
    ISAME( 2 ) = TRANS.EQ.TRANS
    ISAME( 3 ) = NS.EQ.N
    ISAME( 4 ) = KS.EQ.K
    ISAME( 5 ) = ALS.EQ.ALPHA
    ISAME( 6 ) = LZE( AS, AA, LAA )
    ISAME( 7 ) = LDAS.EQ.LDA
    ISAME( 8 ) = LZE( BS, BB, LBB )
    ISAME( 9 ) = LDBS.EQ.LDB
    IF( CONJ )THEN
        ISAME( 10 ) = RBETS.EQ.RBETA
    ELSE
        ISAME( 10 ) = BETS.EQ.BETA
    END IF
    IF( NULL )THEN
        ISAME( 11 ) = LZE( CS, CC, LCC )
    ELSE
        ISAME( 11 ) = LZERES( 'HE', UPLO, N, N, CS,&
                             CC, LDC )
    END IF
    ISAME( 12 ) = LDCS.EQ.LDC

    !
    !
    !
    If data was incorrectly changed, report and
    return.

    SAME = .TRUE.
    DO 40 I = 1, NARGS
        SAME = SAME.AND.ISAME( I )
        IF( .NOT.ISAME( I ) )&
            WRITE( NOUT, FMT = 9998 ) I
    CONTINUE
    IF( .NOT.SAME )THEN
        FATAL = .TRUE.
        GO TO 150
    END IF

    !
    !
    !
    IF( .NOT.NULL )THEN

        Check the result column by column.

        IF( CONJ )THEN
            TRANST = 'C'
        ELSE
            TRANST = 'T'
        END IF
        JJAB = 1
        JC = 1
        DO 70 J = 1, N
            IF( UPPER )THEN
                JJ = 1
                LJ = J
            ELSE
                JJ = J
                LJ = N - J + 1
            END IF
            IF( TRAN )THEN
                DO 50 I = 1, K
                    W( I ) = ALPHA*AB( ( J - 1 ) * 2 * &
                                         NMAX + K + I )
                    IF( CONJ )THEN
                        W( K + I ) = DCONJG( ALPHA ) * &
                                      AB( ( J - 1 ) * 2 * &
                                         NMAX + I )
                    ELSE
                        W( K + I ) = ALPHA * &
                                      AB( ( J - 1 ) * 2 * &
                                         NMAX + I )
                    END IF
                CONTINUE
                CALL ZMMCH( TRANST, 'N', LJ, 1, 2 * K, &
                           ONE, AB( JJAB ), 2 * NMAX, W, &
                           2 * NMAX, BETA, C( JJ, J ), &
                           NMAX, CT, G, CC( JC ), LDC, &
                           EPS, ERR, FATAL, NOUT, &
                           .TRUE. )
            ELSE
                DO 60 I = 1, K
                    IF( CONJ )THEN
                        W( I ) = ALPHA * DCONJG( AB( ( K + &
                                                         I - 1 ) * NMAX + J ) )
                        W( K + I ) = DCONJG( ALPHA * &
                                              AB( ( I - 1 ) * NMAX + &
                                                  J ) )
                    ELSE
                        W( I ) = ALPHA * AB( ( K + I - 1 ) * &
                                              NMAX + J )
                        W( K + I ) = ALPHA * &
                                              AB( ( I - 1 ) * NMAX + &
                                                  J )
                    END IF
                CONTINUE
                CALL ZMMCH( 'N', 'N', LJ, 1, 2 * K, ONE, &
                           AB( JJ ), NMAX, W, 2 * NMAX, &
                           BETA, C( JJ, J ), NMAX, CT, &
                           G, CC( JC ), LDC, EPS, ERR, &
                           FATAL, NOUT, .TRUE. )
            END IF
            IF( UPPER )THEN
                JC = JC + LDC
            END IF
        END DO
    END IF

```

```

        ELSE
            JC = JC + LDC + 1
            IF( TRAN )&
                JJAB = JJAB + 2*NMAX
        END IF
        ERRMAX = MAX( ERRMAX, ERR )
        If got really bad answer, report and
        return.
        IF( FATAL )&
            GO TO 140
70      CONTINUE
        END IF
!
80      CONTINUE
!
90      CONTINUE
!
100     CONTINUE
!
110     CONTINUE
!
120     CONTINUE
!
130 CONTINUE
!
        Report result.
!
        IF( ERRMAX.LT.THRESH )THEN
            WRITE( NOUT, FMT = 9999 )SNAME, NC
        ELSE
            WRITE( NOUT, FMT = 9997 )SNAME, NC, ERRMAX
        END IF
        GO TO 160
!
140 CONTINUE
        IF( N.GT.1 )&
            WRITE( NOUT, FMT = 9995 )J
!
150 CONTINUE
        WRITE( NOUT, FMT = 9996 )SNAME
        IF( CONJ )THEN
            WRITE( NOUT, FMT = 9994 )NC, SNAME, UPLO, TRANS, N, K, ALPHA,&
                LDA, LDB, RBETA, LDC
        ELSE
            WRITE( NOUT, FMT = 9993 )NC, SNAME, UPLO, TRANS, N, K, ALPHA,&
                LDA, LDB, BETA, LDC
        END IF
!
160 CONTINUE
        RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE COMPUTATIONAL TESTS (', I6, ' CALL',&
    'S)' )
9998 FORMAT( ' ***** FATAL ERROR - PARAMETER NUMBER ', I2, ' WAS CH',&
    'ANGED INCORRECTLY *****' )
9997 FORMAT( ' ', A6, ' COMPLETED THE COMPUTATIONAL TESTS (', I6, ' C',&
    'ALLS)', /' ***** BUT WITH MAXIMUM TEST RATIO', F8.2,&
    ' - SUSPECT *****' )
9996 FORMAT( ' ***** ', A6, ' FAILED ON CALL NUMBER:' )
9995 FORMAT( ' THESE ARE THE RESULTS FOR COLUMN ', I3 )
9994 FORMAT( 1X, I6, ': ', A6, '(', 2( ' ', A1, ' ', ' ', ' ), 2( I3, ' ', ' ),&
    '(', F4.1, ' ', ' ', F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ', ' ', F4.1,&
    ' ', C, ' ', I3, ' ', ' ' )
9993 FORMAT( 1X, I6, ': ', A6, '(', 2( ' ', A1, ' ', ' ', ' ), 2( I3, ' ', ' ),&
    '(', F4.1, ' ', ' ', F4.1, ' ', A, ' ', I3, ' ', B, ' ', I3, ' ', ' ', F4.1,&
    ' ', ' ', F4.1, ' ', C, ' ', I3, ' ', ' ' )
9992 FORMAT( ' ***** FATAL ERROR - ERROR-EXIT TAKEN ON VALID CALL *',&
    '*****' )
!
        End of ZCHK5.
!
        END SUBROUTINE
        SUBROUTINE ZCHKE( ISNUM, SRNAMT, NOUT )
!
! Tests the error exits from the Level 3 Blas.
! Requires a special version of the error-handling routine XERBLA.
! A, B and C should not need to be defined.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! 3-19-92: Initialize ALPHA, BETA, RALPHA, and RBETA (eca)
! 3-19-92: Fix argument 12 in calls to ZSYMM and ZHEMM
!         with INFOT = 9 (eca)
! 10-9-00: Declared INTRINSIC DCMPLX (susan)
!
! .. Scalar Arguments ..
INTEGER          ISNUM, NOUT
CHARACTER*6      SRNAMT
!
! .. Scalars in Common ..
INTEGER          INFOT, NOUTC
LOGICAL          LERR, OK
!
! .. Parameters ..
REAL             ONE, TWO
PARAMETER        ( ONE = 1.0D0, TWO = 2.0D0 )
!
! .. Local Scalars ..
COMPLEX*16       ALPHA, BETA
DOUBLE PRECISION RALPHA, RBETA
!
! .. Local Arrays ..
COMPLEX*16       A( 2, 1 ), B( 2, 1 ), C( 2, 1 )
!
! .. External Subroutines ..
EXTERNAL          ZGEMM, ZHEMM, ZHER2K, ZHERK, CHKXER, ZSYMM,
$                ZSYR2K, ZSYRK, ZTRMM, ZTRSM
!
! .. Intrinsic Functions ..
INTRINSIC         DCMPLX
!
! .. Common blocks ..
COMMON            /INFOC/INFOT, NOUTC, OK, LERR
!
! .. Executable Statements ..
! OK is set to .FALSE. by the special version of XERBLA or by CHKXER

```

```

! if anything is wrong.
! OK = .TRUE.
! LERR is set to .TRUE. by the special version of XERBLA each time
! it is called, and is then tested and re-set by CHKXER.
! LERR = .FALSE.
!
! Initialize ALPHA, BETA, RALPHA, and RBETA.
!
ALPHA = DCMLPX( ONE, -ONE )
BETA = DCMLPX( TWO, -TWO )
RALPHA = ONE
RBETA = TWO
!
GO TO ( 10, 20, 30, 40, 50, 60, 70, 80, &
      90 ) ISNUM
10 INFOT = 1
CALL ZGEMM( '/', 'N', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 1
CALL ZGEMM( '/', 'C', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 1
CALL ZGEMM( '/', 'T', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZGEMM( 'N', '/', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZGEMM( 'C', '/', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZGEMM( 'T', '/', 0, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'N', 'N', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'N', 'C', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'N', 'T', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'C', 'N', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'C', 'C', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'C', 'T', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'T', 'N', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'T', 'C', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZGEMM( 'T', 'T', -1, 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'N', 'N', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'N', 'C', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'N', 'T', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'C', 'N', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'C', 'C', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'C', 'T', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'T', 'N', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'T', 'C', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZGEMM( 'T', 'T', 0, -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'N', 'N', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'N', 'C', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'N', 'T', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'C', 'N', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'C', 'C', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'C', 'T', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'T', 'N', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'T', 'C', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 5
CALL ZGEMM( 'T', 'T', 0, 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )

```


[illegible]


```

CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZHER2K( 'U', 'C', 0, 2, ALPHA, A, 2, B, 1, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZHER2K( 'L', 'N', 2, 0, ALPHA, A, 2, B, 1, RBETA, C, 2 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZHER2K( 'L', 'C', 0, 2, ALPHA, A, 2, B, 1, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZHER2K( 'U', 'N', 2, 0, ALPHA, A, 2, B, 2, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZHER2K( 'U', 'C', 2, 0, ALPHA, A, 1, B, 1, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZHER2K( 'L', 'N', 2, 0, ALPHA, A, 2, B, 2, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZHER2K( 'L', 'C', 2, 0, ALPHA, A, 1, B, 1, RBETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
GO TO 100
90 INFOT = 1
CALL ZSYR2K( '/', 'N', 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 2
CALL ZSYR2K( 'U', 'C', 0, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZSYR2K( 'U', 'N', -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZSYR2K( 'U', 'T', -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZSYR2K( 'L', 'N', -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 3
CALL ZSYR2K( 'L', 'T', -1, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZSYR2K( 'U', 'N', 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZSYR2K( 'U', 'T', 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZSYR2K( 'L', 'N', 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 4
CALL ZSYR2K( 'L', 'T', 0, -1, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZSYR2K( 'U', 'N', 2, 0, ALPHA, A, 1, B, 1, BETA, C, 2 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZSYR2K( 'U', 'T', 0, 2, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZSYR2K( 'L', 'N', 2, 0, ALPHA, A, 1, B, 1, BETA, C, 2 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 7
CALL ZSYR2K( 'L', 'T', 0, 2, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZSYR2K( 'U', 'N', 2, 0, ALPHA, A, 2, B, 1, BETA, C, 2 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZSYR2K( 'U', 'T', 0, 2, ALPHA, A, 2, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZSYR2K( 'L', 'N', 2, 0, ALPHA, A, 2, B, 1, BETA, C, 2 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 9
CALL ZSYR2K( 'L', 'T', 0, 2, ALPHA, A, 2, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZSYR2K( 'U', 'N', 2, 0, ALPHA, A, 2, B, 2, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZSYR2K( 'U', 'T', 2, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZSYR2K( 'L', 'N', 2, 0, ALPHA, A, 2, B, 2, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
INFOT = 12
CALL ZSYR2K( 'L', 'T', 2, 0, ALPHA, A, 1, B, 1, BETA, C, 1 )
CALL CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
100 IF( OK )THEN
    WRITE( NOUT, FMT = 9999 )SRNAMT
ELSE
    WRITE( NOUT, FMT = 9998 )SRNAMT
END IF
RETURN
!
9999 FORMAT( ' ', A6, ' PASSED THE TESTS OF ERROR-EXITS' )
9998 FORMAT( ' ***** ', A6, ' FAILED THE TESTS OF ERROR-EXITS *****', &
    '***' )
!
! End of ZCHKE.
!
END SUBROUTINE
SUBROUTINE ZMAKE( TYPE, UPLO, DIAG, M, N, A, NMAX, AA, LDA, RESET,&
    TRANSL )
!
! Generates values for an M by N matrix A.
! Stores the values in the array AA in the data structure required
! by the routine, with unwanted elements set to rogue value.
!
! TYPE is 'GE', 'HE', 'SY' or 'TR'.
!
! Auxiliary routine for test program for Level 3 Blas.

```

```

!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
!
! .. Parameters ..
COMPLEX*16      ZERO, ONE
PARAMETER      ( ZERO = ( 0.0D0, 0.0D0 ), &
                ONE = ( 1.0D0, 0.0D0 ) )
COMPLEX*16      ROGUE
PARAMETER      ( ROGUE = ( -1.0D10, 1.0D10 ) )
DOUBLE PRECISION RZERO
PARAMETER      ( RZERO = 0.0D0 )
DOUBLE PRECISION RROGUE
PARAMETER      ( RROGUE = -1.0D10 )
! .. Scalar Arguments ..
COMPLEX*16      TRANSL
INTEGER         LDA, M, N, NMAX
LOGICAL         RESET
CHARACTER*1     DIAG, UPLO
CHARACTER*2     TYPE
! .. Array Arguments ..
COMPLEX*16      A( NMAX, * ), AA( * )
! .. Local Scalars ..
INTEGER         I, IBEG, IEND, J, JJ
LOGICAL         GEN, HER, LOWER, SYM, TRI, UNIT, UPPER
! .. External Functions ..
COMPLEX*16      ZBEG
EXTERNAL        ZBEG
! .. Intrinsic Functions ..
INTRINSIC       DCMLPX, DCONJG, DBLE
! .. Executable Statements ..
GEN = TYPE.EQ.'GE'
HER = TYPE.EQ.'HE'
SYM = TYPE.EQ.'SY'
TRI = TYPE.EQ.'TR'
UPPER = ( HER.OR.SYM.OR.TRI ).AND.UPLO.EQ.'U'
LOWER = ( HER.OR.SYM.OR.TRI ).AND.UPLO.EQ.'L'
UNIT = TRI.AND.DIAG.EQ.'U'
!
! Generate data in array A.
!
DO 20 J = 1, N
  DO 10 I = 1, M
    IF( GEN.OR.( UPPER.AND.I.LE.J ).OR.( LOWER.AND.I.GE.J ) )&
      THEN
      A( I, J ) = ZBEG( RESET ) + TRANSL
      IF( I.NE.J )THEN
        Set some elements to zero
        IF( N.GT.3.AND.J.EQ.N/2 )&
          A( I, J ) = ZERO
        IF( HER )THEN
          A( J, I ) = DCONJG( A( I, J ) )
        ELSE IF( SYM )THEN
          A( J, I ) = A( I, J )
        ELSE IF( TRI )THEN
          A( J, I ) = ZERO
        END IF
      END IF
    END IF
  10 CONTINUE
  IF( HER )&
    A( J, J ) = DCMLPX( DBLE( A( J, J ) ), RZERO )
  IF( TRI )&
    A( J, J ) = A( J, J ) + ONE
  IF( UNIT )&
    A( J, J ) = ONE
20 CONTINUE
!
! Store elements in array AS in data structure required by routine.
!
IF( TYPE.EQ.'GE' )THEN
  DO 50 J = 1, N
    DO 30 I = 1, M
      AA( I + ( J - 1 )*LDA ) = A( I, J )
    30 CONTINUE
    DO 40 I = M + 1, LDA
      AA( I + ( J - 1 )*LDA ) = ROGUE
    40 CONTINUE
  50 CONTINUE
ELSE IF( TYPE.EQ.'HE'.OR.TYPE.EQ.'SY'.OR.TYPE.EQ.'TR' )THEN
  DO 90 J = 1, N
    IF( UPPER )THEN
      IBEG = 1
      IF( UNIT )THEN
        IEND = J - 1
      ELSE
        IEND = J
      END IF
    ELSE
      IF( UNIT )THEN
        IBEG = J + 1
      ELSE
        IBEG = J
      END IF
      IEND = N
    END IF
    DO 60 I = 1, IBEG - 1
      AA( I + ( J - 1 )*LDA ) = ROGUE
    60 CONTINUE
    DO 70 I = IBEG, IEND
      AA( I + ( J - 1 )*LDA ) = A( I, J )
    70 CONTINUE
    DO 80 I = IEND + 1, LDA
      AA( I + ( J - 1 )*LDA ) = ROGUE
    80 CONTINUE
    IF( HER )THEN
      JJ = J + ( J - 1 )*LDA
      AA( JJ ) = DCMLPX( DBLE( AA( JJ ) ), RROGUE )
    END IF
  90 CONTINUE
END IF

```



