AD-A261 511

12

NSWCDD/TR-92/425

NSWC LIBRARY OF MATHEMATICS SUBROUTINES

BY ALFRED H. MORRIS, JR.
STRATEGIC AND SPACE SYSTEMS DEPARTMENT

JANUARY 1993

DTIC
ELECTE
FEB 25 1993
S D
B

Approved for public release; distribution unlimited.

93-03904

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION
Dahlgren, Virginia 22448-5000

REPRODUCED BY
U.S. DEPARTMENT OF COMMERCE
NATIONAL TECHNICAL
INFORMATION SERVICE
SPRINGFIELD, VA 22161

NSWCDD/TR-92/425

NSWC LIBRARY OF MATHEMATICS SUBROUTINES

BY ALFRED H. MORRIS, JR.
STRATEGIC AND SPACE SYSTEMS DEPARTMENT

JANUARY 1993

Approved for public release; distribution unlimited.

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION
Dahlgren, Virginia 22448-5000

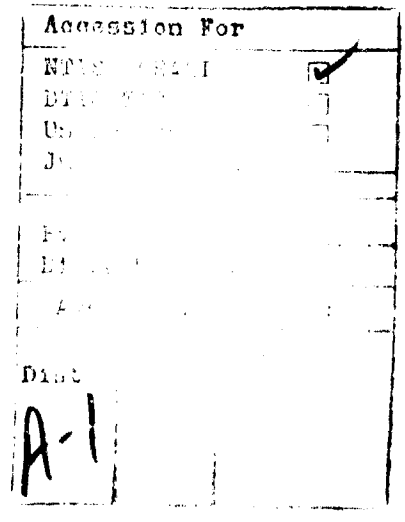
FOREWORD

In 1976 development of the NSWC library of general purpose numerical mathematics subroutines began. Since that time six editions of the library have been released for general use. This report describes the subroutines in the 1993 edition of the library, the seventh edition. The report supersedes NSWC TR 90–21 (1990). The development of the library is funded by the Computing Systems and Networks Division, Strategic and Space Systems Department, NSWCDD.

Approved by:

R. L. Schmidt

R. L. SCHMIDT, Head
Strategic and Space Systems
Department



ABSTRACT

The NSWC library is a library of general purpose Fortran subroutines that provide a basic computational capability for a variety of mathematical activities. Emphasis has been placed on the transportability of the codes. Subroutines are available in the following areas: elementary operations, geometry, special functions, polynomials, vectors, matrices, large dense systems of linear equations, banded matrices, sparse matrices, eigenvalues and eigenvectors, ℓ_1 solution of linear equations, least-squares solution of linear equations, optimization, transforms, approximation of functions, curve fitting, surface fitting, manifold fitting, numerical integration, integral equations, ordinary differential equations, partial differential equations, and random number generation.

CONTENTS

Page

Introduction 1

Elementary Operations

Machine Constants – SPMPAR,DPMPAR,IPMPAR 3
Argument Bounds for the Exponential Function – EPSLN,EXPARG,DEPSLN,DXPARG 5
Sorting Lists – ISHELL,SHELL,AORD,RISORT,SHELL2,DSORT, DAORD,DISORT,DDSORT,QSORTI,QSORTR,QSORTD,IORDER, RORDER,DORDER 7
Cube Root – CBRT,DCBRT 11
Four Quadrant Arctangent – ARTNQ,DARTNQ 11
Length of a Two Dimensional Vector – CPABS,DCPABS 11
Reciprocal of a Complex Number – CREC,DCREC 13
Division of Complex Numbers – CDVI,DIVID 13
Square Root of a Double Precision Complex Number – DCSQRT 13
Conversion of Polar to Cartesian Coordinates – POCA 15
Conversion of Cartesian to Polar Coordinates – CAPO 15
Rotation of Axes – ROTA 15
Planar Givens Rotations – SROTG,DROTG 17
Three Dimensional Rotations – ROT3 19
Rotation of a Point on the Unit Sphere to the North Pole – CONSTR 21
Computation of the Angle Between Two Vectors – ANG 23
Trigonometric Functions – SIN1,COS1,DSIN1,DCOS1 25
Hyperbolic Sine and Cosine Functions – SNHCSH 27
Exponentials – REXP,DREXP 29
Logarithms – ALNREL,RLOC,RLOG1,DLNREL,DRLOG,DRLOG1 31

Geometry

Determining if a Point is Inside or Outside a Polygon – LOCPT 33
Intersection of a Straight Line and Polygonal Path – PFIND 35
The Convex Hull for a Finite Planar Set – HULL 37
Areas of Planar Polygons – PAREA 39
Hamiltonian Circuits – HC 41

Special Functions

Error Function – CERF,CERFC,ERF,ERFC,ERFC1,DCERF,DCERFC, DERF,DERFC,DERFC1 45
Inverse Error Function – ERFI,DERFI 51
Difference of Error Functions – AERF,DAERF 53
Normal Probability Distribution Function – PNDF 55
Inverse Normal Probability Distribution Function – PNI,DPNI 57
Dawson's Integral – DAW,DPDAW 59

Complex Fresnel Integral – CFRNLI 61
Real Fresnel Integrals – FRNL 63
Exponential Integral Function – CEXPLI,EXPLI,DEI,DEI1 65
Sine and Cosine Integral Functions – SI,CIN 69
Exponential Exponential Integral Function – CEXEXI 71
Dilogarithm Function – CLI,ALI 73
Gamma Function – CGAMMA,GAMMA,GAMLN,DCGAMA, DGAMMA,DGAMLN 75
Digamma Function – CPSI,PSI,DCPSI,DPSI 79
Derivatives of the Digamma Function – PSIDF 81
Incomplete Gamma Ratio Functions – GRATIO,RCOMP,DGRAT DRCOMP 83
Inverse Incomplete Gamma Ratio Function – GAMINV,DGINV 85
Logarithm of the Beta Function – BETALN,DBETLN 87
Incomplete Beta Function – BRATIO,ISUBX,BRCOMP 89
Bessel Function $J_\nu(z)$ – CBSSLJ,BSSLJ,BESJ 91
Bessel Function $Y_\nu(z)$ – BSSLY 93
Modified Bessel Function $I_\nu(z)$ – CBSSLI,BSSLI,BESI 95
Modified Bessel Function $K_\nu(z)$ – CBESK,CBSSLK,BSSLK 97
Airy functions – CAI,CBI,AI,AIE,BI,BIE 99
Complete Complex Elliptic Integrals of the First and Second Kinds – CK,CKE 103
Real Elliptic Integrals of the First and Second Kinds – ELLPI,RFVAL,RDVAL,DELLPI,DRFVAL,DRDVAL 107
Real Elliptic Integrals of the Third Kind – EPI,RJVAL, DEPI,DRJVAL 111
Jacobian Elliptic Functions – ELLPF,ELPFC1 115
Weierstrass Elliptic Function for the Equianharmonic and Lemniscatic Cases – PEQ,PEQ1,PLEM,PLEM1 119
Integral of the Bivariate Density Function over Arbitrary Polygons and Semi-infinite Angular Regions – VALR2 123
Circular Coverage Function – CIRCV 125
Elliptical Coverage Function – PKILL 127

Polynomials

Copying Polynomials – PLCOPY,DPCOPY 129
Addition of Polynomials – PADD,DPADD 131
Subtraction of Polynomials – PSUBT,DPSUBT 133
Multiplication of Polynomials – PMULT,DPMULT 135
Division of Polynomials – PDIV,DPDIV 137
Real Powers of Polynomials – PLPWR,DPLPWR 139
Inverses of Power Series – PINV,DPINV 141
Derivatives and Integrals of Polynomials – MPLNMV 143
Evaluation of Chebyshev Expansions – CSEVL,DCSEVL 145
Lagrange Polynomials – LGRNGN,LGRNGV,LGRNGX 147
Orthogonal Polynomials on Finite Sets – ORTHOS,ORTHOV, ORTHOX 149

Solutions of Nonlinear Equations

Zeros of Continuous Functions – ZEROIN,DZERO 151
Solution of Systems of Nonlinear Equations – HBRD 153
Solutions of Quadratic, Cubic, and Quartic Equations –
QDCRT,CBCRT,QTCRT,DQDCRT,DCBCRT,DQTCRT 155
Double Precision Roots of Polynomials – DRPOLY,DCPOLY 157
Accuracy of the Roots of Polynomials – RBND,CBND 159

Vectors

Copying Vectors – SCOPY,DCOPY,CCOPY 161
Interchanging Vectors – SSWAP,DSWAP,CSWAP 163
Planar Rotation of Vectors – SROT,DROT,CSROT 165
Modified Givens Rotations – SROTMG,DROTMG,SROTM,DROTM 167
Dot Products of Vectors – SDOT,DDOT,CDOTC,CDOTU 171
Scaling Vectors – SSCAL,DSCAL,CSCAL,CSSCAL 173
Vector Addition – SAXPY,DAXPY,CAXPY 175
L_1 Norm of a Vector – SASUM,DASUM,SCASUM 177
L_2 Norm of a Vector – SNRM2,DNRM2,SCNRM2 179
L_∞ Norm of a Vector – ISAMAX,IDAMAX,ICAMAX 181

Matrices

Packing and Unpacking Symmetric Matrices – MCVFS,DMCVFS,
MCVSF,DMCVSF 183
Conversion of Real Matrices to and from Double Precision Form –
MCVRD,MCVDR 185
Storage of Real Matrices in the Complex Matrix Format – MCVRC 187
The Real and Imaginary Parts of a Complex Matrix –
CMREAL,CMIMAG 189
Copying Matrices – MCOPY,SMCOPY,DMCOPY,CMCOPY 191
Computation of the Conjugate of a Complex Matrix – CMCONJ 193
Transposing Matrices – TPOSE,DTPOSE,CTPOSE,TIP,DTIP,CTIP 195
Computing Adjoints of Complex Matrices – CMADJ,CTRANS 197
Matrix Addition – MADD,SMADD,DMADD,CMADD 199
Matrix Subtraction – MSUBT,SMSUBT,DMSUBT,CMSUBT 201
Matrix Multiplication – MTMS,DMTMS,CMTMS,MPROD,DMPROD,
CMPROD 203
Product of a Packed Symmetric Matrix and a Vector –
SVPRD,DSVPRD 205
Transpose Matrix Products – TMPROD 207
Symmetric Matrix Products – SMPROD 209
Kronecker Product of Matrices – KPROD,DKPROD, CKPROD 211
Rank of a Real Matrix – RNK,DRNK 213
Inverting General Real Matrices and Solving General
Systems of Real Linear Equations – CROUT,KROUT,
NPIVOT,MSLV,DMSLV,MSLV1,DMSLV1 215
Solution of Real Equations with Iterative Improvement – SLVMP 221

Solution of Almost Block Diagonal Systems of Linear Equations – ARCECO,ARCESL 223
Solution of Almost Block Tridiagonal Systems of Linear Equations – BTSLV 225
Inverting Symmetric Real Matrices and Solving Symmetric Systems of Real Linear Equations – SMSLV,DSMSLV 227
Inverting Positive Definite Symmetric Matrices and Solving Positive Definite Symmetric Systems of Linear Equations – PCHOL,DPCHOL 231
Solution of Toeplitz Systems of Linear Equations – TOPLX,DTOPLX 233
Inverting General Complex Matrices and Solving General Systems of Complex Linear Equations – CMSLV,CMSLV1, DCMSLV 235
Solution of Complex Equations with Iterative Improvement – CSLVMP 239
Singular Value Decomposition of a Matrix – SSVDC,DSVDC, CSVDC 241
Evaluation of the Characteristic Polynomial of a Matrix – DET,DPDET,CDET 243
Solution of the Matrix Equation $AX + XB = C$ – ABSLV,DABSLV 245
Solution of the Matrix Equation $A^tX + XA = C$ when C is Symmetric – TASLV,DTASLV 247
Solution of the Matrix Equation $AX^2 + BX + C = 0$ – SQUINT 249
Exponential of a Real Matrix – MEXP,DMEXP 251

Large Dense Systems of Linear Equations

Solving systems of 200–400 Linear Equations – LE,DPLE,CLE 253

Banded Matrices

Band Matrix Storage 255
Conversion of Banded Matrices to and from the Standard Format – CVBR,CVBD,CVBC,CVRB,CVDB,CVCB,CVRB1, CVDB1,CVCB1 257
Conversion of Banded Matrices to and from Sparse Form – MCVBS,DMCVBS,CMCVBS,MCVSB,DMCVSB,CMCVSB 259
Conversion of Banded Real Matrices to and from Double Precision Form – BCVRD,BCVDR 261
The Real and Imaginary Parts of a Banded Complex Matrix – BREAL,BIMAG 263
Computing $A + Bi$ for Banded Real Matrices A and B – BCVRC 265
Transposing Banded Matrices – BPOSE,DBPOSE,CBPOSE 267
Addition of Banded Matrices – BADD,DBADD,CBADD 269
Subtraction of Banded Matrices – BSUBT,DBSUBT,CBSUBT 271
Multiplication of Banded Matrices – BPROD,DBPROD,CBPROD 273
Product of a Real Banded Matrix and Vector – BVPRD,BVPRD1, BTPRD,BTPRD1 275