```

130      CONTINUE
      ELSE
        DO 150 K = 1, KK
          DO 140 I = 1, M
            CT( I ) = CT( I ) + DCONJG( A( K, I ) ) * %
              B( J, K )
            G( I ) = G( I ) + ABS1( A( K, I ) ) * %
              ABS1( B( J, K ) )
          CONTINUE
        CONTINUE
      END IF
      ELSE
        IF( CTRANB ) THEN
          DO 170 K = 1, KK
            DO 160 I = 1, M
              CT( I ) = CT( I ) + A( K, I ) * %
                DCONJG( B( J, K ) )
              G( I ) = G( I ) + ABS1( A( K, I ) ) * %
                ABS1( B( J, K ) )
            CONTINUE
          CONTINUE
        ELSE
          DO 190 K = 1, KK
            DO 180 I = 1, M
              CT( I ) = CT( I ) + A( K, I ) * B( J, K )
              G( I ) = G( I ) + ABS1( A( K, I ) ) * %
                ABS1( B( J, K ) )
            CONTINUE
          CONTINUE
        END IF
      END IF
      DO 200 I = 1, M
        CT( I ) = ALPHA * CT( I ) + BETA * C( I, J )
        G( I ) = ABS1( ALPHA ) * G( I ) + %
          ABS1( BETA ) * ABS1( C( I, J ) )
      CONTINUE
!
!   Compute the error ratio for this result.
!
      ERR = ZERO
      DO 210 I = 1, M
        ERR1 = ABS1( CT( I ) - CC( I, J ) ) / EPS
        IF( G( I ) .NE. RZERO ) %
          ERR1 = ERR1 / G( I )
        ERR = MAX( ERR, ERR1 )
        IF( ERR * SQRT( EPS ) .GE. RONE ) %
          GO TO 230
      CONTINUE
210
!
220 CONTINUE
!
!   If the loop completes, all results are at least half accurate.
GO TO 250
!
!   Report fatal error.
!
230 FATAL = .TRUE.
WRITE( NOUT, FMT = 9999 )
DO 240 I = 1, M
  IF( MV ) THEN
    WRITE( NOUT, FMT = 9998 ) I, CT( I ), CC( I, J )
  ELSE
    WRITE( NOUT, FMT = 9998 ) I, CC( I, J ), CT( I )
  END IF
240 CONTINUE
IF( N.GT.1 ) %
  WRITE( NOUT, FMT = 9997 ) J
!
250 CONTINUE
RETURN
!
9999 FORMAT( ' ***** FATAL ERROR - COMPUTED RESULT IS LESS THAN HAL', &
  'F ACCURATE *****', /, ' EXPECTED RE', &
  'SULT', ' COMPUTED RESULT' )
9998 FORMAT( 1X, I7, 2( ' (', G15.6, ', ', G15.6, ' )' ) )
9997 FORMAT( ' THESE ARE THE RESULTS FOR COLUMN ', I3 )
!
!   End of ZMMCH.
!
END SUBROUTINE
LOGICAL FUNCTION LZE( RI, RJ, LR )
!
!   Tests if two arrays are identical.
!
!   Auxiliary routine for test program for Level 3 Blas.
!
!   -- Written on 8-February-1989.
!   Jack Dongarra, Argonne National Laboratory.
!   Iain Duff, AERE Harwell.
!   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!   Sven Hammarling, Numerical Algorithms Group Ltd.
!
!   .. Scalar Arguments ..
INTEGER    LR
!   .. Array Arguments ..
COMPLEX*16 RI( * ), RJ( * )
!   .. Local Scalars ..
INTEGER    I
!   .. Executable Statements ..
DO 10 I = 1, LR
  IF( RI( I ) .NE. RJ( I ) ) %
    GO TO 20
10 CONTINUE
LZE = .TRUE.
GO TO 30
20 CONTINUE
LZE = .FALSE.
30 RETURN
!
!   End of LZE.
!
END FUNCTION
LOGICAL FUNCTION LZERES( TYPE, UPLO, M, N, AA, AS, LDA )

```

```

!
! Tests if selected elements in two arrays are equal.
!
! TYPE is 'GE' or 'HE' or 'SY'.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
INTEGER          LDA, M, N
CHARACTER*1      UPLO
CHARACTER*2      TYPE
! .. Array Arguments ..
COMPLEX*16       AA( LDA, * ), AS( LDA, * )
! .. Local Scalars ..
INTEGER          I, IBEG, IEND, J
LOGICAL          UPPER
! .. Executable Statements ..
UPPER = UPLO.EQ.'U'
IF( TYPE.EQ.'GE' )THEN
  DO 20 J = 1, N
    DO 10 I = M + 1, LDA
      IF( AA( I, J ).NE.AS( I, J ) )&
        GO TO 70
    CONTINUE
  10 CONTINUE
20 CONTINUE
ELSE IF( TYPE.EQ.'HE'.OR.TYPE.EQ.'SY' )THEN
  DO 50 J = 1, N
    IF( UPPER )THEN
      IBEG = 1
      IEND = J
    ELSE
      IBEG = J
      IEND = N
    END IF
    DO 30 I = 1, IBEG - 1
      IF( AA( I, J ).NE.AS( I, J ) )&
        GO TO 70
    CONTINUE
  30 CONTINUE
  DO 40 I = IEND + 1, LDA
    IF( AA( I, J ).NE.AS( I, J ) )&
      GO TO 70
  40 CONTINUE
50 CONTINUE
END IF
!
LZERES = .TRUE.
GO TO 80
70 CONTINUE
LZERES = .FALSE.
80 RETURN
!
! End of LZERES.
!
!
! END FUNCTION
COMPLEX*16       FUNCTION ZBEG( RESET )
!
! Generates complex numbers as pairs of random numbers uniformly
! distributed between -0.5 and 0.5.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
LOGICAL          RESET
! .. Local Scalars ..
INTEGER          I, IC, J, MI, MJ
! .. Save statement ..
SAVE             I, IC, J, MI, MJ
! .. Intrinsic Functions ..
INTRINSIC        DCMLPX
! .. Executable Statements ..
IF( RESET )THEN
  Initialize local variables.
  MI = 891
  MJ = 457
  I = 7
  J = 7
  IC = 0
  RESET = .FALSE.
END IF
!
! The sequence of values of I or J is bounded between 1 and 999.
! If initial I or J = 1,2,3,6,7 or 9, the period will be 50.
! If initial I or J = 4 or 8, the period will be 25.
! If initial I or J = 5, the period will be 10.
! IC is used to break up the period by skipping 1 value of I or J
! in 6.
!
! IC = IC + 1
10 I = I*MI
J = J*MJ
I = I - 1000*( I/1000 )
J = J - 1000*( J/1000 )
IF( IC.GE.5 )THEN
  IC = 0
  GO TO 10
END IF
ZBEG = DCMLPX( ( I - 500 )/1001.0D0, ( J - 500 )/1001.0D0 )
RETURN
!
! End of ZBEG.
!
! END FUNCTION

```

```

      DOUBLE PRECISION FUNCTION DDIFF( X, Y )
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
! DOUBLE PRECISION X, Y
! .. Executable Statements ..
! DDIFF = X - Y
! RETURN
!
! End of DDIFF.
!
! END FUNCTION
! SUBROUTINE CHKXER( SRNAMT, INFOT, NOUT, LERR, OK )
!
! Tests whether XERBLA has detected an error when it should.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
! INTEGER INFOT, NOUT
! LOGICAL LERR, OK
! CHARACTER*6 SRNAMT
! .. Executable Statements ..
! IF( .NOT.LERR )THEN
!   WRITE( NOUT, FMT = 9999 )INFOT, SRNAMT
!   OK = .FALSE.
! END IF
! LERR = .FALSE.
! RETURN
!
! 9999 FORMAT( ' ***** ILLEGAL VALUE OF PARAMETER NUMBER ', I2, ' NOT D', &
!   'ETECTED BY ', A6, ' *****' )
!
! End of CHKXER.
!
! END SUBROUTINE
! SUBROUTINE XERBLA( SRNAME, INFO )
!
! This is a special version of XERBLA to be used only as part of
! the test program for testing error exits from the Level 3 BLAS
! routines.
!
! XERBLA is an error handler for the Level 3 BLAS routines.
!
! It is called by the Level 3 BLAS routines if an input parameter is
! invalid.
!
! Auxiliary routine for test program for Level 3 Blas.
!
! -- Written on 8-February-1989.
! Jack Dongarra, Argonne National Laboratory.
! Iain Duff, AERE Harwell.
! Jeremy Du Croz, Numerical Algorithms Group Ltd.
! Sven Hammarling, Numerical Algorithms Group Ltd.
!
! .. Scalar Arguments ..
! INTEGER INFO
! CHARACTER*6 SRNAME
! .. Scalars in Common ..
! .. Common blocks ..
! COMMON /INFOC/INFOT, NOUT, OK, LERR
! COMMON /SRNAMC/SRNAMT
! .. Executable Statements ..
! LERR = .TRUE.
! IF( INFO.NE.INFOT )THEN
!   IF( INFOT.NE.0 )THEN
!     WRITE( NOUT, FMT = 9999 )INFO, INFOT
!   ELSE
!     WRITE( NOUT, FMT = 9997 )INFO
!   END IF
!   OK = .FALSE.
! END IF
! IF( SRNAME.NE.SRNAMT )THEN
!   WRITE( NOUT, FMT = 9998 )SRNAME, SRNAMT
!   OK = .FALSE.
! END IF
! RETURN
!
! 9999 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, ' INSTEAD', &
!   ' OF ', I2, ' *****' )
! 9998 FORMAT( ' ***** XERBLA WAS CALLED WITH SRNAME = ', A6, ' INSTE', &
!   'AD OF ', A6, ' *****' )
! 9997 FORMAT( ' ***** XERBLA WAS CALLED WITH INFO = ', I6, &
!   ' *****' )
!
! End of XERBLA
!
! END SUBROUTINE
!> \brief \b ZGEMM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
!

```

```

!      SUBROUTINE ZGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 ALPHA,BETA
!      INTEGER K,LDA,LDB,LDC,M,N
!      CHARACTER TRANSA,TRANSB
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
!      ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!>
!> ZGEMM performs one of the matrix-matrix operations
!>
!>   C := alpha*op( A )*op( B ) + beta*C,
!>
!> where op( X ) is one of
!>
!>   op( X ) = X   or   op( X ) = X**T   or   op( X ) = X**H,
!>
!> alpha and beta are scalars, and A, B and C are matrices, with op( A )
!> an m by k matrix, op( B ) a k by n matrix and C an m by n matrix.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] TRANSA
!> \verbatim
!>   TRANSA is CHARACTER*1
!>   On entry, TRANSA specifies the form of op( A ) to be used in
!>   the matrix multiplication as follows:
!>
!>       TRANSA = 'N' or 'n',   op( A ) = A.
!>
!>       TRANSA = 'T' or 't',   op( A ) = A**T.
!>
!>       TRANSA = 'C' or 'c',   op( A ) = A**H.
!> \endverbatim
!> \param[in] TRANSB
!> \verbatim
!>   TRANSB is CHARACTER*1
!>   On entry, TRANSB specifies the form of op( B ) to be used in
!>   the matrix multiplication as follows:
!>
!>       TRANSB = 'N' or 'n',   op( B ) = B.
!>
!>       TRANSB = 'T' or 't',   op( B ) = B**T.
!>
!>       TRANSB = 'C' or 'c',   op( B ) = B**H.
!> \endverbatim
!> \param[in] M
!> \verbatim
!>   M is INTEGER
!>   On entry, M specifies the number of rows of the matrix
!>   op( A ) and of the matrix C. M must be at least zero.
!> \endverbatim
!> \param[in] N
!> \verbatim
!>   N is INTEGER
!>   On entry, N specifies the number of columns of the matrix
!>   op( B ) and the number of columns of the matrix C. N must be
!>   at least zero.
!> \endverbatim
!> \param[in] K
!> \verbatim
!>   K is INTEGER
!>   On entry, K specifies the number of columns of the matrix
!>   op( A ) and the number of rows of the matrix op( B ). K must
!>   be at least zero.
!> \endverbatim
!> \param[in] ALPHA
!> \verbatim
!>   ALPHA is COMPLEX*16
!>   On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!> \param[in] A
!> \verbatim
!>   A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!>   k when TRANSA = 'N' or 'n', and is m otherwise.
!>   Before entry with TRANSA = 'N' or 'n', the leading m by k
!>   part of the array A must contain the matrix A, otherwise
!>   the leading k by m part of the array A must contain the
!>   matrix A.
!> \endverbatim
!> \param[in] LDA
!> \verbatim
!>   LDA is INTEGER
!>   On entry, LDA specifies the first dimension of A as declared
!>   in the calling (sub) program. When TRANSA = 'N' or 'n' then
!>   LDA must be at least max( 1, m ), otherwise LDA must be at
!>   least max( 1, k ).
!> \endverbatim
!> \param[in] B
!> \verbatim
!>   B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is
!>   n when TRANSB = 'N' or 'n', and is k otherwise.
!>   Before entry with TRANSB = 'N' or 'n', the leading k by n
!>   part of the array B must contain the matrix B, otherwise
!>   the leading n by k part of the array B must contain the
!>   matrix B.

```