Product of a Double Precision Banded Matrix and Vector – DBVPD,DBVPD1,DBTPD,DBTPD1 277
Product of a Complex Banded Matrix and Vector – CBVPD, CBVPD1,CBTPD,CBTPD1 279
L_1 Norm of a Real Banded Matrix – B1NRM,DB1NRM 281
L_∞ Norm of a Real Banded Matrix – BNRM,DBNRM 283
Solution of Banded Systems of Real Linear Equations – BSLV,BSLV1 285
Computation of the Condition Number of a Real Banded Matrix – B1CND 287
Double Precision Solution of Banded Systems of Real Linear Equations – DBSLV,DBSLV1 289
Computation of the Condition Number of a Double Precision Banded Matrix – DB1CND 291
Solution of Banded Systems of Complex Linear Equations – CBSLV,CBSLV1 293

Sparse Matrices

Storage of Sparse Matrices 295
Conversion of Sparse Matrices to and from the Standard Format – CVRS,CVDS,CVCS,CVSR,CVSD,CVSC 297
Conversion of Sparse Real Matrices to and from Double Precision Form – SCVRD,SCVDR 299
The Real and Imaginary Parts of a Sparse Complex Matrix – CSREAL,CSIMAG 301
Computing $A + Bi$ for Sparse Real Matrices A and B – SCVRC 303
Copying Sparse Matrices – RSCOPY,DSCOPY,CSCOPY 305
Computing Conjugates of Sparse Complex Matrices – SCONJ 307
Transposing Sparse Real Matrices – RPOSE,RPOSE1 309
Transposing Sparse Double Precision Matrices – DPOSE,DPOSE1 311
Transposing Sparse Complex Matrices – CPOSE,CPOSE1 313
Addition of Sparse Matrices – SADD,DSADD,CSADD 315
Subtraction of Sparse Matrices – SSUBT,DSSUBT,CSSUBT 317
Multiplication of Sparse Matrices – SPROD,DSPROD,CSPROD 319
Product of a Real Sparse Matrix and Vector – MVPRD,MVPRD1, MTPRD,MTPRD1 321
Product of a Double Precision Sparse Matrix and Vector – DVPRD,DVPRD1,DTPRD,DTPRD1 323
Product of a Complex Sparse Matrix and Vector – CVPRD, CVPRD1,CTPRD,CTPRD1 325
L_1 Norm of a Sparse Real Matrix – S1NRM,DS1NRM 327
L_∞ Norm of a Sparse Real Matrix – SNRM,DSNRM 329
Ordering the Rows of a Sparse Matrix by Increasing Length – SPORD 331
Reordering Sparse Matrices into Block Triangular Form – BLKORD 333

Solution of Sparse Systems of Real Linear Equations – SPSLV,RSLV,TSLV 335
Computation of the Condition Number of a Real Sparse Matrix – S1CND 339
Double Precision Solution of Sparse Systems of Real Linear Equations – DSPSLV,DSLV,DTSLV 341
Computation of the Condition Number of a Double Precision Sparse Matrix – DS1CND 345
Solution of Sparse Systems of Complex Linear Equations – CSPSLV,CSLV,CTSLV 347

Eigenvalues and Eigenvectors

Computation of Eigenvalues of General Real Matrices – EIG,EIG1 351
Computation of Eigenvalues and Eigenvectors of General Real Matrices – EIGV,EIGV1 353
Double Precision Computation of Eigenvalues of Real Matrices – DEIG 355
Double Precision Computation of Eigenvalues ai Eigenvectors of Real Matrices – DEIGV 357
Computation of Eigenvalues of Symmetric Real Matrices – SEIG,SEIG1 359
Computation of Eigenvalues and Eigenvectors of Symmetric Real Matrices – SEIGV,SEIGV1 361
Double Precision Computation of Eigenvalues of Symmetric Real Matrices – DSEIG 363
Double Precision Computation of Eigenvalues and Eigenvectors of Symmetric Real Matrices – DSEIGV 365
Computation of Eigenvalues of Complex Matrices – CEIG 367
Computation of Eigenvalues and Eigenvectors of Complex Matrices – CEIGV 369
Double Precision Computation of Eigenvalues of Complex Matrices – DCEIG 371
Double Precision Computation of Eigenvalues and Eigenvectors of Complex Matrices – DCEIGV 373

ℓ_1 Solution of Linear Equations

ℓ_1 Solution of Systems of Linear Equations with Equality and Inequality Constraints – CL1 375

Least Squares Solution of Linear Equations

Least Squares Solution of Systems of Linear Equations – LLSQ,LSQR,HFTI,HFTI2 377
Least Squares Solution of Overdetermined Systems of Linear Equations with Iterative Improvement – LLSQMP 383
Double Precision Least Squares Solution of Systems of Linear Equations – DLLSQ,DLSQR,DHFTI,DHFTI2 385

Least Squares Solution of Systems of Linear Equations with Equality and Inequality Constraints – LSEI 391
Least Squares Solution of Systems of Linear Equations with Equality and Nonnegativity Constraints – WNNLS 395
Least Squares Iterative Improvement Solution of Systems of Linear Equations with Equality Constraints – L2SLV 399
Iterative Least Squares Solution of Banded Linear Equations – BLSQ 403
Iterative Least Squares Solution of Sparse Linear Equations – SPLSQ,STLSQ 405

Optimization

Minimization of Functions of a Single Variable – FMIN 407
Minimization of Functions of *n* Variables – OPTF 409
Unconstrained Minimum of the Sum of Squares of Nonlinear Functions – LMDIFF 411
Linear Programming – SMPLX,SSPLX 413
The Assignment Problem – ASSGN 417
0–1 Knapsack Problem – MKP 419

Transforms

Inversion of the Laplace Transform – LAINV 421
Fast Fourier Transform – FFT,FFT1 425
Multivariate Fast Fourier Transform – MFFT,MFFT1 427
Discrete Cosine and Sine Transforms – COSQI,COSQB,COSQF, SINQB,SINQF 429

Approximation of Functions

Rational Minimax Approximation of Functions – CHEBY 433
L_p Approximation of Functions – ADAPT 435
Calculation of the Taylor Series of a Complex Analytic Function – CPSC,DCPSC 439

Curve Fitting

Linear Interpolation – TRP 443
Lagrange Interpolation – LTRP 445
Hermite Interpolation – HTRP 447
Conversion of Real Polynomials from Newton to Taylor Series Form – PCOEFF 449
Least Squares Polynomial Fit – PFIT 451
Weighted Least Squares Polynomial Fit – WPFIT 453
Cubic Spline Interpolation – CBSPL,SPLIFT 455
Weighted Least Squares Cubic Spline Fitting – SPFIT 457
Least Squares Cubic Spline Fitting with Equality and Inequality Constraints – CSPFIT 459
Cubic Spline Evaluation – SCOMP,SCOMP1,SCOMP2 461

Cubic Spline Evaluation and Differentiation – SEVAL, SEVAL1,SEVAL2 463
Integrals of Cubic Splines – CSINT,CSINT1,CSINT2 465
Periodic Cubic Spline Interpolation – PDSPL 467
Least Squares Periodic Cubic Spline Fitting – PDFIT 469
Periodic Cubic Spline Evaluation and Differentiation – PSCMP,PSEVL 471
N-Dimensional Cubic Spline Closed Curve Fitting – CSLOOP, LOPCMP,LOPDF 473
Spline under Tension Interpolation – CURV1 475
Spline under Tension Evaluation – CURV2 477
Differentiation and Integration of Splines under Tension – CURVD,CURVI 479
Two Dimensional Spline under Tension Curve Fitting – KURV1,KURV2 481
Two Dimensional Spline under Tension Closed Curve Fitting – KURVP1,KURVP2 483
Three Dimensional Spline under Tension Curve Fitting – QURV1,QURV2 485
B-Splines 487
Finding the Interval that Contains a Point – INTRVL 489
Evaluation and Differentiation of Piecewise Polynomials from their *B*-Spline Representations – BVAL 491
Evaluation of the Indefinite Integral of a Piecewise Polynomial from its *B*-Spline Representation – BVALI 493
Conversion of Piecewise Polynomials from *B*-Spline to Taylor Series Form – BSPP 495
Evaluation of Piecewise Polynomials from their Taylor Series Representation – PPVAL 497
Piecewise Polynomial Interpolation – BSTRP 499
Weighted Least Squares Piecewise Polynomial Fitting – BSLSQ 501
Least Squares Piecewise Polynomial Fitting with Equality and Inequality Constraints – BFIT 503

Surface Fitting over Rectangular Grids

Bicubic Splines and Bisplines under Tension 505
Weighted Least Squares Bicubic Spline Fitting – SPFIT2 507
Evaluation and Differentiation of Bicubic Splines CSURF,CSURF1,CSRF,CSRF2 509
Bispline under Tension Surface Interpolation – SURF 513
Bispline under Tension Evaluation – SURF2,NSURF2 515
Bivariate B-Spline Piecewise Polynomial Interpolation – BSTRP2 517
Bivariate B-Spline Piecewise Polynomial Least Squares Fitting – BSLSQ2 519
Evaluation and Differentiation of Bivariate Piecewise Polynomials from their B-Spline Representation – BVAL2 521

Surface Fitting over Arbitrarily Positioned Data Points

Surface Interpolation for Arbitrarily Positioned Data Points – TRMESH,GRADG,GRADL,SFVAL,SFVAL2 523

Manifold Fitting

Weighted Least Squares Fitting with Polynomials of n Variables – MFIT,DMFIT,MEVAL,DMEVAL 527

Numerical Integration

Evaluation of Integrals over Finite Intervals – QAGS, QXGS,QSUBA,DQAGS,DQXGS 531
Evaluation of Integrals over Infinite Intervals – QAGI,DQAGI 539
Evaluation of Double Integrals over Triangles – CUBTRI 543

Integral Equations

Solution of Fredholm Integral Equations of the Second Kind – IESLV 545

Ordinary Differential Equations/Initial Value Problems

The Initial Value Solvers – Introductory Comments 549
Adaptive Adams Solution of Nonstiff Differential Equations – ODE 551
Adaptive Block RKF Solution of Nonstiff Differential Equations – BRKF45 555
Adaptive RKF Solution of Nonstiff Differential Equations – RKF45 559
Adaptive RKF Solution of Nonstiff Differential Equations with Global Error Estimation – GERK 563
Adaptive Solution of Stiff Differential Equations – SFODE,SFODE1 567
Fourth-Order Runge-Kutta – RK 571
Eighth-Order Runge-Kutta – RK8 573

Partial Differential Equations

Separable Second-Order Elliptic Equations on Rectangular Domains – SEPDE 575

Discrete Random Number Generation

Uniform Random Selection of Values from a Finite Set of Integers – URGET 579

Continuous Random Number Generation

Uniform Random Number Generator – URNG,DURNG 581
Generating Points Uniformly in a Square – URNG2,DURNG2 583
Generating Points Uniformly in a Circle – RCIR,DRCIR 585
Normal Random Number Generator – RNOR,DRNOR,NRNG,DNRNG 587
Multivariate Normal Random Vector Generator – NRVG,DNRVG NRVG1,DNRVG1 589
Exponential Random Number Generator – RANEXP,DRNEXP 593
Gamma Random Number Generator and the Chi-Square Distribution – RGAM,DRGAM 595

Beta Random Number Generator – RBETA,DRBETA 597
F-Distribution Random Number Generator – FRAN,DFRAN 599
Student t-Distribution Random Number Generator – TRAN,DTRAN 601
First Order Markov Random Number Generator – RMK1,DRMK1 603

Appendix. Installation of the NSWC Library and Conversion of
Codes from Single to Double Precision Form..... 605

Index 607

INTRODUCTION

In 1976 development of the NSWC library began. The objective was to form a library of general purpose Fortran subroutines that would provide a basic computational capability for a variety of mathematical activities. Even though the subroutines were intended for use on the CDC 6000–7000 series computers, emphasis was placed on their transportability. Currently, the library is used on a variety of computers, ranging from supercomputers such as the Cray Y-MP to personal computers such as the IBM PC. A brief appendix is included, which provides information for installing the library on all computers.

The subroutines in the library originate from a variety of sources. Approximately 40% of the subroutines were developed at NSWCDD. The remaining subroutines are from government, commercial, and university research centers in the U.S. and abroad. The 1993 edition of the library contains 1060+ subroutines. This report describes the 576 subroutines that are intended for general use. The remaining subroutines are supportive, normally being of little interest to most users. The library contains single and double precision subroutines. The single precision subroutines are designed for single precision floating arithmetics which have 6–14 digits of accuracy, and the double precision subroutines are designed for double precision arithmetics which have 12–30 digits of accuracy.

All subroutines are thoroughly examined and tested before being accepted for the library. Primary considerations are the reliability and transportability of the subroutine, its efficiency, and its ease of use. The subroutines in the library are always subject to reexamination and possible modification. If a subroutine becomes obsolete, then the subroutine may be eliminated.

The major issues concerning reliability are accuracy, the stability and robustness of the algorithm being used, and the overall quality of the code. Testing is performed for determining the accuracy and efficiency of the subroutine, checking for defects in the code, searching for regions of numerical instability, and checking the robustness of the algorithm. In most cases the testing must be highly selective, being used along with an examination of the algorithm and code. After the algorithm and code have been examined and testing is finished, an assessment is made of the overall performance of the code.

In regard to transportability, it is clear that machine dependent constants and precision dependent algorithms cannot be avoided. However, machine dependent code is not permitted. In order for a code to be acceptable, it is required that the code satisfy the 1977 ANSI Fortran standard. The code of a subroutine is examined for illegal Fortran constructs, detection of operations that are known to be error prone when not used with extreme care, location of global variables or ENTRY statements (which are not permitted), and checking for the use of EQUIVALENCE statements and common blocks (which are allowed, but with extreme disapproval). Variables that are defined by DATA statements are never permitted to be redefined. (The 1977 Fortran standard permits such a variable to be redefined by an assignment statement. However, when the subroutine terminates the variable then becomes undefined.)

The ease of use criterion is of considerable importance. The main purpose of the library is to provide a service to as broad an audience as is possible. Hence, it is important that the subroutines be as simple to use and comprehensive in scope as is practical, and that the subroutines do not unnecessarily restrict the user. The only requirement for the software that has a direct bearing o[illegible] issue involves the use of I/O. No I/O statements are permitted. If error detection [illegible]rmed in a function, then it is required that the function be assigned a special valu[illegible] the error occurs. (No arguments of a function in the library may be altered dur[illegible]pucation.) If error detection is performed by a subroutine that is not a function, [illegible]e call line of the subroutine must contain one or more parameters for reporting the er[illegible]. The use of such parameters allows the user to have total control over the sequence of [illegible]ents that follow.

If the precision of a subroutine [illegible]e library is established, then this information is given with the description of the sub[illegible]ine in the report. All precision estimates are for the CDC 6000–7000 series 14-digit si[illegible]precision and 28-digit double precision truncation arithmetics. ***The estimates do not [illegible]de inherent error***. Thus, the reported accuracy of a subroutine may be better or [illegible]han the inherent error of the function that it is computing.

Since the subroutines in the h[illegible]ary originate from a variety of sources, standards concerning in-line documentation and the style of code cannot be imposed. In general, all supportive subroutines not intended for direct use are not described in the report. This makes it possible to modify or replace the code without bothering the programmer. This capability is extremely important. During the last decade a vast amount of research has resulted in the development of new, more powerful algorithms for a variety of problems. Many of the results affect current codes, occasionally making some codes obsolete.

Since the beginning of the development of the library, no proprietary or otherwise restricted codes have been permitted in the library. Only general purpose mathematical subroutines for use by the entire NSWCDD scientific community have been [illegible]nsidered for the library. Restrictions on the use of any library can severely impair its v[illegible]ue, both for theoretical purposes (where the source codes are frequently of prime importance), and for general use in applications. Since expertise is so widely scattered, reliable codes are so very difficult to obtain, and many of the libraries do not provide source code, the policy has been to make the source code of the library readily available to the general scientific community.

MACHINE CONSTANTS

Assume that the integer arithmetic being used has base b, and that the integers are represented in the form

$$\pm(k_0 + k_1 b + \cdots + k_{n-1} b^{n-1})$$

where $k_0, k_1, \ldots, k_{n-1}$ are integers such that $0 \leq k_i < b$ $(i = 0, 1, \ldots, n-1)$. The value n is the number of base b digits k_i, and $b^n - 1$ and $-(b^n - 1)$ are the largest positive and negative integers that occur.

It is assumed that the single and double precision arithmetics being used have the same base, say β, and that the nonzero numbers can be represented in the form

$$\pm(\frac{k_1}{\beta} + \cdots + \frac{k_m}{\beta^m})\beta^\ell$$

where $k_1, \ldots, k_m$ are integers such that $0 \leq k_i < \beta$ $(i = 1, \ldots, m), k_1 \geq 1$, and ℓ is an integer such that $\ell_{\min} \leq \ell \leq \ell_{\max}$. The value m is the number of base β digits k_i, and $k_1 \geq 1$ is the requirement that the floating point numbers are normalized. The values $\ell_{\min}$ and $\ell_{\max}$ are the largest negative and positive exponents that arise where the floating point numbers maintain full m digit accuracy. Then $x_{\min} = \beta^{\ell_{\min}-1}$ is the smallest positive number that is represented and $x_{\max} = (1 - \beta^{-m})\beta^{\ell_{\max}}$ the largest.

Associated with m is the constant $\epsilon = \beta^{-m+1}$, called the ***relative precision*** of the floating point arithmetic being used. Theoretically, ϵ is the smallest number for which $1 + \epsilon$, when stored in memory, is stored exactly and has a value greater than 1. Normally ϵ will be the smallest number that satisfies these conditions. However, there do exist computer arithmetics (such as the CDC 6000–7000 series double precision arithmetics) for which this is not the case. In such arithmetics, some arithmetic results are able to be stored more accurately than others.

The functions SPMPAR, DPMPAR, and IPMPAR are available for obtaining the above constants. IPMPAR is the only subprogram in the library that is machine dependent.

SPMPAR(i)

SPMPAR is a real valued function. It is assumed that $i = 1, 2$, or 3. SPMPAR provides the following constants for the single precision arithmetic being used:

SPMPAR(1) = ϵ, the relative precision
SPMPAR(2) = $x_{\min}$, the smallest positive number
SPMPAR(3) = $x_{\max}$, the largest positive number

Programming. SPMPAR was written by A. H. Morris. The function IPMPAR is used.

DPMPAR(i)

DPMPAR is a double precision valued function. It is assumed that $i = 1, 2$, or 3. DPMPAR provides the following constants for the double precision arithmetic being used:

$$\begin{aligned}
&\text{DPMPAR}(1) = \epsilon, \text{the relative precision} \\
&\text{DPMPAR}(2) = x_{\min}, \text{the smallest positive number} \\
&\text{DPMPAR}(3) = x_{\max}, \text{the largest positive number}
\end{aligned}$$

DPMPAR must be declared in the calling program to be of type DOUBLE PRECISION.

Programming. DPMPAR was written by A. H. Morris. The function IPMPAR is used.

IPMPAR(i)

IPMPAR is an integer valued function. It is assumed that $i = 1, \ldots, 10$. IPMPAR provides the following constants:

Integer arithmetic

$$\begin{aligned}
&\text{IPMPAR}(1) = b, \text{ the base of the arithmetic} \\
&\text{IPMPAR}(2) = n, \text{ the number of base } b \text{ digits} \\
&\text{IPMPAR}(3) = b^n - 1, \text{ the largest integer}
\end{aligned}$$

Base for the floating arithmetics

$$\text{IPMPAR}(4) = \beta$$

Single precision arithmetic

$$\begin{aligned}
&\text{IPMPAR}(5) = m, \text{ the number of base } \beta \text{ digits} \\
&\text{IPMPAR}(6) = \ell_{\min}, \text{ the largest negative exponent} \\
&\text{IPMPAR}(7) = \ell_{\max}, \text{ the largest positive exponent}
\end{aligned}$$

Double precision arithmetic

$$\begin{aligned}
&\text{IPMPAR}(8) = m, \text{ the number of base } \beta \text{ digits} \\
&\text{IPMPAR}(9) = \ell_{\min}, \text{ the largest negative exponent} \\
&\text{IPMPAR}(10) = \ell_{\max}, \text{ the largest positive exponent}
\end{aligned}$$

Programming. IPMPAR is an adaptation by A. H. Morris of the function I1MACH, designed by P. A. Fox, A. D. Hall, and N. L. Schryer (Bell Laboratories). The constants for the various computers are from Bell Laboratories, NSWC, and other sources.

ARGUMENT BOUNDS FOR THE EXPONENTIAL FUNCTION

The functions EPSLN and EXPARG are available for obtaining the argument bounds for the exponential function EXP, and the functions DEPSLN and DXPARG are available for obtaining the argument bounds for DEXP.

EPSLN(ℓ)

If ϵ is the relative precision of the single precision floating arithmetic being used (i.e., if SPMPAR(1) = ϵ), then EPSLN(ℓ) is assigned the logarithm value $\ln \epsilon$. The argument ℓ is ignored.

Programming. EPSLN employs the function IPMPAR. EPSLN was written by A.H. Morris.

EXPARG(ℓ)

The argument ℓ may be an integer. If $\ell = 0$ then EXPARG(ℓ) = the largest positive value w for which EXP(w) can be computed. Otherwise, if $\ell \neq 0$ then EXPARG(ℓ) = the largest (in magnitude) negative value for which EXP(w) does not underflow.

Precision. EXPARG(ℓ) is accurate to within 1 unit of the 5^{th} significant digit for any single precision floating arithmetic, being slightly less in magnitude than the largest value w being approximated.

Programming. EXPARG employs the function IPMPAR. EXPARG was written by A.H. Morris.

DEPSLN(ℓ)

If ϵ is the relative precision of the double precision floating arithmetic being used (i.e., if DPMPAR(1) = ϵ), then DEPSLN(ℓ) is assigned the logarithm value $\ln \epsilon$. The argument ℓ is ignored.

Remark. DEPSLN must be declared in the calling program to be of type DOUBLE PRECISION.

Programming. DEPSLN employs the function IPMPAR. DEPSLN was written by A.H. Morris.

DXPARG(ℓ)

The argument ℓ may be any integer. If $\ell = 0$ then DXPARG(ℓ) = the largest positive double precision value w for which DEXP(w) can be computed. Otherwise, if $\ell \neq 0$ then DXPARG(ℓ) = the largest (in magnitude) negative double precision value for which DEXP(w) does not underflow.

Remark. DXPARG must be declared to be of type DOUBLE PRECISION in the calling

program.

Precision. DXPARG(ℓ) is accurate to within 1 unit of the 12^{th} significant digit for any double precision floating arithmetic, being slightly less in magnitude than the largest value being approximated.

Programming. DXPARG employs the function IPMPAR. DXPARG was written by A.H. Morris.

SORTING LISTS

Let A be an array containing $n \geq 1$ elements $a_1, \ldots, a_n$. Then the following subroutines are available for reordering the elements of A.

CALL ISHELL(A, n)

It is assumed that A is an integer array. When ISHELL is called, the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$.

Algorithm. The Shell sorting algorithm with increments $(3^k - 1)/2$ is employed.

Programmer. A. H. Morris

Reference. Knuth, D. E., ***The Art of Computer Programming. Vol. 3, Sorting and Searching***. Addison-Wesley, Reading, Mass., 1973, pp. 84–95.

CALL SHELL(A, n)
CALL AORD(A, n)

It is assumed that A is a real array. If SHELL is called, then the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. Otherwise, if AORD is called, then the elements of A are reordered so that $|a_i| \leq |a_{i+1}|$ for $i = 1, \ldots, n-1$.

Programmer. A. H. Morris

CALL RISORT(A, L, n)

It is assumed that A is a real array and L an integer array containing n elements. When RISORT is called, the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. The same permutations are performed on L as on A, thereby reordering the elements of L so as to correspond with the new ordering of A.

Programmer. A. H. Morris

CALL SHELL2(A, B, n)

It is assumed that A and B are real arrays containing n elements. When SHELL2 is called, the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. The same permutations are performed on B as on A, thereby reordering the elements of B so as to correspond with the new ordering of A.

Programmer. A. H. Morris

CALL DSORT(A, n)
CALL DAORD(A, n)

It is assumed that A is a double precision array. If DSORT is called, then the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. Otherwise, if DAORD is called, then the elements of A are reordered so that $|a_i| \leq |a_{i+1}|$ for $i = 1, \ldots, n-1$.

Programmer. A. H. Morris

CALL DISORT(A, L, n)

It is assumed that A is a double precision array and L an integer array containing n elements. When DISORT is called, the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. The same permutations are performed on L as on A, thereby reordering the elements of L so as to correspond with the new ordering of A.

Programmer. A. H. Morris

CALL DDSORT(A, B, n)

It is assumed that A and B are double precision arrays containing n elements. When DDSORT is called, the elements of A are reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$. The same permutations are performed on B as on A, thereby reordering the elements of B so as to correspond with the new ordering of A.