```

!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. When TRANSB = 'N' or 'n' then
!>     LDB must be at least max( 1, k ), otherwise LDB must be at
!>     least max( 1, n ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then C need not be set on input.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>     C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>     Before entry, the leading m by n part of the array C must
!>     contain the matrix C, except when beta is zero, in which
!>     case C need not be set on entry.
!>     On exit, the array C is overwritten by the m by n matrix
!>     ( alpha*op( A )*op( B ) + beta*C ).
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>     LDC is INTEGER
!>     On entry, LDC specifies the first dimension of C as declared
!>     in the calling (sub) program. LDC must be at least
!>     max( 1, m ).
!> \endverbatim
!
! Authors:
! =====
!
! \author Univ. of Tennessee
! \author Univ. of California Berkeley
! \author Univ. of Colorado Denver
! \author NAG Ltd.
!
! \date November 2011
!
! \ingroup complex16_blas_level3
!
! \par Further Details:
! =====
!
! \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
! =====
! SUBROUTINE ZGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER K,LDA,LDB,LDC,M,N
! CHARACTER TRANSA,TRANSB
!
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
! ..
!
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,J,L,NCOLA,NROWA,NROWB
! LOGICAL CONJA,CONJB,NOTA,NOTB
! ..
! .. Parameters ..
! COMPLEX*16 ONE
! PARAMETER (ONE= (1.0D+0,0.0D+0))
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
!
! Set NOTA and NOTB as true if A and B respectively are not
! conjugated or transposed, set CONJA and CONJB as true if A and
! B respectively are to be transposed but not conjugated and set
! NROWA, NCOLA and NROWB as the number of rows and columns of A
! and the number of rows of B respectively.
!
! NOTA = LSAME(TRANSA,'N')
! NOTB = LSAME(TRANSB,'N')
! CONJA = LSAME(TRANSA,'C')
! CONJB = LSAME(TRANSB,'C')
! IF (NOTA) THEN

```

```

        NROWA = M
        NCOLA = K
    ELSE
        NROWA = K
        NCOLA = M
    END IF
    IF (NOTB) THEN
        NROWB = K
    ELSE
        NROWB = N
    END IF
!
!   Test the input parameters.
!
    INFO = 0
    IF ((.NOT.NOTA) .AND. (.NOT.CONJA) .AND. &
        (.NOT.LSAME(TRANSA,'T')) THEN
        INFO = 1
    ELSE IF ((.NOT.NOTB) .AND. (.NOT.CONJB) .AND. &
        (.NOT.LSAME(TRANSB,'T')) THEN
        INFO = 2
    ELSE IF (M.LT.0) THEN
        INFO = 3
    ELSE IF (N.LT.0) THEN
        INFO = 4
    ELSE IF (K.LT.0) THEN
        INFO = 5
    ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
        INFO = 8
    ELSE IF (LDB.LT.MAX(1,NROWB)) THEN
        INFO = 10
    ELSE IF (LDC.LT.MAX(1,M)) THEN
        INFO = 13
    END IF
    IF (INFO.NE.0) THEN
        CALL XERBLA('ZGEMM ',INFO)
        RETURN
    END IF
!
!   Quick return if possible.
!
    IF ((M.EQ.0) .OR. (N.EQ.0) .OR. &
        ((ALPHA.EQ.ZERO) .OR. (K.EQ.0)) .AND. (BETA.EQ.ONE)) RETURN
!
!   And when alpha.eq.zero.
!
    IF (ALPHA.EQ.ZERO) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 20 J = 1,N
                DO 10 I = 1,M
                    C(I,J) = ZERO
                CONTINUE
            10 CONTINUE
            20 CONTINUE
        ELSE
            DO 40 J = 1,N
                DO 30 I = 1,M
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            30 CONTINUE
            40 CONTINUE
        END IF
        RETURN
    END IF
!
!   Start the operations.
!
    IF (NOTB) THEN
        IF (NOTA) THEN
!
!           Form C := alpha*A*B + beta*C.
!
            DO 90 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 50 I = 1,M
                        C(I,J) = ZERO
                    CONTINUE
                50 ELSE IF (BETA.NE.ONE) THEN
                    DO 60 I = 1,M
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                60 END IF
                DO 80 L = 1,K
                    IF (B(L,J).NE.ZERO) THEN
                        TEMP = ALPHA*B(L,J)
                        DO 70 I = 1,M
                            C(I,J) = C(I,J) + TEMP*A(I,L)
                        70 CONTINUE
                    END IF
                80 CONTINUE
                90 CONTINUE
            ELSE IF (CONJA) THEN
!
!           Form C := alpha*A**H*B + beta*C.
!
                DO 120 J = 1,N
                    DO 110 I = 1,M
                        TEMP = ZERO
                        DO 100 L = 1,K
                            TEMP = TEMP + DCONJG(A(L,I))*B(L,J)
                        100 CONTINUE
                        IF (BETA.EQ.ZERO) THEN
                            C(I,J) = ALPHA*TEMP
                        ELSE
                            C(I,J) = ALPHA*TEMP + BETA*C(I,J)
                        END IF
                    110 CONTINUE
                    120 CONTINUE
                ELSE
!
!           Form C := alpha*A**T*B + beta*C
!
                    DO 150 J = 1,N
                        DO 140 I = 1,M
                            TEMP = ZERO
                            DO 130 L = 1,K

```

```

130         TEMP = TEMP + A(L,I)*B(L,J)
            CONTINUE
            IF (BETA.EQ.ZERO) THEN
                C(I,J) = ALPHA*TEMP
            ELSE
                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
            END IF
140         CONTINUE
150     END IF
    ELSE IF (NOTA) THEN
        IF (CONJB) THEN
!
!           Form C := alpha*A*B**H + beta*C.
!
            DO 200 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 160 I = 1,M
                        C(I,J) = ZERO
160                 CONTINUE
                    ELSE IF (BETA.NE.ONE) THEN
                        DO 170 I = 1,M
                            C(I,J) = BETA*C(I,J)
170                 CONTINUE
                        END IF
                    DO 190 L = 1,K
                        IF (B(J,L).NE.ZERO) THEN
                            TEMP = ALPHA*DCONJG(B(J,L))
                            DO 180 I = 1,M
                                C(I,J) = C(I,J) + TEMP*A(I,L)
180                             CONTINUE
                            END IF
                        END IF
                    CONTINUE
                ELSE
190                 CONTINUE
200             END IF
!
!           Form C := alpha*A*B**T          + beta*C
!
            DO 250 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 210 I = 1,M
                        C(I,J) = ZERO
210                 CONTINUE
                    ELSE IF (BETA.NE.ONE) THEN
                        DO 220 I = 1,M
                            C(I,J) = BETA*C(I,J)
220                 CONTINUE
                        END IF
                    DO 240 L = 1,K
                        IF (B(J,L).NE.ZERO) THEN
                            TEMP = ALPHA*B(J,L)
                            DO 230 I = 1,M
                                C(I,J) = C(I,J) + TEMP*A(I,L)
230                             CONTINUE
                            END IF
                        END IF
                    CONTINUE
                ELSE
240                 CONTINUE
250             END IF
            ELSE IF (CONJA) THEN
                IF (CONJB) THEN
!
!           Form C := alpha*A**H*B**H + beta*C.
!
                    DO 280 J = 1,N
                        DO 270 I = 1,M
                            TEMP = ZERO
                            DO 260 L = 1,K
                                TEMP = TEMP + DCONJG(A(L,I))*DCONJG(B(J,L))
260                             CONTINUE
                            IF (BETA.EQ.ZERO) THEN
                                C(I,J) = ALPHA*TEMP
                            ELSE
                                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
                            END IF
                        END IF
                    CONTINUE
270                 CONTINUE
280             END IF
!
!           Form C := alpha*A**H*B**T + beta*C
!
                    DO 310 J = 1,N
                        DO 300 I = 1,M
                            TEMP = ZERO
                            DO 290 L = 1,K
                                TEMP = TEMP + DCONJG(A(L,I))*B(J,L)
290                             CONTINUE
                            IF (BETA.EQ.ZERO) THEN
                                C(I,J) = ALPHA*TEMP
                            ELSE
                                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
                            END IF
                        END IF
                    CONTINUE
300                 CONTINUE
310             END IF
            ELSE
                IF (CONJB) THEN
!
!           Form C := alpha*A**T*B**H + beta*C
!
                    DO 340 J = 1,N
                        DO 330 I = 1,M
                            TEMP = ZERO
                            DO 320 L = 1,K
                                TEMP = TEMP + A(L,I)*DCONJG(B(J,L))
320                             CONTINUE
                            IF (BETA.EQ.ZERO) THEN
                                C(I,J) = ALPHA*TEMP
                            ELSE
                                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
                            END IF
                        END IF
                    CONTINUE
330                 CONTINUE
340             END IF
            ELSE
!

```

```

!      Form C := alpha*A**T*B**T + beta*C
!
      DO 370 J = 1,N
      DO 360 I = 1,M
      TEMP = ZERO
      DO 350 L = 1,K
      TEMP = TEMP + A(L,I)*B(J,L)
350      CONTINUE
      IF (BETA.EQ.ZERO) THEN
      C(I,J) = ALPHA*TEMP
      ELSE
      C(I,J) = ALPHA*TEMP + BETA*C(I,J)
      END IF
360      CONTINUE
370      CONTINUE
      END IF
!
      RETURN
!
!      End of ZGEMM .
!
      END SUBROUTINE

!> \brief \b ZHEMM
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE ZHEMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 ALPHA,BETA
!      INTEGER LDA,LDB,LDC,M,N
!      CHARACTER SIDE,UPLO
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
!      ..
!
!> \par Purpose:
!      =====
!>
!> \verbatim
!> ZHEMM performs one of the matrix-matrix operations
!>
!>      C := alpha*A*B + beta*C,
!>
!> or
!>
!>      C := alpha*B*A + beta*C,
!>
!> where alpha and beta are scalars, A is an hermitian matrix and B and
!> C are m by n matrices.
!> \endverbatim
!
!      Arguments:
!      =====
!
!> \param[in] SIDE
!> \verbatim
!>      SIDE is CHARACTER*1
!>      On entry, SIDE specifies whether the hermitian matrix A
!>      appears on the left or right in the operation as follows:
!>
!>          SIDE = 'L' or 'l'  C := alpha*A*B + beta*C,
!>
!>          SIDE = 'R' or 'r'  C := alpha*B*A + beta*C,
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!>      UPLO is CHARACTER*1
!>      On entry, UPLO specifies whether the upper or lower
!>      triangular part of the hermitian matrix A is to be
!>      referenced as follows:
!>
!>          UPLO = 'U' or 'u'  Only the upper triangular part of the
!>          hermitian matrix is to be referenced.
!>
!>          UPLO = 'L' or 'l'  Only the lower triangular part of the
!>          hermitian matrix is to be referenced.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!>      M is INTEGER
!>      On entry, M specifies the number of rows of the matrix C.
!>      M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>      N is INTEGER
!>      On entry, N specifies the number of columns of the matrix C.
!>      N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>      ALPHA is COMPLEX*16
!>      On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>      A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is

```

```

!>      m when SIDE = 'L' or 'l' and is n otherwise.
!>      Before entry with SIDE = 'L' or 'l', the m by m part of
!>      the array A must contain the hermitian matrix, such that
!>      when UPLO = 'U' or 'u', the leading m by m upper triangular
!>      part of the array A must contain the upper triangular part
!>      of the hermitian matrix and the strictly lower triangular
!>      part of A is not referenced, and when UPLO = 'L' or 'l',
!>      the leading m by m lower triangular part of the array A
!>      must contain the lower triangular part of the hermitian
!>      matrix and the strictly upper triangular part of A is not
!>      referenced.
!>      Before entry with SIDE = 'R' or 'r', the n by n part of
!>      the array A must contain the hermitian matrix, such that
!>      when UPLO = 'U' or 'u', the leading n by n upper triangular
!>      part of the array A must contain the upper triangular part
!>      of the hermitian matrix and the strictly lower triangular
!>      part of A is not referenced, and when UPLO = 'L' or 'l',
!>      the leading n by n lower triangular part of the array A
!>      must contain the lower triangular part of the hermitian
!>      matrix and the strictly upper triangular part of A is not
!>      referenced.
!>      Note that the imaginary parts of the diagonal elements need
!>      not be set, they are assumed to be zero.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>      LDA is INTEGER
!>      On entry, LDA specifies the first dimension of A as declared
!>      in the calling (sub) program. When SIDE = 'L' or 'l' then
!>      LDA must be at least max( 1, m ), otherwise LDA must be at
!>      least max( 1, n ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!>      B is COMPLEX*16 array of DIMENSION ( LDB, n ).
!>      Before entry, the leading m by n part of the array B must
!>      contain the matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>      LDB is INTEGER
!>      On entry, LDB specifies the first dimension of B as declared
!>      in the calling (sub) program. LDB must be at least
!>      max( 1, m ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>      BETA is COMPLEX*16
!>      On entry, BETA specifies the scalar beta. When BETA is
!>      supplied as zero then C need not be set on input.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>      C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>      Before entry, the leading m by n part of the array C must
!>      contain the matrix C, except when beta is zero, in which
!>      case C need not be set on entry.
!>      On exit, the array C is overwritten by the m by n updated
!>      matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>      LDC is INTEGER
!>      On entry, LDC specifies the first dimension of C as declared
!>      in the calling (sub) program. LDC must be at least
!>      max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level3
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>      Level 3 Blas routine.
!>
!>      -- Written on 8-February-1989.
!>      Jack Dongarra, Argonne National Laboratory.
!>      Iain Duff, AERE Harwell.
!>      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>      Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!> =====
!>      SUBROUTINE ZHEMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!>
!>      -- Reference BLAS level3 routine (version 3.4.0) --
!>      -- Reference BLAS is a software package provided by Univ. of Tennessee, --
!>      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!>      November 2011
!>
!>      .. Scalar Arguments ..
!>      COMPLEX*16 ALPHA,BETA
!>      INTEGER LDA,LDB,LDC,M,N
!>      CHARACTER SIDE,UPLO
!>
!>      ..
!>      .. Array Arguments ..

```

```

COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
..
=====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC DBLE,DCONJG,MAX
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP1,TEMP2
! INTEGER I,INFO,J,K,NROWA
! LOGICAL UPPER
! ..
! .. Parameters ..
! COMPLEX*16 ONE
! PARAMETER (ONE= (1.0D+0,0.0D+0))
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
!
! Set NROWA as the number of rows of A.
!
! IF (LSAME(SIDE,'L')) THEN
!   NROWA = M
! ELSE
!   NROWA = N
! END IF
! UPPER = LSAME(UPLO,'U')
!
! Test the input parameters.
!
! INFO = 0
! IF ((.NOT.LSAME(SIDE,'L')) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!   INFO = 1
! ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!   INFO = 2
! ELSE IF (M.LT.0) THEN
!   INFO = 3
! ELSE IF (N.LT.0) THEN
!   INFO = 4
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!   INFO = 7
! ELSE IF (LDB.LT.MAX(1,M)) THEN
!   INFO = 9
! ELSE IF (LDC.LT.MAX(1,M)) THEN
!   INFO = 12
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZHEMM ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF ((M.EQ.0) .OR. (N.EQ.0) .OR. &
!   ((ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN
!
! And when alpha.eq.zero.
!
! IF (ALPHA.EQ.ZERO) THEN
!   IF (BETA.EQ.ZERO) THEN
!     DO 20 J = 1,N
!       DO 10 I = 1,M
!         C(I,J) = ZERO
!       CONTINUE
!     CONTINUE
!   ELSE
!     DO 40 J = 1,N
!       DO 30 I = 1,M
!         C(I,J) = BETA*C(I,J)
!       CONTINUE
!     CONTINUE
!   END IF
!   RETURN
! END IF
!
! Start the operations.
!
! IF (LSAME(SIDE,'L')) THEN
!
!   Form C := alpha*A*B + beta*C.
!
!   IF (UPPER) THEN
!     DO 70 J = 1,N
!       DO 60 I = 1,M
!         TEMP1 = ALPHA*B(I,J)
!         TEMP2 = ZERO
!         DO 50 K = 1,I - 1
!           C(K,J) = C(K,J) + TEMP1*A(K,I)
!           TEMP2 = TEMP2 + B(K,J)*DCONJG(A(K,I))
!         CONTINUE
!         IF (BETA.EQ.ZERO) THEN
!           C(I,J) = TEMP1*DBLE(A(I,I)) + ALPHA*TEMP2
!         ELSE
!           C(I,J) = BETA*C(I,J) + TEMP1*DBLE(A(I,I)) +&
!             ALPHA*TEMP2
!         END IF
!       CONTINUE
!     CONTINUE
!   ELSE
!     DO 100 J = 1,N
!       DO 90 I = M,1,-1
!         TEMP1 = ALPHA*B(I,J)
!         TEMP2 = ZERO
!         DO 80 K = I + 1,M
!           C(K,J) = C(K,J) + TEMP1*A(K,I)
!           TEMP2 = TEMP2 + B(K,J)*DCONJG(A(K,I))

```