Programmer. A. H. Morris

CALL QSORTI(A, L, n)
CALL QSORTR(A, L, n)
CALL QSORTD(A, L, n)

QSORTI is used if A is an integer array, QSORTR is used if A is a real array, and QSORTD is used if A is a double precision array. It is assumed that $n \leq 2097152$ and that L is an integer array of dimension n or larger. When the subroutine is called, the indices $i_1, \ldots, i_n$ are stored in L where $a_{i_1} \leq \cdots \leq a_{i_n}$. A is not modified by the routine.

Remarks.

(1) After L has been obtained, A may be reordered so that $a_i \leq a_{i+1}$ for $i = 1, \ldots, n-1$ by IORDER, RORDER, or DORDER (see below).

(2) QSORTI, QSORTR, and QSORTD employ a quick sort procedure. These routines frequently take less than half the time required by the corresponding subroutines ISHELL, SHELL, and DSORT, which use a SHELL sort procedure.

Programmer. Robert Renka (Oak Ridge National Laboratory).

CALL IORDER(A, L, n)
CALL RORDER(A, L, n)
CALL DORDER(A, L, n)

IORDER is used if A is an integer array, RORDER is used if A is a real array, and DORDER is used if A is a double precision array. L is an integer array containing a

permutation $i_1, \ldots, i_n$ of $1, \ldots, n$. If A initially contains $a_1, \ldots, a_n$ then A contains the reordered sequence $a_{i_1}, \ldots, a_{i_n}$ when the subroutine terminates.

Programmer. Robert Renka (Oak Ridge National Laboratory).

CUBE ROOT

The following functions are available for computing the real cube root of a real number.

CBRT(x)
DCBRT(x)

CBRT is used if x is a single precision number, and DCBRT is used if x is a double precision number. CBRT is a single precision function and DCBRT a double precision function. The value of the function is $\sqrt[3]{x}$.

Remark. DCBRT must be declared in the calling program to be of type DOUBLE PRECISION.

Programmer. A. H. Morris

FOUR QUADRANT ARCTANGENT

The function ARTNQ is similar to the ATAN2 function, the differences being that its value lies in the interval $[0, 2\pi)$ and its value at the origin is 0. DARTNQ is the double precision counterpart of ARTNQ.

ARTNQ(y, x)
DARTNQ(y, x)

ARTNQ is used if x and y are single precision values, and DARTNQ is used if x and y are double precision values. ARTNQ is a single precision function and DARTNQ is a double precision function.

If (x, y) is a point in the plane other than the origin $(0, 0)$, let L denote the straight line connecting the points $(0, 0)$ and (x, y). Then the function is assigned the value θ where θ is the angle between L and the positive x-axis measured in a counterclockwise direction. Here $0 \leq \theta < 2\pi$. Otherwise, if (x, y) is the origin $(0, 0)$, then the function is assigned the value 0.

Remark. DARTNQ must be declared in the calling program to be of type DOUBLE PRECISION.

Programmer. Richard Pasto

LENGTH OF A TWO DIMENSIONAL VECTOR

The following functions are available for computing the length of a real vector (x, y).

CPABS(x, y)
DCPABS(x, y)

CPABS is used if x and y are single precision values, and DCPABS is used if x and y are double precision values. CPABS is a single precision function and DCPABS is a double precision function. The value of the function is $\sqrt{x^2 + y^2}$.

Remark. DCPABS must be declared in the calling program to be of type DOUBLE PRECISION.

Programmer. A. H. Morris

RECIPROCAL OF A COMPLEX NUMBER

The following subroutines are available for computing the reciprocal of a complex number.

CALL CREC(x, y, u, v)
CALL DCREC(x, y, u, v)

CREC is used if x and y are single precision real numbers and u and v real variables, and DCREC is used if x and y are double precision numbers and u and v double precision variables. It is assumed that x and y are the real and imaginary parts of a nonzero complex number z. When CREC or DCREC is called, u and v are set to the real and imaginary parts of $1/z$, respectively.

Programmer. A. H. Morris

DIVISION OF COMPLEX NUMBERS

The function CDIV and subroutine CDIVID are available for dividing complex numbers.

CDIV(a, b)

CDIV$(a, b) = a/b$ for any complex numbers a and b where $b \neq 0$.

Remark. CDIV must be declared in the calling program to be of type COMPLEX.

Programmer. A. H. Morris.

CALL CDIVID$(a_1, a_2, b_1, b_2, u, v)$

The arguments a_1, a_2, b_1, b_2 are double precision numbers, and u and v are double precision variables. It is assumed that a_1 and a_2 are the real and imaginary parts of a complex number a, and that b_1 and b_2 are the real and imaginary parts of a nonzero complex number b. When CDIVID is called, u and v are set to the real and imaginary parts of a/b respectively.

Programmer. A. H. Morris.

SQUARE ROOT OF A DOUBLE PRECISION COMPLEX NUMBER

The following subroutine is available for computing the square root of a double precision complex number.

CALL DCSQRT(Z, W)

Z and W are double precision arrays of dimension 2. It is assumed that $Z(1)$ and $Z(2)$ are the real and imaginary parts of a complex number z. When DCSQRT is called, if $z = 0$ then $W(1)$ and $W(2)$ are set to 0. Otherwise, if $z \neq 0$ then the square root $w = \sqrt{z}$ where $-\pi/2 < \arg(w) \leq \pi/2$ is computed and stored in W. $W(1)$ and $W(2)$ contain the real and imaginary parts of w, respectively.

Note. Z and W may reference the same storage area.

Programming. DCSQRT calls the function DCPABS. DCSQRT was written by A. H. Morris.

CONVERSION OF POLAR TO CARTESIAN COORDINATES

The following subroutine is available for converting polar coordinates (r, θ) to cartesian coordinates (x, y).

CALL POCA(r, θ, x, y)

Let (r, θ) be the polar coordinates of a point in the plane and let x, y be variables. When the routine is called, x and y are assigned the values $x = r \cos \theta$ and $y = r \sin \theta$.

CONVERSION OF CARTESIAN TO POLAR COORDINATES

The following subroutine is available for converting cartesian coordinates (x, y) to polar coordinates (r, θ).

CALL CAPO(x, y, r, θ)

Let (x, y) be the cartesian coordinates of a point in the plane and let r, θ be variables. If (x, y) is the origin then CAPO sets $r = \theta = 0$. Otherwise, if (x, y) is a point other than the origin, let L denote the straight line connecting the points $(0, 0)$ and (x, y). Then when CAPO is called, r is assigned the value $\sqrt{x^2 + y^2}$ and θ is defined to be the angle between L and the positive x axis. Here $-\pi < \theta \leq \pi$.

ROTATION OF AXES

Let (x_1, y_1) be the (cartesian) coordinates for a point in the plane. The following subroutine computes the new coordinates (x_2, y_2) for the point after the x, y axes have been rotated by an angle θ.

CALL ROTA$(x_1, y_1, \theta, x_2, y_2)$

The arguments x_2 and y_2 are variables. When ROTA is called, x_2 and y_2 are assigned the values:

$$x_2 = x_1 \cos \theta + y_1 \sin \theta$$
$$y_2 = -x_1 \sin \theta + y_1 \cos \theta$$

Programmer. A. H. Morris.

PLANAR GIVENS ROTATIONS

If a and b are real numbers where $a^2 + b^2 \neq 0$, then there is an orthogonal matrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ such that $\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$. In this case $r^2 = a^2 + b^2, c = a/r$, and $s = b/r$. The matrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ represents what is called a ***Givens rotation***. Given a and b, the matrix is uniquely defined up to the sign of r. For any real a, let $\mathrm{sgn}(a) = 1$ if $a \geq 0$ and $\mathrm{sgn}(a) = -1$ if $a < 0$. If we define $r = \sigma\sqrt{a^2 + b^2}$ where

$$\sigma = \begin{cases} \mathrm{sgn}(a) & \text{if } |a| > |b| \\ \mathrm{sgn}(b) & \text{if } |a| \leq |b| \end{cases}$$

then for $r \neq 0$ we note that $|c| > |s|$ implies $c > 0$, and that $|c| \leq |s|$ implies $s > 0$. For convenience, when $r = 0$ we set $c = 1$ and $s = 0$.

The value σ is not needed for the constuction of a Givens rotation matrix, but its use permits the representation of c and s by a single value z. For each c and s, z is defined as follows:

$$z = \begin{cases} s & \text{if } |s| < c \text{ or } c = 0 \\ 1/c & \text{if } 0 < |c| \leq s \end{cases}$$

The mapping $(c, s) \mapsto z$ is 1-1. If the user wishes to reconstruct c and s from z, then this can be done as follows:

If $z = 1$ then set $c = 0$ and $s = 1$.

If $|z| < 1$ then set $c = \sqrt{1 - z^2}$ and $s = z$.

If $|z| > 1$ then set $c = 1/z$ and $s = \sqrt{1 - c^2}$.

The subroutines SROTG and DROTG are available for computing c, s, r, and z. SROTG is used when a and b are single precision real numbers, and DROTG is used when a and b are double precision numbers.

CALL SROTG(AR,BZ,C,S)
CALL DROTG(AR,BZ,C,S)

When SROTG is used then AR, BZ, C, and S are real variables. Otherwise, when DROTG is used then AR, BZ, C, and S are double precision variables. On input, AR = a and BZ = b. When the routine terminates AR = r, BZ = z, $C = c$, and $S = s$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. SROTG and DROTG were written by Charles Lawson (Jet Propulsion Laboratory).

THREE DIMENSIONAL ROTATIONS

If $A = (a_{ij})$ is a 3×3 orthogonal matrix, then A can be represented in the form $A = R_3 R_2 R_1 E$ where

$$R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{pmatrix} \quad R_2 = \begin{pmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \pm 1 \end{pmatrix}.$$

R_1 represents a rotation around the x-axis, R_2 a rotation around the y-axis, and R_3 a rotation around the z-axis. Since A is orthogonal the determinant $\det(A) = \pm 1$. If $\det(A) = 1$ then E is the identity matrix and A is the combined rotation $R_3 R_2 R_1$. If $\det(A) = -1$ then A is composed of the rotation $R_3 R_2 R_1$ and the reflection $E = \mathrm{diag}(1, 1, -1)$. The following subroutine is available for finding the angles $\theta_1, \theta_2, \theta_3$ where $-\pi < \theta_1 \leq \pi$, $|\theta_2| \leq \pi/2$, and $-\pi < \theta_3 \leq \pi$.

CALL ROT3(A, THETA)

THETA is an array of dimension 3 or larger. When ROT3 is called, the angles $\theta_1, \theta_2, \theta_3$ are computed and stored in THETA.

Algorithm. If $a_{11} = a_{21} = 0$ then let $\theta_3 = 0$. Otherwise, let $\theta_3 = \mathrm{ATAN2}(a_{21}, a_{11})$. Then

$$R_3^t A = \begin{pmatrix} r_1 & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = A'$$

where $r_1 = \sqrt{a_{11}^2 + a_{21}^2}$. Also, if $\theta_2 = \mathrm{ATAN2}(a_{31}, r_1)$ then

$$R_2^t A' = \begin{pmatrix} r_2 & a''_{12} & a''_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{pmatrix} = A''$$

where $r_2 = \sqrt{r_1^2 + a_{32}^2}$. Since $r_2 \geq 0$, by orthogonality it follows that $r_2 = 1$ and $a''_{12} = a''_{13} = 0$. Finally, if $\theta_1 = \mathrm{ATAN2}(a'_{32}, a'_{22})$ then

$$R_1^t A'' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & r_3 & a''_{23} \\ 0 & 0 & a''_{33} \end{pmatrix}$$

where $r_3 = \sqrt{(a'_{22})^2 + (a'_{32})^2}$. Since $r_3 \geq 0$, by orthogonality we obtain $r_3 = 1$, $a''_{23} = 0$, and $a''_{33} = \pm 1$.

Programmer. A. H. Morris

ROTATION OF A POINT ON THE UNIT SPHERE TO THE NORTH POLE

Given the point (x, y, z) where $x^2 + y^2 + z^2 = 1$. Then there exist orthogonal matrices

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{pmatrix} \quad \text{and} \quad R_y = \begin{pmatrix} c_y & 0 & -s_y \\ 0 & 1 & 0 \\ s_y & 0 & c_y \end{pmatrix}$$

such that $R_y R_x \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. R_x represents a rotation about the x-axis and R_y a rotation about the y-axis. The following subroutine is available for obtaining the values c_x, s_x, c_y, s_y.

CALL CONSTR$(x, y, z,$ CX,SX,CY,SY)

CX,SX,CY,SY are variables. When CONSTR is called, these variables are assigned the values c_x, s_x, c_y, s_y.

Programmer. Robert J. Renka (Oak Ridge National Laboratory)

COMPUTATION OF THE ANGLE BETWEEN TWO VECTORS

Given the points $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ where $n \geq 2$, $x \neq 0$, and $y \neq 0$. Let L_x denote the straight line connecting 0 and x, and L_y the straight line connecting 0 and y. Then the function ANG is available for computing the angle ϕ between the two lines L_x and L_y.

ANG(n, X, Y)

X and Y are arrays of dimension n containing $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. ANG(n, X, Y) $= \phi$ where $0 \leq \phi \leq \pi$.

Error Return. ANG$(n, X, Y) = -1$ if $n < 2$, $x = 0$, or $y = 0$.

Programming. ANG employs the function SNRM2. ANG was written by A. H. Morris.

TRIGONOMETRIC FUNCTIONS

The following functions are available for computing $\sin \pi x$ and $\cos \pi x$ for real x.

SIN1(x)

SIN1$(x) = \sin \pi x$ for all real x.

Algorithm. Minimax approximations are used for $\sin \pi x$ and $\cos \pi x$ when $|x| \leq \pi/4$.

Precision. SIN1(x) is accurate to within 1 unit of the 14^{th} significant digit when $0 < x < 1$.

Programming. SIN1 was written by A.H. Morris. The function IPMPAR is used.

COS1(x)

COS1$(x) = \cos \pi x$ for all real x.

Algorithm. Minimax approximations are used for $\sin \pi x$ and $\cos \pi x$ when $|x| \leq \pi/4$.

Precision. COS1(x) is accurate to within 1 unit of the 14^{th} significant digit when $0 \leq x \leq 1$ and $x \neq 1/2$.

Programming. COS1 was written by A. H. Morris. The function IPMPAR is used.

DSIN1(x)

The argument x is a double precision number. DSIN1(x) is the double precision value for $\sin \pi x$.

Remark. DSIN1 must be declared to be of type DOUBLE PRECISION in the calling program.

Algorithm. For $|x| \leq \pi/4$, $\sin \pi x$ and $\cos \pi x$ are computed by the power series corresponding to the Chebyshev expansions developed by J. L. Schonfelder (University of Birmingham, England). The power series were obtained by A. H. Morris.

Precision. DSIN1(x) is accurate to within 1 unit of the 28^{th} significant digit when $0 < x < 1$.

Reference. Schonfelder, J.L.,"Very High Accuracy Chebyshev Expansions for the Basic Trigonometric Functions," ***Math Comp***. 34 (1980), pp. 237-244.

Programming. DSIN1 was written by A.H. Morris. The function IPMPAR is used.

DCOS1(x)

The argument x is a double precision number. DCOS1(x) is the double precision value for $\cos \pi x$.

Remark. DCOS1 must be declared to be of type DOUBLE PRECISION in the calling program.

Algorithm. For $|x| \leq \pi/4$, $\sin \pi x$ and $\cos \pi x$ are computed by the power series corresponding to the Chebyshev expansions developed by J. L. Schonfelder (University of Birmingham, England). The power series were obtained by A. H. Morris.

Precision. DCOS1(x) is accurate to within 1 unit of the 28^{th} significant digit when $0 \leq x \leq 1$ and $x \neq 1/2$.

References. Schonfelder, J.L.,"Very High Accuracy Chebyshev Expansions for the Basic Trigonometric Functions," ***Math Comp***. **34** (1980), pp. 237–244.

Programming. DCOS1 was written by A. H. Morris. The function IPMPAR is used.

HYPERBOLIC SINE AND COSINE FUNCTIONS

The following subroutine is available for computing $\sinh(x) - x$, $\cosh(x) - 1$, and $\cosh(x) - 1 - x^2/2$ for real x.

CALL SNHCSH(S, C, x, IND)

S and C are variables, and IND is an input argument which specifies the functions to be computed. IND takes the values:

IND $= -1$ if only $\sinh(x) - x$ is desired.
IND $= 0$ if $\sinh(x) - x$ and $\cosh(x) - 1$ are desired.
IND $= 1$ if only $\cosh(x) - 1$ is desired.
IND $= 2$ if only $\cosh(x) - 1 - x^2/2$ is desired.
IND $= 3$ if $\sinh(x) - x$ and $\cosh(x) - 1 - x^2/2$ are desired.

S is assigned the value $\sinh(x) - x$ if this function is requested. When $\cosh(x) - 1$ or $\cosh(x) - 1 - x^2/2$ is computed then the value is stored in C.

Precision. For all x, $\sinh(x) - x$ has a relative error less than 2.3E-14, $\cosh(x) - 1$ has a relative error less than 2.2E-14, and $\cosh(x) - 1 - x^2/2$ has a relative error less than 3.8E-14.

Programming. Written by A. K. Cline and R. J. Renka (University of Texas at Austin). Modified by A. H. Morris.

EXPONENTIALS

The functions REXP and DREXP are available for computing $e^x - 1$. DREXP is a double precision function.

REXP(x)

REXP$(x) = e^x - 1$ for real x.

Algorithm. See pages 378–379 and appendix B of the reference

Precision. REXP(x) is accurate to within 2 units of the 14^{th} significant digit when REXP$(x) \neq 0$.

Reference. DiDonato, A. R., and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** 12 (1986), pp. 377–393.

Programmer. A. H. Morris.

DREXP(x)

The argument x is a double precision number. DREXP(x) is the double precision value for $e^x - 1$.

Remark. DREXP must be declared in the calling program to be of type DOUBLE PRECISION.

Precision. DREXP(x) is accurate to within 1 unit of the 28^{th} significant digit when DREXP$(x) \neq 0$.

Programmer. A. H. Morris.

LOGARITHMS

The functions ALNREL, RLOG, RLOG1, DLNREL, DRLOG, and DRLOG1 are available for computing $\ln(1+a)$, $x-1-\ln(x)$, and $a-\ln(1+a)$. DLNREL, DRLOG, and DRLOG1 are double precision functions.

ALNREL(a)

ALNREL(a) $= \ln(1+a)$ for $a > -1$.

Algorithm. See page 378 and appendix A of the reference.

Precision. ALNREL(a) is accurate to within 2 units of the 14^{th} significant digit when ALNREL(a) $\neq 0$.

Reference. DiDonato, A. R., and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** 12 (1986), pp. 377–393.

Programmer. A. H. Morris.

RLOG(x)

RLOG(x) $= x-1-\ln(x)$ for $x > 0$.

Algorithm. See page 379 and appendix E of the reference.

Precision. RLOG(x) is accurate to within 2 units of the 14^{th} significant digit when RLOG(x) $\neq 0$.

Reference. DiDonato, A. R., and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** 12 (1986), pp. 377–393.

Programmer. A. H. Morris.

RLOG1(a)

RLOG1(a) $= a-\ln(1+a)$ for $a > -1$.

Algorithm. See page 379 and appendix E of the reference.

Precision. RLOG1(a) is accurate to within 2 units of the 14^{th} significant digit when RLOG1(a) $\neq 0$.

Reference. DiDonato, A. R., and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** 12 (1986), pp. 377–393.

Programmer. A. H. Morris.

DLNREL(a)

The argument a is a double precision number where $a > -1$. DLNREL(a) is the double precision value for $\ln(1 + a)$.

Remark. DLNREL must be declared in the calling program to be of type DOUBLE PRECISION.

Precision. DLNREL(a) is accurate to within 1 unit of the 28^{th} significant digit when DLNREL(a) $\neq 0$.

Programmer. A. H. Morris.

DRLOG(x)

The argument x is a positive double precision number. DRLOG(x) is the double precision value for $x - 1 - \ln(x)$.

Remark. DRLOG must be declared in the calling program to be of type DOUBLE PRECISION.

Precision. DRLOG(x) is accurate to within 1 unit of the 28^{th} significant digit when DRLOG(x) $\neq 0$.

Programmer. A. H. Morris.

DRLOG1(a)

The argument a is a double precision number $a > -1$. DRLOG1(a) is the double precision value for $a - \ln(1 + a)$.

Remark. DRLOG1 must be declared in the calling program to be of type DOUBLE PRECISION.

Precision. DRLOG1(a) is accurate to within 1 unit of the 28^{th} significant digit when DRLOG1(a) $\neq 0$.

Programmer. A. H. Morris.

DETERMINING IF A POINT IS INSIDE OR OUTSIDE A POLYGON

Given a sequence of points $\nu_i = (x_i, y_i)$ $(i = 1, \ldots, n)$. Let τ denote the polygonal line which begins at point ν_1, traverses the points ν_i in the order that they are indexed, and is the straight line segment connecting ν_i to ν_{i+1} for each $i = 1, \ldots, n-1$. It is assumed that $\nu_n = \nu_1$, or that there is also a straight line segment from ν_n to ν_1 when $\nu_n \neq \nu_1$. Consequently, the polygonal path τ is a loop beginning and ending at ν_1.

For any point $\nu_0 = (x_0, y_0)$ not on the path τ, let $\eta(\tau, \nu_0)$ denote the ***winding number*** of the path τ around the point ν_0. Then $\eta(\tau, \nu_0) = 0$ if ν_0 is outside the polygon whose boundary is τ. If ν_0 is inside the path then $\eta(\tau, \nu_0) = m$ where m is an integer. If $m > 0$ then the path τ loops m times in a counterclockwise direction around the point ν_0. Otherwise, if $m < 0$ then τ loops $|m|$ times in a clockwise direction around ν_0.

Given an arbitrary point $\nu_0 = (x_0, y_0)$, the following subroutine is available for determining whether ν_0 is on the path, outside the path, or inside the path. If the point ν_0 is inside τ then the winding number $\eta(\tau, \nu_0) = m$ is also computed.

CALL LOCPT$(x_0, y_0, X, Y, n, \ell, m)$

It is assumed that $n \geq 1$. X and Y are arrays containing $x_1 \ldots, x_n$ and $y_1, \ldots, y_n$. The arguments ℓ and m are variables. When LOCPT terminates ℓ has one of the following values:

$\ell = -1$ if (x_0, y_0) is outside the path
$\ell = \ 0$ if (x_0, y_0) lies on the path
$\ell = \ 1$ if (x_0, y_0) is inside the path

The variable m is assigned the value 0 if (x_0, y_0) is on or outside the path. If (x_0, y_0) is inside the path then $m =$ the winding number of the path τ around the point (x_0, y_0).

Remark. There are no restrictions on the points (x_i, y_i). Consequently, the path may intersect itself.

Programming. The function SPMPAR is used. LOCPT was written by A. H. Morris.

INTERSECTION OF A STRAIGHT LINE AND POLYGONAL PATH

Given a sequence of points $\nu_i = (x_i, y_i)$ $(i = 1, \ldots, n)$. Let τ denote the polygonal path which begins at point ν_1, traverses the points ν_i in the order that they are indexed, and is the straight line segment from ν_i to ν_{i+1} for each $i = 1, \ldots, n-1$. Also consider a straight line ℓ connecting two points (a_1, a_2) and (b_1, b_2). If ℓ and τ intersect at a finite number of points, then the following subroutine is available for obtaining the intersecting points.

CALL PFIND$(a, b, X, Y, n, U, V, m, k,$IERR$)$

The arguments a and b are arrays of dimension 2 containing the points (a_1, a_2) and (b_1, b_2). It is assumed that $a \neq b$ and $n \geq 2$. X and Y are arrays containing $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively.

The argument m is the estimated maximum number of points at which the line ℓ and path τ may intersect $(m < n)$. U and V are arrays of dimension m or larger, and k and IERR are variables. When PFIND is called, if no input errors are detected then IERR is set to 0, k = the number of points (u_j, v_j) where the line ℓ and path τ intersect, and U and V contain the abscissas $u_1, \ldots, u_k$ and ordinates $v_1, \ldots, v_k$ of these points. The points (u_j, v_j) are listed in the order that the path intersects ℓ when proceeding from (x_1, y_1) to (x_n, y_n).

Error Return. If an input error is detected than IERR has one of the following values:

IERR = 1 Either $a = b$ or the path contains a single point.

IERR = 2 U and V require more storage. The argument m must be increased.

IERR = i The i^{th} line segment of the path contains a segment of the line ℓ.

When an error is detected, k = the number of points (u_j, v_j) found where the line ℓ and path τ intersect, and U and V contain the abscissas and ordinates of these points.

Remark. The intersecting points (u_j, v_j) need not be distinct. The path may intersect ℓ at the same point a number of times when proceeding from (x_1, y_1) to (x_n, y_n).

Programming. PFIND was written by A. H. Morris. The function SPMPAR is used.

THE CONVEX HULL FOR A FINITE PLANAR SET

If $(x_1, y_1), \ldots, (x_m, y_m)$ are m distinct points in the plane, then the following subroutine is available for finding the smallest convex polygon which with its interior contains the points.