```

80          CONTINUE
          IF (BETA.EQ.ZERO) THEN
            C(I,J) = TEMP1*DBLE(A(I,I)) + ALPHA*TEMP2
          ELSE
            C(I,J) = BETA*C(I,J) + TEMP1*DBLE(A(I,I)) +&
              ALPHA*TEMP2
          END IF
90        CONTINUE
100      CONTINUE
      END IF
    ELSE
!      Form C := alpha*B*A + beta*C.
!
      DO 170 J = 1,N
        TEMP1 = ALPHA*DBLE(A(J,J))
        IF (BETA.EQ.ZERO) THEN
          DO 110 I = 1,M
            C(I,J) = TEMP1*B(I,J)
110          CONTINUE
        ELSE
          DO 120 I = 1,M
            C(I,J) = BETA*C(I,J) + TEMP1*B(I,J)
120          CONTINUE
        END IF
        DO 140 K = 1,J - 1
          IF (UPPER) THEN
            TEMP1 = ALPHA*A(K,J)
          ELSE
            TEMP1 = ALPHA*DCONJG(A(J,K))
          END IF
          DO 130 I = 1,M
            C(I,J) = C(I,J) + TEMP1*B(I,K)
130          CONTINUE
140          CONTINUE
          DO 160 K = J + 1,N
            IF (UPPER) THEN
              TEMP1 = ALPHA*DCONJG(A(J,K))
            ELSE
              TEMP1 = ALPHA*A(K,J)
            END IF
            DO 150 I = 1,M
              C(I,J) = C(I,J) + TEMP1*B(I,K)
150            CONTINUE
160            CONTINUE
170          CONTINUE
        END IF
      RETURN
!
!      End of ZHEMM .
!
      END SUBROUTINE
!> \brief \b ZSYMM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER LDA,LDB,LDC,M,N
! CHARACTER SIDE,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
! ..
!
!> \par Purpose:
! =====
!
!> \verbatim
!>
!> ZSYMM performs one of the matrix-matrix operations
!>
!>   C := alpha*A*B + beta*C,
!>
!> or
!>
!>   C := alpha*B*A + beta*C,
!>
!> where alpha and beta are scalars, A is a symmetric matrix and B and
!> C are m by n matrices.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] SIDE
!> \verbatim
!>
!>   SIDE is CHARACTER*1
!>   On entry, SIDE specifies whether the symmetric matrix A
!>   appears on the left or right in the operation as follows:
!>
!>     SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,
!>
!>     SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!>
!>   UPLO is CHARACTER*1
!>   On entry, UPLO specifies whether the upper or lower
!>   triangular part of the symmetric matrix A is to be
!>   referenced as follows:
!>
!>     UPLO = 'U' or 'u'   Only the upper triangular part of the

```

```

!>                                     symmetric matrix is to be referenced.
!>
!>         UPLO = 'L' or 'l'   Only the lower triangular part of the
!>                               symmetric matrix is to be referenced.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!>     M is INTEGER
!>     On entry, M specifies the number of rows of the matrix C.
!>     M must be at least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the number of columns of the matrix C.
!>     N must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!>     m when SIDE = 'L' or 'l' and is n otherwise.
!>     Before entry with SIDE = 'L' or 'l', the m by m part of
!>     the array A must contain the symmetric matrix, such that
!>     when UPLO = 'U' or 'u', the leading m by m upper triangular
!>     part of the array A must contain the upper triangular part
!>     of the symmetric matrix and the strictly lower triangular
!>     part of A is not referenced, and when UPLO = 'L' or 'l',
!>     the leading m by m lower triangular part of the array A
!>     must contain the lower triangular part of the symmetric
!>     matrix and the strictly upper triangular part of A is not
!>     referenced.
!>     Before entry with SIDE = 'R' or 'r', the n by n part of
!>     the array A must contain the symmetric matrix, such that
!>     when UPLO = 'U' or 'u', the leading n by n upper triangular
!>     part of the array A must contain the upper triangular part
!>     of the symmetric matrix and the strictly lower triangular
!>     part of A is not referenced, and when UPLO = 'L' or 'l',
!>     the leading n by n lower triangular part of the array A
!>     must contain the lower triangular part of the symmetric
!>     matrix and the strictly upper triangular part of A is not
!>     referenced.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When SIDE = 'L' or 'l' then
!>     LDA must be at least max( 1, m ), otherwise LDA must be at
!>     least max( 1, n ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!>     B is COMPLEX*16 array of DIMENSION ( LDB, n ).
!>     Before entry, the leading m by n part of the array B must
!>     contain the matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. LDB must be at least
!>     max( 1, m ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta. When BETA is
!>     supplied as zero then C need not be set on input.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>     C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>     Before entry, the leading m by n part of the array C must
!>     contain the matrix C, except when beta is zero, in which
!>     case C need not be set on entry.
!>     On exit, the array C is overwritten by the m by n updated
!>     matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>     LDC is INTEGER
!>     On entry, LDC specifies the first dimension of C as declared
!>     in the calling (sub) program. LDC must be at least
!>     max( 1, m ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level3
!>
!> \par Further Details:

```

```

! =====
!>
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
! =====
      SUBROUTINE ZSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
!   November 2011
!
! .. Scalar Arguments ..
      COMPLEX*16 ALPHA,BETA
      INTEGER LDA,LDB,LDC,M,N
      CHARACTER SIDE,UPLO
!
! .. Array Arguments ..
      COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
!
! =====
!
! .. External Functions ..
      LOGICAL LSAME
      EXTERNAL LSAME
!
! .. External Subroutines ..
      EXTERNAL XERBLA
!
! .. Intrinsic Functions ..
      INTRINSIC MAX
!
! .. Local Scalars ..
      COMPLEX*16 TEMP1,TEMP2
      INTEGER I,INFO,J,K,NROWA
      LOGICAL UPPER
!
! .. Parameters ..
      COMPLEX*16 ONE
      PARAMETER (ONE= (1.0D+0,0.0D+0))
      COMPLEX*16 ZERO
      PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
!
! Set NROWA as the number of rows of A.
!
      IF (LSAME(SIDE,'L')) THEN
         NROWA = M
      ELSE
         NROWA = N
      END IF
      UPPER = LSAME(UPLO,'U')
!
! Test the input parameters.
!
      INFO = 0
      IF ((.NOT.LSAME(SIDE,'L')) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
         INFO = 1
      ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
         INFO = 2
      ELSE IF (M.LT.0) THEN
         INFO = 3
      ELSE IF (N.LT.0) THEN
         INFO = 4
      ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
         INFO = 7
      ELSE IF (LDB.LT.MAX(1,M)) THEN
         INFO = 9
      ELSE IF (LDC.LT.MAX(1,M)) THEN
         INFO = 12
      END IF
      IF (INFO.NE.0) THEN
         CALL XERBLA('ZSYMM ',INFO)
         RETURN
      END IF
!
! Quick return if possible.
!
      IF ((M.EQ.0) .OR. (N.EQ.0) .OR. &
         (ALPHA.EQ.ZERO) .AND. (BETA.EQ.ONE))) RETURN
!
! And when alpha.eq.zero.
!
      IF (ALPHA.EQ.ZERO) THEN
         IF (BETA.EQ.ZERO) THEN
            DO 20 J = 1,N
               DO 10 I = 1,M
                  C(I,J) = ZERO
               CONTINUE
            CONTINUE
         ELSE
            DO 40 J = 1,N
               DO 30 I = 1,M
                  C(I,J) = BETA*C(I,J)
               CONTINUE
            CONTINUE
         END IF
         RETURN
      END IF
!
! Start the operations.
!
      IF (LSAME(SIDE,'L')) THEN

```

```

!      Form C := alpha*A*B + beta*C.
!
!      IF (UPPER) THEN
!        DO 70 J = 1,N
!          DO 60 I = 1,M
!            TEMP1 = ALPHA*B(I,J)
!            TEMP2 = ZERO
!            DO 50 K = 1, I - 1
!              C(K,J) = C(K,J) + TEMP1*A(K,I)
!              TEMP2 = TEMP2 + B(K,J)*A(K,I)
50          CONTINUE
!            IF (BETA.EQ.ZERO) THEN
!              C(I,J) = TEMP1*A(I,I) + ALPHA*TEMP2
!            ELSE
!              C(I,J) = BETA*C(I,J) + TEMP1*A(I,I) +&
!                ALPHA*TEMP2
!            END IF
!          CONTINUE
60        CONTINUE
70      CONTINUE
!      ELSE
!        DO 100 J = 1,N
!          DO 90 I = M,1,-1
!            TEMP1 = ALPHA*B(I,J)
!            TEMP2 = ZERO
!            DO 80 K = I + 1,M
!              C(K,J) = C(K,J) + TEMP1*A(K,I)
!              TEMP2 = TEMP2 + B(K,J)*A(K,I)
80          CONTINUE
!            IF (BETA.EQ.ZERO) THEN
!              C(I,J) = TEMP1*A(I,I) + ALPHA*TEMP2
!            ELSE
!              C(I,J) = BETA*C(I,J) + TEMP1*A(I,I) +&
!                ALPHA*TEMP2
!            END IF
!          CONTINUE
90        CONTINUE
100      CONTINUE
!      END IF
!      ELSE
!        Form C := alpha*B*A + beta*C.
!
!        DO 170 J = 1,N
!          TEMP1 = ALPHA*A(J,J)
!          IF (BETA.EQ.ZERO) THEN
!            DO 110 I = 1,M
!              C(I,J) = TEMP1*B(I,J)
110          CONTINUE
!          ELSE
!            DO 120 I = 1,M
!              C(I,J) = BETA*C(I,J) + TEMP1*B(I,J)
120          CONTINUE
!          END IF
!          DO 140 K = 1, J - 1
!            IF (UPPER) THEN
!              TEMP1 = ALPHA*A(K,J)
!            ELSE
!              TEMP1 = ALPHA*A(J,K)
!            END IF
!            DO 130 I = 1,M
!              C(I,J) = C(I,J) + TEMP1*B(I,K)
130          CONTINUE
140          CONTINUE
!          DO 160 K = J + 1,N
!            IF (UPPER) THEN
!              TEMP1 = ALPHA*A(J,K)
!            ELSE
!              TEMP1 = ALPHA*A(K,J)
!            END IF
!            DO 150 I = 1,M
!              C(I,J) = C(I,J) + TEMP1*B(I,K)
150          CONTINUE
160          CONTINUE
170          CONTINUE
!        END IF
!
!      RETURN
!
!      End of ZSYMM .
!
!      END SUBROUTINE
!> \brief \b ZTRMM
!
!      ===== DOCUMENTATION =====
!
!      Online html documentation available at
!      http://www.netlib.org/lapack/explore-html/
!
!      Definition:
!      =====
!
!      SUBROUTINE ZTRMM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
!      .. Scalar Arguments ..
!      COMPLEX*16 ALPHA
!      INTEGER LDA,LDB,M,N
!      CHARACTER DIAG,SIDE,TRANSA,UPLO
!
!      ..
!      .. Array Arguments ..
!      COMPLEX*16 A(LDA,*),B(LDB,*)
!      ..
!
!
!> \par Purpose:
!      =====
!> \verbatim
!>
!> ZTRMM performs one of the matrix-matrix operations
!>
!>   B := alpha*op( A )*B,   or   B := alpha*B*op( A )
!>
!> where alpha is a scalar, B is an m by n matrix, A is a unit, or
!> non-unit, upper or lower triangular matrix and op( A ) is one of
!>

```

```

!> op( A ) = A or op( A ) = A**T or op( A ) = A**H.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] SIDE
!> \verbatim
!> SIDE is CHARACTER*1
!> On entry, SIDE specifies whether op( A ) multiplies B from
!> the left or right as follows:
!>
!> SIDE = 'L' or 'l' B := alpha*op( A )*B.
!>
!> SIDE = 'R' or 'r' B := alpha*B*op( A ).
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the matrix A is an upper or
!> lower triangular matrix as follows:
!>
!> UPLO = 'U' or 'u' A is an upper triangular matrix.
!>
!> UPLO = 'L' or 'l' A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANSA
!> \verbatim
!> TRANSA is CHARACTER*1
!> On entry, TRANSA specifies the form of op( A ) to be used in
!> the matrix multiplication as follows:
!>
!> TRANSA = 'N' or 'n' op( A ) = A.
!>
!> TRANSA = 'T' or 't' op( A ) = A**T.
!>
!> TRANSA = 'C' or 'c' op( A ) = A**H.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!> DIAG is CHARACTER*1
!> On entry, DIAG specifies whether or not A is unit triangular
!> as follows:
!>
!> DIAG = 'U' or 'u' A is assumed to be unit triangular.
!>
!> DIAG = 'N' or 'n' A is not assumed to be unit
!> triangular.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!> M is INTEGER
!> On entry, M specifies the number of rows of B. M must be at
!> least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the number of columns of B. N must be
!> at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is COMPLEX*16
!> On entry, ALPHA specifies the scalar alpha. When alpha is
!> zero then A is not referenced and B need not be set before
!> entry.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!> A is COMPLEX*16 array of DIMENSION ( LDA, k ), where k is m
!> when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'.
!> Before entry with UPLO = 'U' or 'u', the leading k by k
!> upper triangular part of the array A must contain the upper
!> triangular matrix and the strictly lower triangular part of
!> A is not referenced.
!> Before entry with UPLO = 'L' or 'l', the leading k by k
!> lower triangular part of the array A must contain the lower
!> triangular matrix and the strictly upper triangular part of
!> A is not referenced.
!> Note that when DIAG = 'U' or 'u', the diagonal elements of
!> A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. When SIDE = 'L' or 'l' then
!> LDA must be at least max( 1, m ), when SIDE = 'R' or 'r'
!> then LDA must be at least max( 1, n ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!> B is (input/output) COMPLEX*16 array of DIMENSION ( LDB, n ).
!> Before entry, the leading m by n part of the array B must
!> contain the matrix B, and on exit is overwritten by the
!> transformed matrix.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!> LDB is INTEGER
!> On entry, LDB specifies the first dimension of B as declared
!> in the calling (sub) program. LDB must be at least
!> max( 1, m ).

```

```

!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level3
!
!> \par Further Details:
! =====
!>
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!
! =====
! SUBROUTINE ZTRMM(SIDE,UPOLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPOLO
!
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*)
! ..
!
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX
!
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,J,K,NROWA
! LOGICAL LSIDE,NOCONJ,NOUNIT,UPPER
!
! ..
! .. Parameters ..
! COMPLEX*16 ONE
! PARAMETER (ONE= (1.0D+0,0.0D+0))
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
!
! Test the input parameters.
!
! LSIDE = LSAME(SIDE,'L')
! IF (LSIDE) THEN
!   NROWA = M
! ELSE
!   NROWA = N
! END IF
! NOCONJ = LSAME(TRANSA,'T')
! NOUNIT = LSAME(DIAG,'N')
! UPPER = LSAME(UPOLO,'U')
!
! INFO = 0
! IF ((.NOT.LSIDE) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!   INFO = 1
! ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPOLO,'L'))) THEN
!   INFO = 2
! ELSE IF ((.NOT.LSAME(TRANSA,'N')) .AND. &
!   (.NOT.LSAME(TRANSA,'T')) .AND. &
!   (.NOT.LSAME(TRANSA,'C'))) THEN
!   INFO = 3
! ELSE IF ((.NOT.LSAME(DIAG,'U')) .AND. (.NOT.LSAME(DIAG,'N'))) THEN
!   INFO = 4
! ELSE IF (M.LT.0) THEN
!   INFO = 5
! ELSE IF (N.LT.0) THEN
!   INFO = 6
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!   INFO = 9
! ELSE IF (LDB.LT.MAX(1,M)) THEN
!   INFO = 11
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZTRMM ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF (M.EQ.0 .OR. N.EQ.0) RETURN
!
! And when alpha.eq.zero.
!
! IF (ALPHA.EQ.ZERO) THEN
!   DO 20 J = 1,N

```

```

DO 10 I = 1,M
  B(I,J) = ZERO
10  CONTINUE
20  CONTINUE
  RETURN
END IF
!
! Start the operations.
!
IF (LSIDE) THEN
  IF (LSAME(TRANSA,'N')) THEN
!
! Form B := alpha*A*B.
!
    IF (UPPER) THEN
      DO 50 J = 1,N
        DO 40 K = 1,M
          IF (B(K,J).NE.ZERO) THEN
            TEMP = ALPHA*B(K,J)
            DO 30 I = 1,K - 1
              B(I,J) = B(I,J) + TEMP*A(I,K)
            30  CONTINUE
            IF (NOUNIT) TEMP = TEMP*A(K,K)
            B(K,J) = TEMP
          END IF
        CONTINUE
      CONTINUE
    ELSE
      DO 80 J = 1,N
        DO 70 K = M,1,-1
          IF (B(K,J).NE.ZERO) THEN
            TEMP = ALPHA*B(K,J)
            B(K,J) = TEMP
            IF (NOUNIT) B(K,J) = B(K,J)*A(K,K)
            DO 60 I = K + 1,M
              B(I,J) = B(I,J) + TEMP*A(I,K)
            60  CONTINUE
          END IF
        CONTINUE
      CONTINUE
    END IF
    CONTINUE
  END IF
  ELSE
!
! Form B := alpha*A**T*B or B := alpha*A**H*B.
!
    IF (UPPER) THEN
      DO 120 J = 1,N
        DO 110 I = M,1,-1
          TEMP = B(I,J)
          IF (NOCONJ) THEN
            IF (NOUNIT) TEMP = TEMP*A(I,I)
            DO 90 K = 1,I - 1
              TEMP = TEMP + A(K,I)*B(K,J)
            90  CONTINUE
          ELSE
            IF (NOUNIT) TEMP = TEMP*DCONJG(A(I,I))
            DO 100 K = 1,I - 1
              TEMP = TEMP + DCONJG(A(K,I))*B(K,J)
            100 CONTINUE
          END IF
          B(I,J) = ALPHA*TEMP
        CONTINUE
      CONTINUE
    ELSE
      DO 160 J = 1,N
        DO 150 I = 1,M
          TEMP = B(I,J)
          IF (NOCONJ) THEN
            IF (NOUNIT) TEMP = TEMP*A(I,I)
            DO 130 K = I + 1,M
              TEMP = TEMP + A(K,I)*B(K,J)
            130 CONTINUE
          ELSE
            IF (NOUNIT) TEMP = TEMP*DCONJG(A(I,I))
            DO 140 K = I + 1,M
              TEMP = TEMP + DCONJG(A(K,I))*B(K,J)
            140 CONTINUE
          END IF
          B(I,J) = ALPHA*TEMP
        CONTINUE
      CONTINUE
    END IF
  END IF
  ELSE
    IF (LSAME(TRANSA,'N')) THEN
!
! Form B := alpha*B*A.
!
      IF (UPPER) THEN
        DO 200 J = N,1,-1
          TEMP = ALPHA
          IF (NOUNIT) TEMP = TEMP*A(J,J)
          DO 170 I = 1,M
            B(I,J) = TEMP*B(I,J)
          170 CONTINUE
          DO 190 K = 1,J - 1
            IF (A(K,J).NE.ZERO) THEN
              TEMP = ALPHA*A(K,J)
              DO 180 I = 1,M
                B(I,J) = B(I,J) + TEMP*B(I,K)
              180 CONTINUE
            END IF
          CONTINUE
        CONTINUE
      ELSE
        DO 240 J = 1,N
          TEMP = ALPHA
          IF (NOUNIT) TEMP = TEMP*A(J,J)
          DO 210 I = 1,M
            B(I,J) = TEMP*B(I,J)
          210 CONTINUE
          DO 230 K = J + 1,N
            IF (A(K,J).NE.ZERO) THEN
              TEMP = ALPHA*A(K,J)

```

```

                DO 220 I = 1,M
                  B(I,J) = B(I,J) + TEMP*B(I,K)
220              CONTINUE
            END IF
230          CONTINUE
240        CONTINUE
      END IF
    ELSE
      Form B := alpha*B*A**T or B := alpha*B*A**H.
      IF (UPPER) THEN
        DO 280 K = 1,N
          DO 260 J = 1,K - 1
            IF (A(J,K).NE.ZERO) THEN
              IF (NOCONJ) THEN
                TEMP = ALPHA*A(J,K)
              ELSE
                TEMP = ALPHA*DCONJG(A(J,K))
              END IF
              DO 250 I = 1,M
                B(I,J) = B(I,J) + TEMP*B(I,K)
250            CONTINUE
              END IF
            CONTINUE
            TEMP = ALPHA
            IF (NOUNIT) THEN
              IF (NOCONJ) THEN
                TEMP = TEMP*A(K,K)
              ELSE
                TEMP = TEMP*DCONJG(A(K,K))
              END IF
            END IF
            IF (TEMP.NE.ONE) THEN
              DO 270 I = 1,M
                B(I,K) = TEMP*B(I,K)
270              CONTINUE
            END IF
            CONTINUE
          END IF
        ELSE
          DO 320 K = N,1,-1
            DO 300 J = K + 1,N
              IF (A(J,K).NE.ZERO) THEN
                IF (NOCONJ) THEN
                  TEMP = ALPHA*A(J,K)
                ELSE
                  TEMP = ALPHA*DCONJG(A(J,K))
                END IF
                DO 290 I = 1,M
                  B(I,J) = B(I,J) + TEMP*B(I,K)
290                CONTINUE
              END IF
            CONTINUE
            TEMP = ALPHA
            IF (NOUNIT) THEN
              IF (NOCONJ) THEN
                TEMP = TEMP*A(K,K)
              ELSE
                TEMP = TEMP*DCONJG(A(K,K))
              END IF
            END IF
            IF (TEMP.NE.ONE) THEN
              DO 310 I = 1,M
                B(I,K) = TEMP*B(I,K)
310              CONTINUE
            END IF
            CONTINUE
          END IF
        END IF
      END IF
    RETURN
  !
  ! End of ZTRMM .
  !
  END SUBROUTINE
!> \brief \b ZTRSM
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> ZTRSM solves one of the matrix equations
!>
!>   op( A )*X = alpha*B, or X*op( A ) = alpha*B,
!>
!> where alpha is a scalar, X and B are m by n matrices, A is a unit, or
!> non-unit, upper or lower triangular matrix and op( A ) is one of
!>
!>   op( A ) = A or op( A ) = A**T or op( A ) = A**H.
!>
!> The matrix X is overwritten on B.
!> \endverbatim

```

```

!
! Arguments:
! =====
!
!> \param[in] SIDE
!> \verbatim
!>     SIDE is CHARACTER*1
!>     On entry, SIDE specifies whether op( A ) appears on the left
!>     or right of X as follows:
!>
!>         SIDE = 'L' or 'l'    op( A ) * X = alpha * B.
!>
!>         SIDE = 'R' or 'r'    X * op( A ) = alpha * B.
!> \endverbatim
!>
!> \param[in] UPLO
!> \verbatim
!>     UPLO is CHARACTER*1
!>     On entry, UPLO specifies whether the matrix A is an upper or
!>     lower triangular matrix as follows:
!>
!>         UPLO = 'U' or 'u'    A is an upper triangular matrix.
!>
!>         UPLO = 'L' or 'l'    A is a lower triangular matrix.
!> \endverbatim
!>
!> \param[in] TRANSA
!> \verbatim
!>     TRANSA is CHARACTER*1
!>     On entry, TRANSA specifies the form of op( A ) to be used in
!>     the matrix multiplication as follows:
!>
!>         TRANSA = 'N' or 'n'    op( A ) = A.
!>
!>         TRANSA = 'T' or 't'    op( A ) = A**T.
!>
!>         TRANSA = 'C' or 'c'    op( A ) = A**H.
!> \endverbatim
!>
!> \param[in] DIAG
!> \verbatim
!>     DIAG is CHARACTER*1
!>     On entry, DIAG specifies whether or not A is unit triangular
!>     as follows:
!>
!>         DIAG = 'U' or 'u'    A is assumed to be unit triangular.
!>
!>         DIAG = 'N' or 'n'    A is not assumed to be unit
!>                             triangular.
!> \endverbatim
!>
!> \param[in] M
!> \verbatim
!>     M is INTEGER
!>     On entry, M specifies the number of rows of B. M must be at
!>     least zero.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the number of columns of B. N must be
!>     at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha. When alpha is
!>     zero then A is not referenced and B need not be set before
!>     entry.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, k ),
!>     where k is m when SIDE = 'L' or 'l'
!>     and k is n when SIDE = 'R' or 'r'.
!>     Before entry with UPLO = 'U' or 'u', the leading k by k
!>     upper triangular part of the array A must contain the upper
!>     triangular matrix and the strictly lower triangular part of
!>     A is not referenced.
!>     Before entry with UPLO = 'L' or 'l', the leading k by k
!>     lower triangular part of the array A must contain the lower
!>     triangular matrix and the strictly upper triangular part of
!>     A is not referenced.
!>     Note that when DIAG = 'U' or 'u', the diagonal elements of
!>     A are not referenced either, but are assumed to be unity.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When SIDE = 'L' or 'l' then
!>     LDA must be at least max( 1, m ), when SIDE = 'R' or 'r'
!>     then LDA must be at least max( 1, n ).
!> \endverbatim
!>
!> \param[in,out] B
!> \verbatim
!>     B is COMPLEX*16 array of DIMENSION ( LDB, n ).
!>     Before entry, the leading m by n part of the array B must
!>     contain the right-hand side matrix B, and on exit is
!>     overwritten by the solution matrix X.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. LDB must be at least
!>     max( 1, m ).
!> \endverbatim

```

```

!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level3
!
!> \par Further Details:
! =====
!>
!> \verbatim
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!
! =====
! SUBROUTINE ZTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA
! INTEGER LDA,LDB,M,N
! CHARACTER DIAG,SIDE,TRANSA,UPLO
!
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*)
! ..
!
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
! INTRINSIC DCONJG,MAX
!
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP
! INTEGER I,INFO,J,K,NROWA
! LOGICAL LSIDE,NOCONJ,NOUNIT,UPPER
!
! ..
! .. Parameters ..
! COMPLEX*16 ONE=(1.0D+0,0.0D+0)
! COMPLEX*16 ZERO=(0.0D+0,0.0D+0)
! ..
!
! Test the input parameters.
!
! LSIDE = LSAME(SIDE,'L')
! IF (LSIDE) THEN
!   NROWA = M
! ELSE
!   NROWA = N
! END IF
! NOCONJ = LSAME(TRANSA,'T')
! NOUNIT = LSAME(DIAG,'N')
! UPPER = LSAME(UPLO,'U')
!
! INFO = 0
! IF ((.NOT.LSIDE) .AND. (.NOT.LSAME(SIDE,'R'))) THEN
!   INFO = 1
! ELSE IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!   INFO = 2
! ELSE IF ((.NOT.LSAME(TRANSA,'N')) .AND.&
!   (.NOT.LSAME(TRANSA,'T')) .AND.&
!   (.NOT.LSAME(TRANSA,'C'))) THEN
!   INFO = 3
! ELSE IF ((.NOT.LSAME(DIAG,'U')) .AND. (.NOT.LSAME(DIAG,'N'))) THEN
!   INFO = 4
! ELSE IF (M.LT.0) THEN
!   INFO = 5
! ELSE IF (N.LT.0) THEN
!   INFO = 6
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!   INFO = 9
! ELSE IF (LDB.LT.MAX(1,M)) THEN
!   INFO = 11
! END IF
! IF (INFO.NE.0) THEN
!   CALL XERBLA('ZTRSM ',INFO)
!   RETURN
! END IF
!
! Quick return if possible.
!
! IF (M.EQ.0 .OR. N.EQ.0) RETURN
!
! And when alpha.eq.zero.
!
! IF (ALPHA.EQ.ZERO) THEN
!   DO 20 J = 1,N
!     DO 10 I = 1,M

```