CALL HULL$(X, Y, m, \text{BX}, \text{BY}, k, \text{VX}, \text{VY}, n)$

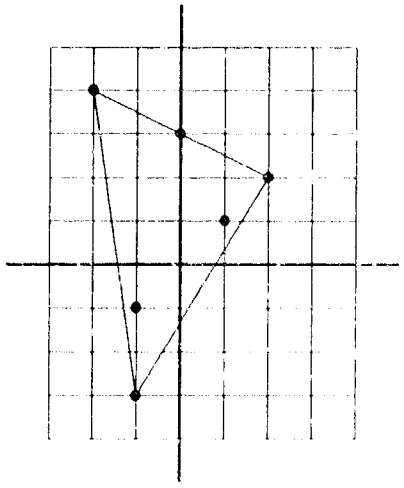
It is assumed that $m \geq 2$. X and Y are arrays containing the abscissas $x_1, \ldots, x_m$ and ordinates $y_1, \ldots, y_m$, respectively. When HULL is called the points are reordered so that $y_1 \leq \cdots \leq y_m$. Thus X and Y may be modified when the routine terminates.

BX and BY are arrays of dimension $m + 1$ or larger, and k is a variable. When HULL terminates, BX and BY contain the abscissas and ordinates of the points (x_i, y_i) which lie on the boundary of the desired convex polygon, and k = the number of points stored in BX and BY. If BX and BY contain the abscissas $x'_1, \ldots, x'_k$ and ordinates $y'_1, \ldots, y'_k$, then the points (x'_i, y'_i) are indexed in the order they occur when traversing the boundary in a counterclockwise manner. Also $(x'_k, y'_k) = (x'_1, y'_1)$.

VX and VY are arrays of dimension $m + 1$ or larger, and n is a variable. When HULL terminates, VX and VY contain the abscissas and ordinates of the vertices of the desired convex polygon, and n = the number of points stored in VX and VY. If VX and VY contain the abscissas $x''_1, \ldots, x''_n$ and ordinates $y''_1, \ldots, y''_n$, then the vertices (x''_i, y''_i) are indexed in the order they occur when traversing the boundary of the convex polygon in a counterclockwise manner. Also $(x''_n, y''_n) = (x''_1, y''_1)$.

Example. Assume that we are given the points $(-1, -3)$, $(1, 1)$ $(0, 3)$, $(2, 2)$, $(-2, 4)$, $(-1, -1)$. When HULL is called X and Y are reordered and we obtain:

X contains $-1, -1, 1, 2, 0, -2$
Y contains $-3, -1, 1, 2, 3, 4$
BX contains $-1, 2, 0, -2, -1$
BY contains $-3, 2, 3, 4, -3$
VX contains $-1, 2, -2, -1$
VY contains $-3, 2, 4, -3$



Programming. HULL calls the subroutine RRSORT and function SPMPAR. HULL was written by A. H. Morris.

AREAS OF PLANAR POLYGONS

Given a sequence of points $v_i = (x_i, y_i)$ $(i = 1, \ldots, n+1)$ where $n \geq 1$ and $v_{n+1} = v_1$. Let τ denote the polygon whose boundary $\partial\tau$ is a polygonal line which begins at point v_1, traverses the points v_i in the order that they are indexed, and is the straight line segment connecting v_i to v_{i+1} for $i = 1, \ldots, n$. Then the function PAREA is available for computing $A(\tau) = \int\int_\tau \mathrm{dxdy}$. If the boundary $\partial\tau$ is a positively (negatively) oriented simple closed curve, then $A(\tau)$ is positive (negative) and $|A(\tau)|$ = the area of τ. However, $\partial\tau$ need not be simple. It may be self-intersecting or have overlapping line segments.

PAREA(X, Y, N)

X and Y are arrays containing the abscissas $x_1, \ldots, x_N$ and ordinates $y_1, \ldots, y_N$, respectively. The argument N may have the value n or $n+1$. Since $v_{n+1} = v_1, x_{n+1}$ and y_{n+1} are not required to appear in X and Y. PAREA(X, Y, N) is assigned the value $A(\tau)$.

Programmer. A. H. Morris

Reference. DiDonato, A. R. and Hageman, R. K., ***Computation of the Integral of the Bivariate Normal Distribution over Arbitrary Polygons***, Report TR 80-166, Naval Surface Weapons Center, Dahlgren, Virginia, 1980.

HAMILTONIAN CIRCUITS

Given a directed graph G containing n vertices, denoted by the integers $1, \ldots, n$. Then any circuit of n arcs which traverses the n vertices, say in the order $i_1, \ldots, i_n$, is called a ***Hamiltonian circuit***. For convenience, it is assumed that for any two vertices i and j, no more than one arc exists which begins at i and ends at j. Then the following subroutine is available for finding the Hamiltonian circuits of G, if any exist.

CALL HC(IND, m, n, P, A, NB, S, IWK,NUM)

The argument m is the number of arcs in the graph G. P is an integer array of dimension $n+1$ and A an integer array of dimension m. The graph is stored in P and A as follows: For $i = 1, \ldots, n$ let

$$R_i = \{j\text{: there exists an arc which begins at } i \text{ and ends at } j\}.$$

Then the vertices in $R_1, \ldots, R_n$ are stored in A, where the data in R_i precedes the data in R_{i+1} for $i = 1, \ldots, n-1$. For each i, the vertices in R_i may be given in any order in A. The array P contains the data

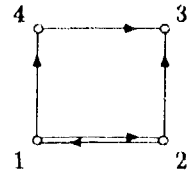
$$P(1) = 0$$
$$P(i+1) = \text{ the total number of vertices in } R_1, \ldots, R_i \ (i = 1, \ldots, n).$$

Hence, if $P(i) < P(i+1)$ then the vertices in R_i are found in locations $P(i)+1, \ldots, P(i+1)$ of A. Otherwise, if $P(i) = P(i+1)$ then $R_i = \phi$ (the empty set); i.e., there are no arcs in G which begin at i (and no Hamiltonian circuits exist). Also, $P(n+1) = m$ since there are m arcs in G.

Example. Consider the graph where

$m = 5, n = 4$
$R_1 = \{2,4\}$ $R_3 = \phi$
$R_2 = \{1,3\}$ $R_4 = \{3\}$.

Then A contains 4,2,1,3,3
P contains 0,2,4,4,5.



When HC is called, a depth-first tree search employing backtracking is used. NB is a variable for controlling the backtracking. If NB ≥ 0 on input, then it is assumed that NB is the maximum number of backtracks that may be performed to find a Hamiltonian circuit. Otherwise, if NB < 0 then it is assumed that no restriction is placed on the backtracking. When HC terminates, NB = the number of backtracks that were actually performed.

S is an integer array of dimension n. When HC is called, if a Hamiltonian circuit is found which traverses the vertices, say in the order $i_1, \ldots, i_n$, then the ordered vertices $i_1, \ldots, i_n$ are stored in S.

IWK is an array of dimension NUM that is a work space for the routine. It is assumed that NUM $\geq m + 8n + 20$.

Only one Hamiltonian circuit will be obtained on a call to HC. However, this routine can be repeatedly called to obtain all the Hamiltonian circuits. IND is a variable which controls the operation of the routine on input, and reports the status of the results on output. It is assumed that IND = 0 on the first call to HC. When the routine terminates, if no input errors are detected then IND has one of the following values:

IND = 1 A Hamiltonian circuit was found and the ordered vertices traversed by the circuit stored in S. To find another circuit, reset NB and recall the routine.

IND = 2 The maximum number of backtracks were performed. To continue, reset NB and recall the routine.

IND = 4 No more circuits exist. The array A has been restored (see the remark on the storage of A below) and the procedure is finished.

We note in passing that on an initial call to the routine, the setting IND = 4 on output indicates that the graph contains no Hamiltonian circuits.

After a call to HC, if IND = 1 or 2 on output then the search procedure can be continued by resetting NB and recalling the routine. ***Do not modify*** IND ***when the tree search is to be continued.*** In this case, the storage of A has been temporarily modified and IWK contains information needed for the search. If a new Hamiltonian circuit is found when HC is recalled, then the vertices traversed by the new circuit will now be stored in S.

After a call to HC, if IND = 1 or 2 on output and it is desired that the search procedure be terminated, then reset IND = 3 and recall the routine. In this case, the array A will be restored and IND = 4 when HC terminates.

Storage of A. When A is restored, the order of the vertices of R_i in A may be modified for each i.

Error Return. If an input error is detected then IND is set to one of the following values:

IND = −1 IND < 0 or IND > 3 on input.

IND = −2 IND was modified, being assigned a value ≠ 3 when HC was recalled. Reset IND to its previous output value, reset NB, and recall HC if another circuit is wanted.

IND = −3 The input setting IND = 3 is not needed when the previous output value for IND was 4. In this case, nothing was done.

IND = −4 NUM < $m + 8n + 20$.

IND = −5 $P(1) \neq 0$ or $P(n+1) \neq m$.

IND = −6 $P(i) > P(i+1)$ for some i.

Remarks.

(1) It is assumed that G contains no loops.

(2) Normally, few backtracks are needed when the number of vertices in each R_i is small, say 10 or less. Consequently, the setting NB = −1 is generally appropriate in such cases.

Programming. HC employs the subroutines HC1, IPATH, FUPD, BUPD, IUPD, and RARC. The search procedure in these routines was written by Silvano Martello (University of Bologna, Italy). The user interface involving the variable IND was written by A. H. Morris.

Reference. Martello, S., "Algorithm 595. An Enumerative Algorithm for Finding Hamiltonian Circuits in a Directed Graph," ***ACM Trans. Math Software*** 9 (1983), pp. 131–138.

ERROR FUNCTION

For any complex z the error function is defined by

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$$

and its complement by $\operatorname{erfc}(z) = 1 - \operatorname{erf}(z)$. The subroutines CERF, CERFC, DCERF, and DCERFC are available for computing $\operatorname{erf}(z)$ and $\operatorname{erfc}(z)$ when z is complex, and the functions ERF, ERFC, ERFC1, DERF, DERFC, and DERFC1 are available for computing $\operatorname{erf}(z)$ and $\operatorname{erfc}(z)$ when z is real. DCERF, DCERFC, DERF, DERFC, and DERFC1 are double precision routines.

CALL CERF(MO, z, w)

MO is an integer, z a complex number, and w a complex variable. When CERF is called, w is assigned the value $\operatorname{erf}(z)$ if MO $= 0$ and the value $\operatorname{erfc}(z)$ if MO $\neq 0$.

Algorithm. For $z = x + iy$ where $x \geq 0$, if z satisfies $|z| \leq 1$ or both of the inequalities $1 < |z| < \sqrt{38}$ and $x^2 - y^2 + .064x^2y^2 \leq 0$, then the series

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{n \geq 0} \frac{(-1)^n z^{2n+1}}{n!(2n+1)} \tag{1}$$

is used. If $1 < |z| < \sqrt{38}$ and $x^2 - y^2 + .064x^2y^2 > 0$ then

$$\operatorname{erf}(z) = 1 - \frac{ze^{-z^2}}{\sqrt{\pi}} \sum_{n=1}^{18} \frac{r_n}{z^2 + \lambda_n} \tag{2}$$

is employed. λ_n and r_n are the poles and residues of the rational function approximation for the complex Fresnel integral $E(z)$ given in the reference. The error function is related to $E(z)$ by $\operatorname{erf}(z) = 1 - i\sqrt{2}E(-z^2)$ for $|\arg(z)| < \pi/2$. If $|z| \geq \sqrt{38}$ and $x \geq .01$ then $\operatorname{erf}(z)$ is computed by the asymptotic expansion $\operatorname{erf}(z) = 1 - \psi(z)$ where

$$\psi(z) = \frac{e^{-z^2}}{\sqrt{\pi}} \left[\frac{1}{z} + \sum_{n \geq 1} (-1)^n \frac{1 \cdot 3 \cdots (2n-1)}{2^n z^{2n+1}} \right]. \tag{3}$$

Otherwise, if $|z| \geq \sqrt{38}$ and $0 \leq x < .01$ then $\operatorname{erf}(z) = -\psi(z)$ is employed. When $x < 0$ then the relation $\operatorname{erf}(-z) = -\operatorname{erf}(z)$ is applied.

Programming. Written by Allen V. Hershey and A. H. Morris.

Reference. Hershey, A. V., ***Approximation of Functions by Sets of Poles***, Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

CALL CERFC(MO, z, w)

MO is an integer, z a complex number, and w a complex variable. When CERFC is called, w is assigned the value erfc(z) if MO = 0 or Re(z) < 0. Otherwise, if MO $\neq$ 0 and Re(z) $\geq$ 0 then w is assigned the value e^{z^2}erfc(z).

Precision. For MO $\neq$ 0, Re(w) and Im(w) are accurate to within 1 unit of the 12^{th} significant digit when $|\text{Re}(w)| \geq 10^{-280}$ and Im(w) $\neq$ 0.

Programming. CERFC employs the subroutine CREC and functions EXPARG and IPMPAR. CERFC was written by Allen V. Hershey and A. H. Morris.

Reference. Hershey, A. V, ***Approximation of Functions by Sets of Poles***, Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

ERF(x)

ERF(x) = erf(x) for any real x.