```

        B(I,J) = ZERO
10      CONTINUE
20      CONTINUE
      RETURN
    END IF
!
!   Start the operations.
!
    IF (LSIDE) THEN
      IF (LSAME(TRANSA,'N')) THEN
!
!       Form B := alpha*inv( A )*B.
!
        IF (UPPER) THEN
          DO 60 J = 1,N
            IF (ALPHA.NE.ONE) THEN
              DO 30 I = 1,M
                B(I,J) = ALPHA*B(I,J)
30              CONTINUE
            END IF
            DO 50 K = M,1,-1
              IF (B(K,J).NE.ZERO) THEN
                IF (NOUNIT) B(K,J) = B(K,J)/A(K,K)
                DO 40 I = 1,K-1
                  B(I,J) = B(I,J) - B(K,J)*A(I,K)
40                CONTINUE
              END IF
            CONTINUE
          END IF
          CONTINUE
60        ELSE
          DO 100 J = 1,N
            IF (ALPHA.NE.ONE) THEN
              DO 70 I = 1,M
                B(I,J) = ALPHA*B(I,J)
70              CONTINUE
            END IF
            DO 90 K = 1,M
              IF (B(K,J).NE.ZERO) THEN
                IF (NOUNIT) B(K,J) = B(K,J)/A(K,K)
                DO 80 I = K+1,M
                  B(I,J) = B(I,J) - B(K,J)*A(I,K)
80                CONTINUE
              END IF
            CONTINUE
          END IF
          CONTINUE
100       END IF
        ELSE
!
!       Form B := alpha*inv( A**T )*B
!       or B := alpha*inv( A**H )*B.
!
          IF (UPPER) THEN
            DO 140 J = 1,N
              DO 130 I = 1,M
                TEMP = ALPHA*B(I,J)
                IF (NOCONJ) THEN
                  DO 110 K = 1,I-1
                    TEMP = TEMP - A(K,I)*B(K,J)
110                  CONTINUE
                ELSE
                  DO 120 K = 1,I-1
                    TEMP = TEMP - DCONJG(A(K,I))*B(K,J)
120                  CONTINUE
                IF (NOUNIT) TEMP = TEMP/DCONJG(A(I,I))
              END IF
              B(I,J) = TEMP
130            CONTINUE
          END IF
          CONTINUE
140        ELSE
          DO 180 J = 1,N
            DO 170 I = M,1,-1
              TEMP = ALPHA*B(I,J)
              IF (NOCONJ) THEN
                DO 150 K = I+1,M
                  TEMP = TEMP - A(K,I)*B(K,J)
150                CONTINUE
              ELSE
                DO 160 K = I+1,M
                  TEMP = TEMP - DCONJG(A(K,I))*B(K,J)
160                CONTINUE
              IF (NOUNIT) TEMP = TEMP/DCONJG(A(I,I))
            END IF
            B(I,J) = TEMP
170          CONTINUE
180        CONTINUE
      END IF
    ELSE
      IF (LSAME(TRANSA,'N')) THEN
!
!       Form B := alpha*B*inv( A ).
!
        IF (UPPER) THEN
          DO 230 J = 1,N
            IF (ALPHA.NE.ONE) THEN
              DO 190 I = 1,M
                B(I,J) = ALPHA*B(I,J)
190              CONTINUE
            END IF
            DO 210 K = 1,J-1
              IF (A(K,J).NE.ZERO) THEN
                DO 200 I = 1,M
                  B(I,J) = B(I,J) - A(K,J)*B(I,K)
200                CONTINUE
              END IF
            CONTINUE
          END IF
          CONTINUE
210        IF (NOUNIT) THEN
          TEMP = ONE/A(J,J)
          DO 220 I = 1,M
            B(I,J) = TEMP*B(I,J)
220          CONTINUE
        END IF
      END IF
    END IF
  END IF

```

```

230      CONTINUE
      ELSE
        DO 280 J = N,1,-1
          IF (ALPHA.NE.ONE) THEN
            DO 240 I = 1,M
              B(I,J) = ALPHA*B(I,J)
240          CONTINUE
            END IF
            DO 260 K = J + 1,N
              IF (A(K,J).NE.ZERO) THEN
                DO 250 I = 1,M
                  B(I,J) = B(I,J) - A(K,J)*B(I,K)
250          CONTINUE
                END IF
              CONTINUE
            END IF
            IF (NOUNIT) THEN
              TEMP = ONE/A(J,J)
              DO 270 I = 1,M
                B(I,J) = TEMP*B(I,J)
270          CONTINUE
              END IF
            CONTINUE
          END IF
        END IF
      ELSE
        Form B := alpha*B*inv( A**T )
        or   B := alpha*B*inv( A**H ).

        IF (UPPER) THEN
          DO 330 K = N,1,-1
            IF (NOUNIT) THEN
              IF (NOCONJ) THEN
                TEMP = ONE/A(K,K)
              ELSE
                TEMP = ONE/DCONJG(A(K,K))
              END IF
              DO 290 I = 1,M
                B(I,K) = TEMP*B(I,K)
290          CONTINUE
              END IF
              DO 310 J = 1,K - 1
                IF (A(J,K).NE.ZERO) THEN
                  IF (NOCONJ) THEN
                    TEMP = A(J,K)
                  ELSE
                    TEMP = DCONJG(A(J,K))
                  END IF
                  DO 300 I = 1,M
                    B(I,J) = B(I,J) - TEMP*B(I,K)
300          CONTINUE
                  END IF
                CONTINUE
              END IF
              IF (ALPHA.NE.ONE) THEN
                DO 320 I = 1,M
                  B(I,K) = ALPHA*B(I,K)
320          CONTINUE
                END IF
              CONTINUE
            ELSE
              DO 380 K = 1,N
                IF (NOUNIT) THEN
                  IF (NOCONJ) THEN
                    TEMP = ONE/A(K,K)
                  ELSE
                    TEMP = ONE/DCONJG(A(K,K))
                  END IF
                  DO 340 I = 1,M
                    B(I,K) = TEMP*B(I,K)
340          CONTINUE
                  END IF
                  DO 360 J = K + 1,N
                    IF (A(J,K).NE.ZERO) THEN
                      IF (NOCONJ) THEN
                        TEMP = A(J,K)
                      ELSE
                        TEMP = DCONJG(A(J,K))
                      END IF
                      DO 350 I = 1,M
                        B(I,J) = B(I,J) - TEMP*B(I,K)
350          CONTINUE
                      END IF
                    CONTINUE
                  END IF
                  IF (ALPHA.NE.ONE) THEN
                    DO 370 I = 1,M
                      B(I,K) = ALPHA*B(I,K)
370          CONTINUE
                    END IF
                  CONTINUE
                END IF
              END IF
            END IF
          END IF
        RETURN
      !
      ! End of ZTRSM .
      !
      END SUBROUTINE
!> \brief \b ZHERK
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZHERK(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!
! .. Scalar Arguments ..
! DOUBLE PRECISION ALPHA,BETA
! INTEGER K,LDA,LDC,N
! CHARACTER TRANS,UPLO
! ..

```

```

! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),C(LDC,*)
! ..
!
!> \par Purpose:
!> =====
!>
!> \verbatim
!>
!> ZHERK performs one of the hermitian rank k operations
!>
!>   C := alpha*A*A**H + beta*C,
!>
!> or
!>
!>   C := alpha*A**H*A + beta*C,
!>
!> where alpha and beta are real scalars, C is an n by n hermitian
!> matrix and A is an n by k matrix in the first case and a k by n
!> matrix in the second case.
!> \endverbatim
!
! Arguments:
! =====
!
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array C is to be referenced as
!> follows:
!>
!>     UPLO = 'U' or 'u' Only the upper triangular part of C
!>                       is to be referenced.
!>
!>     UPLO = 'L' or 'l' Only the lower triangular part of C
!>                       is to be referenced.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>     TRANS = 'N' or 'n' C := alpha*A*A**H + beta*C.
!>
!>     TRANS = 'C' or 'c' C := alpha*A**H*A + beta*C.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix C. N must be
!> at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!> K is INTEGER
!> On entry with TRANS = 'N' or 'n', K specifies the number
!> of columns of the matrix A, and on entry with
!> TRANS = 'C' or 'c', K specifies the number of rows of the
!> matrix A. K must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!> ALPHA is DOUBLE PRECISION .
!> On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!> A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!> k when TRANS = 'N' or 'n', and is n otherwise.
!> Before entry with TRANS = 'N' or 'n', the leading n by k
!> part of the array A must contain the matrix A, otherwise
!> the leading k by n part of the array A must contain the
!> matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!> LDA is INTEGER
!> On entry, LDA specifies the first dimension of A as declared
!> in the calling (sub) program. When TRANS = 'N' or 'n'
!> then LDA must be at least max( 1, n ), otherwise LDA must
!> be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!> BETA is DOUBLE PRECISION.
!> On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!> C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!> Before entry with UPLO = 'U' or 'u', the leading n by n
!> upper triangular part of the array C must contain the upper
!> triangular part of the hermitian matrix and the strictly
!> lower triangular part of C is not referenced. On exit, the
!> upper triangular part of the array C is overwritten by the
!> upper triangular part of the updated matrix.
!> Before entry with UPLO = 'L' or 'l', the leading n by n
!> lower triangular part of the array C must contain the lower
!> triangular part of the hermitian matrix and the strictly
!> upper triangular part of C is not referenced. On exit, the
!> lower triangular part of the array C is overwritten by the
!> lower triangular part of the updated matrix.
!> Note that the imaginary parts of the diagonal elements need

```

```

!>         not be set, they are assumed to be zero, and on exit they
!>         are set to zero.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>         LDC is INTEGER
!>         On entry, LDC specifies the first dimension of C as declared
!>         in the calling (sub) program. LDC must be at least
!>         max( 1, n ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level3
!
!> \par Further Details:
!> =====
!> \verbatim
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!>
!> -- Modified 8-Nov-93 to set C(J,J) to DBLE( C(J,J) ) when BETA = 1.
!>   Ed Anderson, Cray Research Inc.
!> \endverbatim
!>
!> =====
!> SUBROUTINE ZHERK(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
DOUBLE PRECISION ALPHA,BETA
INTEGER K,LDA,LDC,N
CHARACTER TRANS,UPLO
!
! ..
! .. Array Arguments ..
COMPLEX*16 A(LDA,*),C(LDC,*)
!
! ..
! =====
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
INTRINSIC DBLE,DCMPLX,DCONJG,MAX
!
! ..
! .. Local Scalars ..
COMPLEX*16 TEMP
DOUBLE PRECISION RTEMP
INTEGER I,INFO,J,L,NROWA
LOGICAL UPPER
!
! ..
! .. Parameters ..
DOUBLE PRECISION ONE,ZERO
PARAMETER (ONE=1.0D+0,ZERO=0.0D+0)
!
! ..
!
! Test the input parameters.
!
! IF (LSAME(TRANS,'N')) THEN
!     NROWA = N
! ELSE
!     NROWA = K
! END IF
! UPPER = LSAME(UPLO,'U')
!
! INFO = 0
! IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!     INFO = 1
! ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
!         (.NOT.LSAME(TRANS,'C'))) THEN
!     INFO = 2
! ELSE IF (N.LT.0) THEN
!     INFO = 3
! ELSE IF (K.LT.0) THEN
!     INFO = 4
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!     INFO = 7
! ELSE IF (LDC.LT.MAX(1,N)) THEN
!     INFO = 10
! END IF
! IF (INFO.NE.0) THEN
!     CALL XERBLA('ZHERK ',INFO)
!     RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).OR.&
!     (K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN

```

```

!
!   And when  alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
  IF (UPPER) THEN
    IF (BETA.EQ.ZERO) THEN
      DO 20 J = 1,N
        DO 10 I = 1,J
          C(I,J) = ZERO
10        CONTINUE
20      CONTINUE
    ELSE
      DO 40 J = 1,N
        DO 30 I = 1,J - 1
          C(I,J) = BETA*C(I,J)
30        CONTINUE
          C(J,J) = BETA*DBLE(C(J,J))
40        CONTINUE
      END IF
    ELSE
      IF (BETA.EQ.ZERO) THEN
        DO 60 J = 1,N
          DO 50 I = J,N
            C(I,J) = ZERO
50          CONTINUE
60          CONTINUE
      ELSE
        DO 80 J = 1,N
          C(J,J) = BETA*DBLE(C(J,J))
          DO 70 I = J + 1,N
            C(I,J) = BETA*C(I,J)
70          CONTINUE
80          CONTINUE
      END IF
    END IF
    RETURN
  END IF
!
!   Start the operations.
!
IF (LSAME(TRANS,'N')) THEN
!
!   Form C := alpha*A**H + beta*C.
!
  IF (UPPER) THEN
    DO 130 J = 1,N
      IF (BETA.EQ.ZERO) THEN
        DO 90 I = 1,J
          C(I,J) = ZERO
90        CONTINUE
      ELSE IF (BETA.NE.ONE) THEN
        DO 100 I = 1,J - 1
          C(I,J) = BETA*C(I,J)
100        CONTINUE
          C(J,J) = BETA*DBLE(C(J,J))
      ELSE
        C(J,J) = DBLE(C(J,J))
      END IF
      DO 120 L = 1,K
        IF (A(J,L).NE.DCMPLX(ZERO)) THEN
          TEMP = ALPHA*DCONJG(A(J,L))
          DO 110 I = 1,J - 1
            C(I,J) = C(I,J) + TEMP*A(I,L)
110          CONTINUE
          print *, 'I,J=',I,J
          C(J,J) = DBLE(C(J,J)) + DBLE(TEMP*A(I,L))
        END IF
      CONTINUE
120    CONTINUE
130  CONTINUE
  ELSE
    DO 180 J = 1,N
      IF (BETA.EQ.ZERO) THEN
        DO 140 I = J,N
          C(I,J) = ZERO
140        CONTINUE
      ELSE IF (BETA.NE.ONE) THEN
        C(J,J) = BETA*DBLE(C(J,J))
        DO 150 I = J + 1,N
          C(I,J) = BETA*C(I,J)
150        CONTINUE
      ELSE
        C(J,J) = DBLE(C(J,J))
      END IF
      DO 170 L = 1,K
        IF (A(J,L).NE.DCMPLX(ZERO)) THEN
          TEMP = ALPHA*DCONJG(A(J,L))
          C(J,J) = DBLE(C(J,J)) + DBLE(TEMP*A(J,L))
          DO 160 I = J + 1,N
            C(I,J) = C(I,J) + TEMP*A(I,L)
160          CONTINUE
        END IF
      CONTINUE
170    CONTINUE
180  CONTINUE
  END IF
  Form C := alpha*A**H*A + beta*C.
!
  IF (UPPER) THEN
    DO 220 J = 1,N
      DO 200 I = 1,J - 1
        TEMP = ZERO
        DO 190 L = 1,K
          TEMP = TEMP + DCONJG(A(L,I))*A(L,J)
190        CONTINUE
        IF (BETA.EQ.ZERO) THEN
          C(I,J) = ALPHA*TEMP
        ELSE
          C(I,J) = ALPHA*TEMP + BETA*C(I,J)
        END IF
      CONTINUE
      RTEMP = ZERO
      DO 210 L = 1,K
        RTEMP = RTEMP + DCONJG(A(L,J))*A(L,J)
200      CONTINUE
210    CONTINUE
  END IF

```

```

210      CONTINUE
      IF (BETA.EQ.ZERO) THEN
        C(J,J) = ALPHA*RTEMP
      ELSE
        C(J,J) = ALPHA*RTEMP + BETA*DBLE(C(J,J))
      END IF
220    CONTINUE
  ELSE
    DO 260 J = 1,N
      RTEMP = ZERO
      DO 230 L = 1,K
        RTEMP = RTEMP + DCONJG(A(L,J))*A(L,J)
230      CONTINUE
      IF (BETA.EQ.ZERO) THEN
        C(J,J) = ALPHA*RTEMP
      ELSE
        C(J,J) = ALPHA*RTEMP + BETA*DBLE(C(J,J))
      END IF
      DO 250 I = J + 1,N
        TEMP = ZERO
        DO 240 L = 1,K
          TEMP = TEMP + DCONJG(A(L,I))*A(L,J)
240        CONTINUE
        IF (BETA.EQ.ZERO) THEN
          C(I,J) = ALPHA*TEMP
        ELSE
          C(I,J) = ALPHA*TEMP + BETA*C(I,J)
        END IF
      END IF
    CONTINUE
250  CONTINUE
260  CONTINUE
  END IF
END IF
!
! RETURN
!
! End of ZHERK .
!
! END SUBROUTINE
!> \brief \b ZSYRK
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZSYRK(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER K,LDA,LDC,N
! CHARACTER TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),C(LDC,*)
! ..
!
!> \par Purpose:
! =====
!>
!> \verbatim
!> ZSYRK performs one of the symmetric rank k operations
!>
!>   C := alpha*A*A**T + beta*C,
!>
!> or
!>   C := alpha*A**T*A + beta*C,
!>
!> where alpha and beta are scalars, C is an n by n symmetric matrix
!> and A is an n by k matrix in the first case and a k by n matrix
!> in the second case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array C is to be referenced as
!> follows:
!>
!>   UPLO = 'U' or 'u' Only the upper triangular part of C
!>                     is to be referenced.
!>
!>   UPLO = 'L' or 'l' Only the lower triangular part of C
!>                     is to be referenced.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n' C := alpha*A*A**T + beta*C.
!>
!>   TRANS = 'T' or 't' C := alpha*A**T*A + beta*C.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!> N is INTEGER
!> On entry, N specifies the order of the matrix C. N must be
!> at least zero.
!> \endverbatim
!> \param[in] K

```

```

!> \verbatim
!>     K is INTEGER
!>     On entry with TRANS = 'N' or 'n', K specifies the number
!>     of columns of the matrix A, and on entry with
!>     TRANS = 'T' or 't', K specifies the number of rows of the
!>     matrix A. K must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array A must contain the matrix A, otherwise
!>     the leading k by n part of the array A must contain the
!>     matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDA must be at least max( 1, n ), otherwise LDA must
!>     be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>     C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array C must contain the upper
!>     triangular part of the symmetric matrix and the strictly
!>     lower triangular part of C is not referenced. On exit, the
!>     upper triangular part of the array C is overwritten by the
!>     upper triangular part of the updated matrix.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array C must contain the lower
!>     triangular part of the symmetric matrix and the strictly
!>     upper triangular part of C is not referenced. On exit, the
!>     lower triangular part of the array C is overwritten by the
!>     lower triangular part of the updated matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>     LDC is INTEGER
!>     On entry, LDC specifies the first dimension of C as declared
!>     in the calling (sub) program. LDC must be at least
!>     max( 1, n ).
!> \endverbatim
!
! Authors:
! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level3
!
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
!>
! =====
!
! SUBROUTINE ZSYRK(UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.---
! November 2011
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER K,LDA,LDC,N
! CHARACTER TRANS,UPLO
!
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),C(LDC,*)
! ..
!
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..

```

```

!      EXTERNAL XERBLA
!      ..
!      .. Intrinsic Functions ..
INTRINSIC MAX
!      ..
!      .. Local Scalars ..
COMPLEX*16 TEMP
INTEGER I,INFO,J,L,NROWA
LOGICAL UPPER
!      ..
!      .. Parameters ..
COMPLEX*16 ONE
PARAMETER (ONE= (1.0D+0,0.0D+0))
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!      ..
!
!      Test the input parameters.
!
IF (LSAME(TRANS,'N')) THEN
    NROWA = N
ELSE
    NROWA = K
END IF
UPPER = LSAME(UPLO,'U')
!
INFO = 0
IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
    INFO = 1
ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
        (.NOT.LSAME(TRANS,'T'))) THEN
    INFO = 2
ELSE IF (N.LT.0) THEN
    INFO = 3
ELSE IF (K.LT.0) THEN
    INFO = 4
ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
    INFO = 7
ELSE IF (LDC.LT.MAX(1,N)) THEN
    INFO = 10
END IF
IF (INFO.NE.0) THEN
    CALL XERBLA('ZSYRK ',INFO)
    RETURN
END IF
!
!      Quick return if possible.
!
IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).OR.&
    (K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN
!
!      And when alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
    IF (UPPER) THEN
        IF (BETA.EQ.ZERO) THEN
            DO 20 J = 1,N
                DO 10 I = 1,J
                    C(I,J) = ZERO
                CONTINUE
            CONTINUE
        ELSE
            DO 40 J = 1,N
                DO 30 I = 1,J
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            CONTINUE
        END IF
    ELSE
        IF (BETA.EQ.ZERO) THEN
            DO 60 J = 1,N
                DO 50 I = J,N
                    C(I,J) = ZERO
                CONTINUE
            CONTINUE
        ELSE
            DO 80 J = 1,N
                DO 70 I = J,N
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            CONTINUE
        END IF
    END IF
    RETURN
END IF
!
!      Start the operations.
!
IF (LSAME(TRANS,'N')) THEN
!
!      Form C := alpha*A*A**T + beta*C.
!
    IF (UPPER) THEN
        DO 130 J = 1,N
            IF (BETA.EQ.ZERO) THEN
                DO 90 I = 1,J
                    C(I,J) = ZERO
                CONTINUE
            ELSE IF (BETA.NE.ONE) THEN
                DO 100 I = 1,J
                    C(I,J) = BETA*C(I,J)
                CONTINUE
            END IF
            DO 120 L = 1,K
                IF (A(J,L).NE.ZERO) THEN
                    TEMP = ALPHA*A(J,L)
                    DO 110 I = 1,J
                        C(I,J) = C(I,J) + TEMP*A(I,L)
                    CONTINUE
                END IF
            CONTINUE
        CONTINUE
    ELSE
        DO 180 J = 1,N

```

```

        IF (BETA.EQ.ZERO) THEN
            DO 140 I = J,N
                C(I,J) = ZERO
            CONTINUE
140        ELSE IF (BETA.NE.ONE) THEN
            DO 150 I = J,N
                C(I,J) = BETA*C(I,J)
            CONTINUE
150        END IF
        DO 170 L = 1,K
            IF (A(J,L).NE.ZERO) THEN
                TEMP = ALPHA*A(J,L)
                DO 160 I = J,N
                    C(I,J) = C(I,J) + TEMP*A(I,L)
                CONTINUE
160            END IF
        CONTINUE
170    CONTINUE
180    CONTINUE
    END IF
ELSE
    Form C := alpha*A**T*A + beta*C.
    IF (UPPER) THEN
        DO 210 J = 1,N
            DO 200 I = 1,J
                TEMP = ZERO
                DO 190 L = 1,K
                    TEMP = TEMP + A(L,I)*A(L,J)
                CONTINUE
190            IF (BETA.EQ.ZERO) THEN
                C(I,J) = ALPHA*TEMP
            ELSE
                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
            END IF
        CONTINUE
200    CONTINUE
210    ELSE
        DO 240 J = 1,N
            DO 230 I = J,N
                TEMP = ZERO
                DO 220 L = 1,K
                    TEMP = TEMP + A(L,I)*A(L,J)
                CONTINUE
220            IF (BETA.EQ.ZERO) THEN
                C(I,J) = ALPHA*TEMP
            ELSE
                C(I,J) = ALPHA*TEMP + BETA*C(I,J)
            END IF
        CONTINUE
230    CONTINUE
240    CONTINUE
    END IF
END IF
!
RETURN
!
End of ZSYRK .
!
END SUBROUTINE
*> \brief \b ZHER2K
!
! ===== DOCUMENTATION =====
!
! Online html documentation available at
! http://www.netlib.org/lapack/explore-html/
!
! Definition:
! =====
!
! SUBROUTINE ZHER2K (UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA
! DOUBLE PRECISION BETA
! INTEGER K,LDA,LDB,LDC,N
! CHARACTER TRANS,UPLO
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
! ..
!
!> \par Purpose:
! =====
!> \verbatim
!> ZHER2K performs one of the hermitian rank 2k operations
!>
!> C := alpha*A*B**H + conjg( alpha )*B*A**H + beta*C,
!>
!> or
!>
!> C := alpha*A**H*B + conjg( alpha )*B**H*A + beta*C,
!>
!> where alpha and beta are scalars with beta real, C is an n by n
!> hermitian matrix and A and B are n by k matrices in the first case
!> and k by n matrices in the second case.
!> \endverbatim
!
! Arguments:
! =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array C is to be referenced as
!> follows:
!>
!> UPLO = 'U' or 'u' Only the upper triangular part of C
!> is to be referenced.
!>
!> UPLO = 'L' or 'l' Only the lower triangular part of C

```

```

!>                                     is to be referenced.
!> \endverbatim
!>
!> \param[in] TRANS
!> \verbatim
!>     TRANS is CHARACTER*1
!>     On entry, TRANS specifies the operation to be performed as
!>     follows:
!>
!>         TRANS = 'N' or 'n'    C := alpha*A*B**H      +
!>                                conjg( alpha )*B*A**H +
!>                                beta*C.
!>
!>         TRANS = 'C' or 'c'    C := alpha*A**H*B      +
!>                                conjg( alpha )*B**H*A +
!>                                beta*C.
!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix C. N must be
!>     at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with TRANS = 'N' or 'n', K specifies the number
!>     of columns of the matrices A and B, and on entry with
!>     TRANS = 'C' or 'c', K specifies the number of rows of the
!>     matrices A and B. K must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16 .
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array A must contain the matrix A, otherwise
!>     the leading k by n part of the array A must contain the
!>     matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDA must be at least max( 1, n ), otherwise LDA must
!>     be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!>     B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array B must contain the matrix B, otherwise
!>     the leading k by n part of the array B must contain the
!>     matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDB must be at least max( 1, n ), otherwise LDB must
!>     be at least max( 1, k ).
!>     Unchanged on exit.
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is DOUBLE PRECISION .
!>     On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>     C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array C must contain the upper
!>     triangular part of the hermitian matrix and the strictly
!>     lower triangular part of C is not referenced. On exit, the
!>     upper triangular part of the array C is overwritten by the
!>     upper triangular part of the updated matrix.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array C must contain the lower
!>     triangular part of the hermitian matrix and the strictly
!>     upper triangular part of C is not referenced. On exit, the
!>     lower triangular part of the array C is overwritten by the
!>     lower triangular part of the updated matrix.
!>     Note that the imaginary parts of the diagonal elements need
!>     not be set, they are assumed to be zero, and on exit they
!>     are set to zero.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>     LDC is INTEGER
!>     On entry, LDC specifies the first dimension of C as declared
!>     in the calling (sub) program. LDC must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> Authors:

```

```

! =====
!
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!
!> \date November 2011
!
!> \ingroup complex16_blas_level3
!
!> \par Further Details:
! =====
!> \verbatim
!>
!> Level 3 Blas routine.
!>
!> -- Written on 8-February-1989.
!>   Jack Dongarra, Argonne National Laboratory.
!>   Iain Duff, AERE Harwell.
!>   Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>   Sven Hammarling, Numerical Algorithms Group Ltd.
!>
!> -- Modified 8-Nov-93 to set C(J,J) to DBLE( C(J,J) ) when BETA = 1.
!>   Ed Anderson, Cray Research Inc.
!> \endverbatim
!>
! =====
SUBROUTINE ZHER2K(UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd. ---
! November 2011
!
! .. Scalar Arguments ..
COMPLEX*16 ALPHA
DOUBLE PRECISION BETA
INTEGER K,LDA,LDB,LDC,N
CHARACTER TRANS,UPLO
!
! ..
! .. Array Arguments ..
COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
!
! ..
! =====
!
! .. External Functions ..
LOGICAL LSAME
EXTERNAL LSAME
!
! ..
! .. External Subroutines ..
EXTERNAL XERBLA
!
! ..
! .. Intrinsic Functions ..
INTRINSIC DBLE,DCONJG,MAX
!
! ..
! .. Local Scalars ..
COMPLEX*16 TEMP1,TEMP2
INTEGER I,INFO,J,L,NROWA
LOGICAL UPPER
!
! ..
! .. Parameters ..
DOUBLE PRECISION ONE
PARAMETER (ONE=1.0D+0)
COMPLEX*16 ZERO
PARAMETER (ZERO= (0.0D+0,0.0D+0))
!
! ..
!
! Test the input parameters.
!
IF (LSAME(TRANS,'N')) THEN
  NROWA = N
ELSE
  NROWA = K
END IF
UPPER = LSAME(UPLO,'U')
!
INFO = 0
IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
  INFO = 1
ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
(.NOT.LSAME(TRANS,'C'))) THEN
  INFO = 2
ELSE IF (N.LT.0) THEN
  INFO = 3
ELSE IF (K.LT.0) THEN
  INFO = 4
ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
  INFO = 7
ELSE IF (LDB.LT.MAX(1,NROWA)) THEN
  INFO = 9
ELSE IF (LDC.LT.MAX(1,N)) THEN
  INFO = 12
END IF
IF (INFO.NE.0) THEN
  CALL XERBLA('ZHER2K',INFO)
  RETURN
END IF
!
! Quick return if possible.
!
IF ((N.EQ.0) .OR. (((ALPHA.EQ.ZERO).OR.&
(K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN
!
! And when alpha.eq.zero.
!
IF (ALPHA.EQ.ZERO) THEN
  IF (UPPER) THEN
    IF (BETA.EQ.DBLE(ZERO)) THEN
      DO 20 J = 1,N
        DO 10 I = 1,J
          C(I,J) = ZERO

```

```

10         CONTINUE
20     CONTINUE
    ELSE
        DO 40 J = 1,N
            DO 30 I = 1,J - 1
                C(I,J) = BETA*C(I,J)
30         CONTINUE
            C(J,J) = BETA*DBLE(C(J,J))
40     CONTINUE
    END IF
    ELSE
        IF (BETA.EQ.DBLE(ZERO)) THEN
            DO 60 J = 1,N
                DO 50 I = J,N
                    C(I,J) = ZERO
50         CONTINUE
60     CONTINUE
        ELSE
            DO 80 J = 1,N
                C(J,J) = BETA*DBLE(C(J,J))
                DO 70 I = J + 1,N
                    C(I,J) = BETA*C(I,J)
70         CONTINUE
80     CONTINUE
        END IF
    END IF
    RETURN
END IF

!
! Start the operations.
!
IF (LSAME(TRANS,'N')) THEN
!
! Form C := alpha*A*B**H + conjg( alpha )*B*A**H +
! C.
!
    IF (UPPER) THEN
        DO 130 J = 1,N
            IF (BETA.EQ.DBLE(ZERO)) THEN
                DO 90 I = 1,J
                    C(I,J) = ZERO
90         CONTINUE
            ELSE IF (BETA.NE.ONE) THEN
                DO 100 I = 1,J - 1
                    C(I,J) = BETA*C(I,J)
100        CONTINUE
                C(J,J) = BETA*DBLE(C(J,J))
            ELSE
                C(J,J) = DBLE(C(J,J))
            END IF
            DO 120 L = 1,K
                IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                    TEMP1 = ALPHA*DCONJG(B(J,L))
                    TEMP2 = DCONJG(ALPHA*A(J,L))
                    DO 110 I = 1,J - 1
                        C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                            B(I,L)*TEMP2
110        CONTINUE
                        C(J,J) = DBLE(C(J,J)) +&
                            DBLE(A(J,L)*TEMP1+B(J,L)*TEMP2)
                    END IF
                CONTINUE
120        CONTINUE
130    CONTINUE
        ELSE
            DO 180 J = 1,N
                IF (BETA.EQ.DBLE(ZERO)) THEN
                    DO 140 I = J,N
                        C(I,J) = ZERO
140        CONTINUE
                ELSE IF (BETA.NE.ONE) THEN
                    DO 150 I = J + 1,N
                        C(I,J) = BETA*C(I,J)
150        CONTINUE
                C(J,J) = BETA*DBLE(C(J,J))
            ELSE
                C(J,J) = DBLE(C(J,J))
            END IF
            DO 170 L = 1,K
                IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                    TEMP1 = ALPHA*DCONJG(B(J,L))
                    TEMP2 = DCONJG(ALPHA*A(J,L))
                    DO 160 I = J + 1,N
                        C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                            B(I,L)*TEMP2
160        CONTINUE
                        C(J,J) = DBLE(C(J,J)) +&
                            DBLE(A(J,L)*TEMP1+B(J,L)*TEMP2)
                    END IF
                CONTINUE
170        CONTINUE
180    CONTINUE
        END IF
    ELSE
!
! Form C := alpha*A**H*B + conjg( alpha )*B**H*A +
! C.
!
        IF (UPPER) THEN
            DO 210 J = 1,N
                DO 200 I = 1,J
                    TEMP1 = ZERO
                    TEMP2 = ZERO
                    DO 190 L = 1,K
                        TEMP1 = TEMP1 + DCONJG(A(L,I))*B(L,J)
                        TEMP2 = TEMP2 + DCONJG(B(L,I))*A(L,J)
190        CONTINUE
                    IF (I.EQ.J) THEN
                        IF (BETA.EQ.DBLE(ZERO)) THEN
                            C(J,J) = DBLE(ALPHA*TEMP1+&
                                DCONJG(ALPHA)*TEMP2)
                        ELSE
                            C(J,J) = BETA*DBLE(C(J,J)) +&
                                DBLE(ALPHA*TEMP1+&
                                    DCONJG(ALPHA)*TEMP2)
                        END IF
                    END IF
                END IF
            END IF
        END IF
    END IF

```