Precision. ERF(x) is accurate to within 2 units of the 14^{th} significant digit for $x \neq 0$.

Programmer. A. H. Morris

Reference. Cody, W. J., "Rational Chebyshev Approximations for the Error Function," ***Math Comp***. 23 (1969), pp. 631–637.

ERFC(x)

ERFC(x) = erfc(x) for any real x.

Precision. If $x \leq 3$ then ERFC(x) is accurate to within 2 units of the 14^{th} significant digit. Otherwise, if $x > 3$ then ERFC(x) is accurate to within 4 units of the 14^{th} significant digit when ERFC(x) $\neq$ 0.

Programming. ERFC employs the functions EXPARG and IPMPAR. ERFC was written by A. H. Morris.

Reference. Cody, W. J., "Rational Chebyshev Approximations for the Error Function," ***Math Comp***. 23 (1969), pp. 631–637.

ERFC1(IND, x)

IND is an integer and x a real number. ERFC1(IND,x) = erfc(x) when IND = 0, and ERFC1(IND,x) = e^{x^2}erfc(x) when IND $\neq$ 0.

Precision. If $x \leq 3$ then ERFC1(0, x) is accurate to within 2 units of the 14^{th} significant digit. Otherwise, if $x > 3$ then ERFC1(0, x) is accurate to within 4 units of the 14^{th} significant digit when ERFC1(0, x) $\neq$ 0. If IND $\neq$ 0 then ERFC1(IND,x) is accurate to

within 2 units of the 14^{th} significant digit for $x \geq -1$.

Programming. ERFC1 employs the functions EXPARG and IPMPAR. ERFC1 was written by A. H. Morris.

Reference. Cody, W. J., "Rational Chebyshev Approximations for the Error Function," *Math Comp.* 23 (1969), pp. 631–637.

CALL DCERF(MO,Z,W)

MO is an integer, and Z and W are double precision arrays of dimension 2. It is assumed that $Z(1)$ and $Z(2)$ are the real and imaginary parts of a complex number z. If MO $= 0$ then the double precision value for $w = \mathrm{erf}(z)$ is computed, and if MO $\neq 0$ then the double precision value for $w = \mathrm{erfc}(z)$ is computed. $W(1)$ and $W(2)$ contain the real and imaginary parts of w, respectively.

Algorithm. For $z = x + iy$ where $x \geq 0$, if $|z| \leq 1$ then (1) is used, and if $|z - 2| < 1$ and $x \leq 2$ then the Taylor series

$$\mathrm{erfc}(z) = \mathrm{erfc}(a) + \frac{2}{\sqrt{\pi}} e^{-a^2} \sum_{n=1}^{\infty} (-1)^n H_{n-1}(a)(z-a)^n/n! \qquad (a = 2) \tag{4}$$

is employed. Here $H_n(a)$ are the Hermite polynomials. If $1 < |z| < 2.5$ then (4) and the Pade approximation $A_n(z^2)/B_n(z^2)$ for $\sqrt{\pi}(2z)^{-1}e^{z^2}\mathrm{erf}(z)$ are used where

$$\begin{aligned} A_0(z) &= 1 \qquad A_1(z) = 1 + \tfrac{4}{15} z \\ B_0(z) &= 1 \qquad B_1(z) = 1 - \tfrac{2}{5} z \end{aligned} \tag{5}$$

and A_n and B_n satisfy

$$B_{n+1}(z) = \left[1 - \frac{2z}{(4n+1)(4n+5)}\right] B_n(z) + \frac{4n(4n+2)z^2}{(4n-1)(4n+1)^2(4n+3)} B_{n-1}(z) \tag{6}$$

(see pp. 191,192, and 422 of the reference). Also, if $2.5 \leq |z| < 12$ then $A_n(z^2)/B_n(z^2)$ and the Pade approximation $\overline{G}_n(z^2)/\overline{H}_n(z^2)$ for $\sqrt{\pi}ze^{z^2}\mathrm{erfc}(z)$ are employed where

$$\begin{aligned} \overline{G}_0(z) &= 1 \qquad \overline{G}_1(z) = 2 + 2z \\ \overline{H}_0(z) &= 1 \qquad \overline{H}_1(z) = 3 + 2z \end{aligned} \tag{7}$$

and $\overline{G}_n$ and $\overline{H}_n$ satisfy

$$\overline{H}_{n+1}(z) = (2z + 4n + 3)\overline{H}_n(z) - 2n(2n+1)\overline{H}_{n-1}(z) \tag{8}$$

(see pp. 201 and 422 of the reference). Otherwise, if $|z| \geq 12$ and $x \geq .01$ then the asymptotic expansion $\mathrm{erfc}(z) = \psi(z)$ is used where $\psi(z)$ is given by (3). Also, if $|z| \geq 12$ and $0 \leq x < .01$ then $\mathrm{erf}(z) = -\psi(z)$ is employed. When $x < 0$ the relation $\mathrm{erf}(-z) = -\mathrm{erf}(z)$ is applied.

Programming. DCERF calls the subroutines ERFCM2, CDIVID, and DCREC. DCERF and ERFCM2 were written by A. H. Morris. The function DPMPAR is also used.

Reference. Luke, Yudell L., ***The Special Functions and Their Approximations, Vol 2***, Academic Press, New York, 1969.

CALL DCERFC(MO,Z,W)

MO is an integer, and Z and W are double precision arrays of dimension 2. It is assumed that $Z(1)$ and $Z(2)$ are the real and imaginary parts of a complex number z. If MO $= 0$ or Re$(z) < 0$ then the double precision value for $w =$ erfc(z) is computed. Otherwise, if MO $\neq 0$ and Re$(z) \geq 0$ then the double precision value for $w = e^{z^2}$erfc(z) is computed. $W(1)$ and $W(2)$ contain the real and imaginary parts of w, respectively.

Precision. For MO $\neq 0$ and Re$(z) \geq 0$, $W(1)$ and $W(2)$ are accurate to within 3 units of the 26^{th} significant digit when $W(2) \neq 0$.

Programming. DCERFC employs the subroutines ERFCM2, DCIVID, and DCREC, and functions DXPARG, DPMPAR,and IPMPAR. DCERFC and ERFCM2 were written by A. H. Morris.

DERF(x)

The argument x is a double precision number. DERF(x) is the double precision value for erf(x).

Remark.DERF must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $|x| \leq 1$ then the power series corresponding to the Chebyshev expansion given in the SLATEC library by Wayne Fullerton (Los Alamos) is used, and if $|x| > 1$ then minimax approximations are employed. The power series and minimax approximations were obtained by A. H. Morris.

Precision. DERF(x) is accurate to within 2 units of the 28^{th} significant digit for $x \neq 0$.

Programming. DERF employs the function DERFC0. DERF was written by A. H. Morris.

DERFC(x)

The argument x is a double precision number. DERFC(x) is the double precision value for erfc(x).

Remark. DERFC must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $|x| \leq 1$ then the power series corresponding to the Chebyshev expansion given in the SLATEC library by Wayne Fullerton (Los Alamos) is used. Otherwise, if $1 < |x| < 50$

then minimax approximations are used, and if $|x| \geq 50$ then the asymptotic expansion (3) is employed. The power series and minimax approximations were obtained by A. H. Morris.

Precision. DERFC(x) is accurate to within 2 units of the 28^{th} significant digit for $x \leq 2$.

Programming. DERFC employs the functions DERFC0, DXPARG, and IPMPAR. DERFC was written by A. H. Morris.

DERFC1(IND, x)

IND is an integer and x a double precision number. DERFC1(IND,x) is the double precision value for erfc(x) when IND $= 0$, and DERFC1(IND,x) is the double precision value for e^{x^2}erfc(x) when IND $\neq 0$.

Remark. DERFC1 must be declared in the calling program to be of type DOUBLE PRECISION.

Precision. DERFC1(IND,x) is accurate to within 2 units of the 28^{th} significant digit when IND $= 0$ and $x \leq 2$, and when IND $\neq 0$ and $x \geq -1$

Programming. DERFC1 employs the functions DERFC0, DXPARG, and IPMPAR. DERFC1 was written by A. H. Morris.

INVERSE ERROR FUNCTION

For any $0 \leq x < 1$, the following functions are available for obtaining the value $w \geq 0$ for which $erf(w) = x$.

ERFI(x, y)

It is assumed that $x \geq 0$, $y > 0$, and $x + y = 1$. Then ERFI$(x, y) = w$ where $w \geq 0$ and $erf(w) = x$.

Error Return. ERFI$(x, y) = -1$ if $x < 0$ or $y \leq 0$, and ERFI$(x, y) = -2$ if $x + y \neq 1$.

Precision. ERFI(x, y) is accurate to within 3 units of the 14^{th} significant digit.

Programming. ERFI was written by Armido R. DiDonato and modified by A. H. Morris. The function SPMPAR is used.

Reference. Blair, J. M., Edwards, C. A., and Johnson, J. H.,"Rational Chebyshev Approximations for the Inverse of the Error Function," ***Math. Comp.*** **30** (1976), pp. 827–830.

DERFI(x, y)

The arguments x and y are double precision numbers where $x \geq 0$, $y > 0$, and $x+y = 1$. DERFI(x, y) is the double precision value for w where $w \geq 0$ and $erf(w) = x$.

Remark. DERFI must be declared to be of type DOUBLE PRECISION in the calling program.

Error Return. DERFI$(x, y) = -1$ if $x < 0$ or $y \leq 0$, and DERFI$(x, y) = -2$ if $x + y \neq 1$.

Algorithm. If $x \leq 15/16$ then an initial value for w is given by minimax approximations due to A. H. Morris. Otherwise, if $x > 15/16$ then an initial value for w is given by minimax approximations from the reference. These approximations are accurate to 17–18 digits. If more accuracy is needed, then a single iterate of the Newton procedure for solving $F(w) = 0$ is taken to obtain w to machine precision. Here

$$F(w) = \begin{cases} erf(w) - x & \text{if } x \leq 3/4 \\ erfc(w) - y & \text{if } 3/4 < x \leq 15/16 \\ \ln[erfc(w)] - \ln(y) & \text{if } x > 15/16. \end{cases}$$

Precision. DERFI(x, y) is accurate to within 1 unit of the 27^{th} significant digit.

Programming. DERFI employs the functions DERF, DERFC1, DERFC0, DXPARG, DPMPAR, and IPMPAR. DERFI was written by A. H. Morris.

Reference. Blair, J. M., Edwards, C. A., and Johnson, J. H.,"Rational Chebyshev Approximations for the Inverse of the Error Function," ***Math. Comp.*** **30** (1976), pp. 827–830.

DIFFERENCE OF ERROR FUNCTIONS

For any real x and h, let $aerf(x,h) = erf(x+h) - erf(x-h)$. Then the functions AERF and DAERF are available for computing $aerf(x,h)$.

AERF(x,h)

AERF$(x,h) = aerf(x,h)$ for any x and h.

Precision. AERF(x,h) is accurate to within 6 units of the 12^{th} significant digit when AERF$(x,h) \neq 0$.

Programming. AERF employs the functions ERF, ERFC, EPSLN, EXPARG, SPMPAR, and IPMPAR. AERF was written by Armido R. DiDonato and modified by A.H. Morris.

Reference. DiDonato, A.R., ***Significant Digit Computation of the Elliptical Coverage Function***, Report NSWC TR 90–513, Naval Surface Warfare Center, Dahlgren, Virginia, 1990.

DAERF(x,h)

The arguments x and h are double precision numbers. DAERF(x,h) is the double precision value for $aerf(x,h)$.

Remark. DAERF must be declared to be of type DOUBLE PRECISION in the calling program.

Programming. DAERF employs the functions DERF, DERFC, DERFC0, DEPSLN, DXPARG, DPMPAR, and IPMPAR. DAERF was written by Armido R. DiDonato and modified by A. H. Morris.

Reference. DiDonato, A.R., ***Significant Digit Computation of the Elliptical Coverage Function***, Report NSWC TR 90–513, Naval Surface Warfare Center, Dahlgren, Virginia, 1990.

NORMAL PROBABILITY DISTRIBUTION FUNCTION

For any real x, the normal probability distribution function $P(x)$ (of mean 0 and variance 1) is defined by

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$$

and its complement by $Q(x) = 1 - P(x)$. The following function is available for computing $P(x)$ and $Q(x)$.

PNDF(x,IND)

IND is an integer and x a real number. If IND = 0 then

$$\text{PNDF}(x,0) = \begin{cases} P(x) & \text{if } x \geq -8 \\ \dfrac{P'(x)}{P(x)} & \text{if } x < -8 \end{cases}$$

where $P'(x)$ is the derivative of $P(x)$. Otherwise, if IND $\neq 0$ then

$$\text{PNDF}(x,\text{IND}) = \begin{cases} Q(x) & \text{if } x \leq 8 \\ \dfrac{Q'(x)}{Q(x)} & \text{if } x > 8 \end{cases}$$

where $Q'(x)$ is the derivative of $Q(x)$.