```

                ELSE
                    IF (BETA.EQ.DBLE(ZERO)) THEN
                        C(I,J) = ALPHA*TEMP1 + DCONJG(ALPHA)*TEMP2
                    ELSE
                        C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
                            DCONJG(ALPHA)*TEMP2
                    END IF
                END IF
200         CONTINUE
210     CONTINUE
        ELSE
            DO 240 J = 1,N
                DO 230 I = J,N
                    TEMP1 = ZERO
                    TEMP2 = ZERO
                    DO 220 L = 1,K
                        TEMP1 = TEMP1 + DCONJG(A(L,I))*B(L,J)
                        TEMP2 = TEMP2 + DCONJG(B(L,I))*A(L,J)
220                 CONTINUE
                        IF (I.EQ.J) THEN
                            IF (BETA.EQ.DBLE(ZERO)) THEN
                                C(J,J) = DBLE(ALPHA*TEMP1+&
                                    DCONJG(ALPHA)*TEMP2)
                            ELSE
                                C(J,J) = BETA*DBLE(C(J,J)) +&
                                    DBLE(ALPHA*TEMP1+&
                                        DCONJG(ALPHA)*TEMP2)
                            END IF
                        ELSE
                            IF (BETA.EQ.DBLE(ZERO)) THEN
                                C(I,J) = ALPHA*TEMP1 + DCONJG(ALPHA)*TEMP2
                            ELSE
                                C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
                                    DCONJG(ALPHA)*TEMP2
                            END IF
                        END IF
                    END IF
                END IF
230         CONTINUE
240     CONTINUE
        END IF
    END IF
!
    RETURN
!
!   End of ZHER2K.
!
!   END SUBROUTINE
!> \brief \b ZSYR2K
!
!   ===== DOCUMENTATION =====
!
!   Online html documentation available at
!   http://www.netlib.org/lapack/explore-html/
!
!   Definition:
!   =====
!
!   SUBROUTINE ZSYR2K(UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
!   .. Scalar Arguments ..
!   COMPLEX*16 ALPHA,BETA
!   INTEGER K,LDA,LDB,LDC,N
!   CHARACTER TRANS,UPLO
!
!   .. Array Arguments ..
!   COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
!   ..
!
!> \par Purpose:
!   =====
!> \verbatim
!> ZSYR2K performs one of the symmetric rank 2k operations
!>
!>   C := alpha*A*B**T + alpha*B*A**T + beta*C,
!>
!> or
!>
!>   C := alpha*A**T*B + alpha*B**T*A + beta*C,
!>
!> where alpha and beta are scalars, C is an n by n symmetric matrix
!> and A and B are n by k matrices in the first case and k by n
!> matrices in the second case.
!> \endverbatim
!
!   Arguments:
!   =====
!> \param[in] UPLO
!> \verbatim
!> UPLO is CHARACTER*1
!> On entry, UPLO specifies whether the upper or lower
!> triangular part of the array C is to be referenced as
!> follows:
!>
!>   UPLO = 'U' or 'u' Only the upper triangular part of C
!>                     is to be referenced.
!>
!>   UPLO = 'L' or 'l' Only the lower triangular part of C
!>                     is to be referenced.
!> \endverbatim
!> \param[in] TRANS
!> \verbatim
!> TRANS is CHARACTER*1
!> On entry, TRANS specifies the operation to be performed as
!> follows:
!>
!>   TRANS = 'N' or 'n' C := alpha*A*B**T + alpha*B*A**T +
!>                     beta*C.
!>
!>   TRANS = 'T' or 't' C := alpha*A**T*B + alpha*B**T*A +
!>                     beta*C.
!>

```

```

!> \endverbatim
!>
!> \param[in] N
!> \verbatim
!>     N is INTEGER
!>     On entry, N specifies the order of the matrix C. N must be
!>     at least zero.
!> \endverbatim
!>
!> \param[in] K
!> \verbatim
!>     K is INTEGER
!>     On entry with TRANS = 'N' or 'n', K specifies the number
!>     of columns of the matrices A and B, and on entry with
!>     TRANS = 'T' or 't', K specifies the number of rows of the
!>     matrices A and B. K must be at least zero.
!> \endverbatim
!>
!> \param[in] ALPHA
!> \verbatim
!>     ALPHA is COMPLEX*16
!>     On entry, ALPHA specifies the scalar alpha.
!> \endverbatim
!>
!> \param[in] A
!> \verbatim
!>     A is COMPLEX*16 array of DIMENSION ( LDA, ka ), where ka is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array A must contain the matrix A, otherwise
!>     the leading k by n part of the array A must contain the
!>     matrix A.
!> \endverbatim
!>
!> \param[in] LDA
!> \verbatim
!>     LDA is INTEGER
!>     On entry, LDA specifies the first dimension of A as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDA must be at least max( 1, n ), otherwise LDA must
!>     be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] B
!> \verbatim
!>     B is COMPLEX*16 array of DIMENSION ( LDB, kb ), where kb is
!>     k when TRANS = 'N' or 'n', and is n otherwise.
!>     Before entry with TRANS = 'N' or 'n', the leading n by k
!>     part of the array B must contain the matrix B, otherwise
!>     the leading k by n part of the array B must contain the
!>     matrix B.
!> \endverbatim
!>
!> \param[in] LDB
!> \verbatim
!>     LDB is INTEGER
!>     On entry, LDB specifies the first dimension of B as declared
!>     in the calling (sub) program. When TRANS = 'N' or 'n'
!>     then LDB must be at least max( 1, n ), otherwise LDB must
!>     be at least max( 1, k ).
!> \endverbatim
!>
!> \param[in] BETA
!> \verbatim
!>     BETA is COMPLEX*16
!>     On entry, BETA specifies the scalar beta.
!> \endverbatim
!>
!> \param[in,out] C
!> \verbatim
!>     C is COMPLEX*16 array of DIMENSION ( LDC, n ).
!>     Before entry with UPLO = 'U' or 'u', the leading n by n
!>     upper triangular part of the array C must contain the upper
!>     triangular part of the symmetric matrix and the strictly
!>     lower triangular part of C is not referenced. On exit, the
!>     upper triangular part of the array C is overwritten by the
!>     upper triangular part of the updated matrix.
!>     Before entry with UPLO = 'L' or 'l', the leading n by n
!>     lower triangular part of the array C must contain the lower
!>     triangular part of the symmetric matrix and the strictly
!>     upper triangular part of C is not referenced. On exit, the
!>     lower triangular part of the array C is overwritten by the
!>     lower triangular part of the updated matrix.
!> \endverbatim
!>
!> \param[in] LDC
!> \verbatim
!>     LDC is INTEGER
!>     On entry, LDC specifies the first dimension of C as declared
!>     in the calling (sub) program. LDC must be at least
!>     max( 1, n ).
!> \endverbatim
!>
!> Authors:
!> =====
!>
!> \author Univ. of Tennessee
!> \author Univ. of California Berkeley
!> \author Univ. of Colorado Denver
!> \author NAG Ltd.
!>
!> \date November 2011
!>
!> \ingroup complex16_blas_level3
!>
!> \par Further Details:
!> =====
!>
!> \verbatim
!>
!>     Level 3 Blas routine.
!>
!>     -- Written on 8-February-1989.
!>     Jack Dongarra, Argonne National Laboratory.

```

```

!>      Iain Duff, AERE Harwell.
!>      Jeremy Du Croz, Numerical Algorithms Group Ltd.
!>      Sven Hammarling, Numerical Algorithms Group Ltd.
!> \endverbatim
!>
! =====
! SUBROUTINE ZSYR2K(UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
!
! -- Reference BLAS level3 routine (version 3.4.0) --
! -- Reference BLAS is a software package provided by Univ. of Tennessee, --
! -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
! November 2011
!
! .. Scalar Arguments ..
! COMPLEX*16 ALPHA,BETA
! INTEGER K,LDA,LDB,LDC,N
! CHARACTER TRANS,UPLO
!
! ..
! .. Array Arguments ..
! COMPLEX*16 A(LDA,*),B(LDB,*),C(LDC,*)
! ..
! =====
!
! .. External Functions ..
! LOGICAL LSAME
! EXTERNAL LSAME
! ..
! .. External Subroutines ..
! EXTERNAL XERBLA
! ..
! .. Intrinsic Functions ..
! INTRINSIC MAX
! ..
! .. Local Scalars ..
! COMPLEX*16 TEMP1,TEMP2
! INTEGER I,INFO,J,L,NROWA
! LOGICAL UPPER
! ..
! .. Parameters ..
! COMPLEX*16 ONE
! PARAMETER (ONE= (1.0D+0,0.0D+0))
! COMPLEX*16 ZERO
! PARAMETER (ZERO= (0.0D+0,0.0D+0))
! ..
!
! Test the input parameters.
!
! IF (LSAME(TRANS,'N')) THEN
!     NROWA = N
! ELSE
!     NROWA = K
! END IF
! UPPER = LSAME(UPLO,'U')
!
! INFO = 0
! IF ((.NOT.UPPER) .AND. (.NOT.LSAME(UPLO,'L'))) THEN
!     INFO = 1
! ELSE IF ((.NOT.LSAME(TRANS,'N')) .AND. &
!     (.NOT.LSAME(TRANS,'T'))) THEN
!     INFO = 2
! ELSE IF (N.LT.0) THEN
!     INFO = 3
! ELSE IF (K.LT.0) THEN
!     INFO = 4
! ELSE IF (LDA.LT.MAX(1,NROWA)) THEN
!     INFO = 7
! ELSE IF (LDB.LT.MAX(1,NROWA)) THEN
!     INFO = 9
! ELSE IF (LDC.LT.MAX(1,N)) THEN
!     INFO = 12
! END IF
! IF (INFO.NE.0) THEN
!     CALL XERBLA('ZSYR2K',INFO)
!     RETURN
! END IF
!
! Quick return if possible.
!
! IF ((N.EQ.0) .OR. ((ALPHA.EQ.ZERO).OR.&
!     (K.EQ.0)).AND. (BETA.EQ.ONE))) RETURN
!
! And when alpha.eq.zero.
!
! IF (ALPHA.EQ.ZERO) THEN
!     IF (UPPER) THEN
!         IF (BETA.EQ.ZERO) THEN
!             DO 20 J = 1,N
!                 DO 10 I = 1,J
!                     C(I,J) = ZERO
! 10                 CONTINUE
! 20             CONTINUE
!         ELSE
!             DO 40 J = 1,N
!                 DO 30 I = 1,J
!                     C(I,J) = BETA*C(I,J)
! 30                 CONTINUE
! 40             CONTINUE
!         END IF
!     ELSE
!         IF (BETA.EQ.ZERO) THEN
!             DO 60 J = 1,N
!                 DO 50 I = J,N
!                     C(I,J) = ZERO
! 50                 CONTINUE
! 60             CONTINUE
!         ELSE
!             DO 80 J = 1,N
!                 DO 70 I = J,N
!                     C(I,J) = BETA*C(I,J)
! 70                 CONTINUE
! 80             CONTINUE
!         END IF
!     END IF
! END IF

```

```

        RETURN
    END IF
!
!   Start the operations.
!
    IF (LSAME(TRANS,'N')) THEN
!
!       Form C := alpha*A*B**T + alpha*B*A**T + C.
!
        IF (UPPER) THEN
            DO 130 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 90 I = 1,J
                        C(I,J) = ZERO
                    CONTINUE
                ELSE IF (BETA.NE.ONE) THEN
                    DO 100 I = 1,J
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                END IF
                DO 120 L = 1,K
                    IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                        TEMP1 = ALPHA*B(J,L)
                        TEMP2 = ALPHA*A(J,L)
                        DO 110 I = 1,J
                            C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                                B(I,L)*TEMP2
                        CONTINUE
                    END IF
                CONTINUE
            CONTINUE
        CONTINUE
        ELSE
            DO 180 J = 1,N
                IF (BETA.EQ.ZERO) THEN
                    DO 140 I = J,N
                        C(I,J) = ZERO
                    CONTINUE
                ELSE IF (BETA.NE.ONE) THEN
                    DO 150 I = J,N
                        C(I,J) = BETA*C(I,J)
                    CONTINUE
                END IF
                DO 170 L = 1,K
                    IF ((A(J,L).NE.ZERO) .OR. (B(J,L).NE.ZERO)) THEN
                        TEMP1 = ALPHA*B(J,L)
                        TEMP2 = ALPHA*A(J,L)
                        DO 160 I = J,N
                            C(I,J) = C(I,J) + A(I,L)*TEMP1 +&
                                B(I,L)*TEMP2
                        CONTINUE
                    END IF
                CONTINUE
            CONTINUE
        CONTINUE
    END IF
    ELSE
!
!       Form C := alpha*A**T*B + alpha*B**T*A + C.
!
        IF (UPPER) THEN
            DO 210 J = 1,N
                DO 200 I = 1,J
                    TEMP1 = ZERO
                    TEMP2 = ZERO
                    DO 190 L = 1,K
                        TEMP1 = TEMP1 + A(L,I)*B(L,J)
                        TEMP2 = TEMP2 + B(L,I)*A(L,J)
                    CONTINUE
                    IF (BETA.EQ.ZERO) THEN
                        C(I,J) = ALPHA*TEMP1 + ALPHA*TEMP2
                    ELSE
                        C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
                            ALPHA*TEMP2
                    END IF
                CONTINUE
            CONTINUE
        CONTINUE
        ELSE
            DO 240 J = 1,N
                DO 230 I = J,N
                    TEMP1 = ZERO
                    TEMP2 = ZERO
                    DO 220 L = 1,K
                        TEMP1 = TEMP1 + A(L,I)*B(L,J)
                        TEMP2 = TEMP2 + B(L,I)*A(L,J)
                    CONTINUE
                    IF (BETA.EQ.ZERO) THEN
                        C(I,J) = ALPHA*TEMP1 + ALPHA*TEMP2
                    ELSE
                        C(I,J) = BETA*C(I,J) + ALPHA*TEMP1 +&
                            ALPHA*TEMP2
                    END IF
                CONTINUE
            CONTINUE
        CONTINUE
    END IF
    END IF
!
!   RETURN
!
!   End of ZSYR2K.
!
END SUBROUTINE

end program

```