Algorithm. The identities $P(x) = \frac{1}{2}\,\text{erfc}(-x/\sqrt{2})$ and $Q(x) = \frac{1}{2}\,\text{erfc}(x/\sqrt{2})$ are used.

Programming. PNDF calls the function ERFC1. PNDF was written by A.H. Morris.

INVERSE NORMAL PROBABILITY DISTRIBUTION FUNCTION

For any real w, the normal probability distribution function $P(w)$ (of mean 0 and variance 1) is defined by

$$P(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{w} e^{-t^2/2}\, dt$$

and its complement by $Q(w) = 1 - P(w)$. For any $0 < p < 1$ and $q = 1 - p$, the following subroutines are available for obtaining the value w for which $P(w) = p$ and $Q(w) = q$.

CALL PNI$(p, q, d, w, \text{IERR})$
CALL DPNI$(p, q, d, w, \text{IERR})$

PNI is used when p, q, and d are real numbers and w is a real variable, and DPNI is used when p, q, and d are double precision numbers and w is a double precision variable.

It is assumed that $p > 0$, $q > 0$, $p + q = 1$, and $d = p - 1/2$. IERR and w are variables. When PNI or DPNI is called, if no input errors are detected then IERR is set to 0. Also, w is assigned the value for which $P(w) = p$ and $Q(w) = q$.

Error Return. IERR = 1 if $p < 0, q < 0$, or $p + q \neq 1$, and IERR = 2 if $d \neq p - 1/2$.

Algorithm. For $y \geq 0$, let $y = \text{erf}^{-1}(x)$ when $x = \text{erf}(y)$. If $P(w) = p$ then the identities

$$w = \begin{cases} -\sqrt{2}\ \text{erf}^{-1}(1 - 2p) & \text{if } 0 < p \leq 1/2 \\ \sqrt{2}\ \text{erf}^{-1}(2p - 1) & \text{if } 1 > p \geq 1/2 \end{cases}$$

are applied.

Precision. If PNI is called then w is accurate to within 3 units of the 14^{th} significant digit. Otherwise, if DPNI is called then w is accurate to within 1 unit of the 27^{th} significant digit.

Programming. PNI employs the functions ERFI, SPMPAR, and IPMPAR, and DPNI employs the functions DERFI, DERF, DERFC1, DERFC0, DXPARG, DPMPAR, and IPMPAR. PNI was written by Armido R. DiDonato and modified by A. H. Morris. DPNI was adapted from PNI by A. H. Morris.

DAWSON'S INTEGRAL

For any real x, Dawson's integral is defined by

$$F(x) = e^{-x^2} \int_0^x e^{t^2}\, dt.$$

The following functions are available for computing $F(x)$.

DAW(x)

DAW(x) = $F(x)$ for any real x.

Precision. DAW(x) is accurate to within 2 units of the 14^{th} significant digit for $x \neq 0$.

Programming. DAW belongs to the FUNPACK package of subroutines developed at Argonne National Laboratory. The function was modified by A. H. Morris.

Reference. Cody, W.J., Paciorek, K.A., and Tacher, H.C., "Chebyshev Approximations for Dawson's Integral," ***Math Comp.*** **24** (1970), pp. 171–178.

DPDAW(x)

The argument x is a double precision number. DPDAW(x) is the double precision value for $F(x)$.

Remark. DPDAW must be declared to be of type DOUBLE PRECISION in the calling program.

Algorithm. If $|x| \leq 1$ then the power series corresponding to the 19^{th} degree Chebyshev expansion given in the SLATEC library by Wayne Fullerton (Los Alamos) is used, and if $1 < |x| < 4$ then the 44^{th} degree Chebyshev expansion given in the SLATEC library by Wayne Fullerton is employed. Otherwise, if $|x| \geq 4$ then minimax approximations are applied. The power series and minimax approximations were obtained by A. H. Morris.

Programming. DPDAW employs the functions DCSEVL and DPDAW0. DPDAW was written by A. H. Morris.

COMPLEX FRESNEL INTEGRAL

For any complex z not on the positive real axis the complex Fresnel integral $E(z)$ can be defined by

$$E(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \frac{e^t}{\sqrt{t}}\, dt.$$

Here it is assumed that $0 < \arg(z) < 2\pi$ and $\arg(\sqrt{z}) = 1/2 \arg(z)$. $E(z)$ can be extended to the positive real axis by letting $0 \leq \arg(z) < 2\pi$. Then $\mathrm{erf}(z) = 1 - i\sqrt{2}\, E(-z^2)$ for $-\pi/2 \leq \arg(z) < \pi/2$ where $\mathrm{erf}(z)$ is the error function and $\arg(-z^2) = \pi + 2 \arg(z)$. The following subroutine is available for computing $E(z)$.

CALL CFRNLI(MO, z, w)

MO is an integer, z a complex number, and w a complex variable. When CFRNLI is called, w is assigned the value $E(z)$ if MO $= 0$ and the value $e^{-z}E(z)$ if MO $\neq 0$.

Algorithm. If $z = x + iy$ satisfies $|z| \leq 1$ or both of the inequalities $1 < |z| < 38$ and $-x + .016y^2 \leq 0$, then the series

$$E(z) = -\frac{i}{\sqrt{2}} + \sqrt{2z/\pi} \sum_{n \geq 0} \frac{z^n}{n!(2n+1)}$$

is used. If $1 < |z| < 38$ and $-x + .016y^2 > 0$ then

$$E(z) = \sqrt{\frac{z}{2\pi}}\, e^z \sum_{n=1}^{18} \frac{r_n}{z - \delta_n}$$

is employed. If $|z| \geq 38$ and $\mathrm{Im}\ \sqrt{2z/\pi} \geq .008$ then $E(z)$ is computed by the asymptotic expansion $E(z) = \psi(z)$ where

$$\psi(z) = \frac{e^z}{\sqrt{2\pi z}} \left[1 + \sum_{n \geq 1} \frac{1 \cdot 3 \cdots (2n-1)}{(2z)^n} \right].$$

Otherwise, if $|z| \geq 38$ and $\mathrm{Im}\ \sqrt{2z/\pi} < .008$ then $E(z) = -i/\sqrt{2} + \psi(z)$ is employed.

Precision. For MO $\neq 0$, $\mathrm{Re}(w)$ and $\mathrm{Im}(w)$ are accurate to within 1 unit of the 12^{th} significant digit when $\mathrm{Re}(w)$ and $\mathrm{Im}(w)$ are nonzero.

Programming. CFRNLI employs the functions CPABS, EXPARG, and IPMPAR. CFRNLI was written by Allen V. Hershey and A. H. Morris.

Reference. Hershey, A. V., ***Approximation of Functions by Sets of Poles***, Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

REAL FRESNEL INTEGRALS

For any complex z the Fresnel integrals $C(z)$ and $S(z)$ can be defined by

$$C(z) = \int_0^z \cos\left(\frac{\pi}{2}t^2\right) dt$$
$$S(z) = \int_0^z \sin\left(\frac{\pi}{2}t^2\right) dt$$

The following subroutine is available for computing $C(x)$ and $S(x)$ when x is real.

CALL FRNL(x, C, S)

The argument x may be any real number. C and S are variables. When FRNL is called C is assigned the value $C(x)$ and S is assigned the value $S(x)$.

Algorithm. If $0 \le x < 1.65$ then $x^{-1}C(x)$ and $x^{-3}S(x)$ are computed by minimax polynomial approximations. Otherwise, if $x \ge 1.65$ then the relations

$$C(x) = \frac{1}{2} + f(x)\sin\frac{\pi}{2}x^2 - g(x)\cos\frac{\pi}{2}x^2$$
$$S(x) = \frac{1}{2} - f(x)\cos\frac{\pi}{2}x^2 - g(x)\sin\frac{\pi}{2}x^2$$

are invoked. For $1.65 \le x < 6, xf(x)$ and $x^3g(x)$ are computed by rational minimax approximations. Otherwise, for $x \ge 6$ the auxiliary functions $f(x)$ and $g(x)$ are computed by the asymptotic expansions:

$$f(x) = \frac{1}{\pi x}\left[1 + \sum_{i=1}^{m}(-1)^i \frac{1\cdot 3\cdots(4i-1)}{(\pi x^2)^{2i}}\right]$$
$$g(x) = \frac{1}{\pi x}\sum_{i=0}^{m}(-1)^i \frac{1\cdot 3\cdots(4i+1)}{(\pi x^2)^{2i+1}}$$

Here $m = 5$. If $x < 0$ then the relations $C(-x) = -C(x)$ and $S(-x) = -S(x)$ are applied.

Precision. If $|x| < 2$ then FRNL is accurate to within 3 units of the 14^{th} significant digit. Otherwise, if $|x| \ge 2$ then FRNL is accurate to within 1 unit of the 14^{th} significant digit.

Programming. FRNL calls the functions SIN1, COS1, and IPMPAR. FRNL was written by A. H. Morris.

EXPONENTIAL INTEGRAL FUNCTION

For any complex $z \neq 0$ not on the positive real axis the exponential integral function $\mathrm{Ei}(z)$ is defined by

$$\mathrm{Ei}(z) = \int_{-\infty}^{z} \frac{e^t}{t}\, dt.$$

$\mathrm{Ei}(z)$ is an analytic function. If z is replaced by $-z$ and t by $-t$ we obtain the related function

$$E_1(z) = -\mathrm{Ei}(-z) = \int_{z}^{\infty} \frac{e^{-t}}{t}\, dt$$

which is defined for all $z \neq 0$ not on the negative real axis. It can be verified that

$$\mathrm{Ei}(z) = \nu + \ln(-z) + \sum_{n=1}^{\infty} \frac{z^n}{n \cdot n!}$$

everywhere in the plane cut along the positive real axis where ν is the Euler constant. Thus, the values of $\mathrm{Ei}(x)$ on the upper and lower edges of the cut are

$$\mathrm{Ei}(x \pm i0) = \mathrm{ei}(x) \mp \pi i$$

where $\mathrm{ei}(x)$ is the real function defined by

$$\mathrm{ei}(x) = \nu + \ln x + \sum_{n=1}^{\infty} \frac{x^n}{n \cdot n!}$$

for $x > 0$. The function $\mathrm{ei}(x)$, also known as the exponential integral function, has a zero at the point $x_0 = .37250\ 74107\ 81367$. $\mathrm{Ei}(z)$ may be computed by the subroutine CEXPLI when z is complex, and $\mathrm{Ei}(z)$ and $\mathrm{ei}(z)$ may be computed by the subroutine EXPLI and functions DEI and DEI1 when z is real. DEI and DEI1 are double precision functions.

CALL CEXPLI(MO, z, w)

MO is an integer, $z \neq 0$ a complex number, and w a complex variable. When CEXPLI is called, w is assigned the value $\mathrm{Ei}(z)$ if MO $= 0$ and the value $e^{-z}\ \mathrm{Ei}(z)$ if MO $\neq 0$.

Remark. If z is a positive real number and MO $= 0$ then $w = \mathrm{ei}(z) + \pi i$.

Precision. If MO $= 0$ then $\mathrm{Re}(w)$ and $\mathrm{Im}(w)$ are accurate to within 2 units of the 12^{th} significant digit when z is not near a zero of $\mathrm{Re}(\mathrm{Ei}(z))$ or $\mathrm{Im}(\mathrm{Ei}(z))$.

Programming. CEXPLI employs the functions CPABS and SPMPAR. CEXPLI was initially written by Allen V. Hershey, and later rewritten by A. H. Morris.

Reference. Hershey, A. V., ***Approximations of Functions by Sets of Poles***, Rep. No. TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

CALL EXPLI(MO, x, w, IERR)

MO may have the values 1,2, or 3. The argument x is a nonzero real number and w a real variable. When EXPLI is called, if MO = 1 then w is assigned the value $\mathrm{Ei}(x)$ for $x < 0$ and the value $\mathrm{ei}(x)$ for $x > 0$. If MO = 2 then it is assumed that $x > 0$. In this case w is assigned the value $E_1(x)$. Otherwise, if MO = 3 then w is assigned the value $e^{-x}\mathrm{Ei}(x)$ for $x < 0$ and the value $e^{-x}\,\mathrm{ei}(x)$ for $x > 0$.

Error return. IERR is a variable that reports the status of the results. If the requested value w is obtained then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1 Underflow occurs. In this case $w = 0$.
IERR = 2 Overflow occurs.
IERR = 3 (Input error) $x = 0$.
IERR = 4 (Input error) MO = 2 and $x < 0$.

The variable w is not defined when IERR ≥ 2.

Algorithm. If MO $\neq 2$ and $4 \leq x \leq 8$, then the Chebyshev expansion in the SLATEC library obtained by Wayne Fullerton (Los Alamos) is used. The remaining approximations employed are from the references.

Precision. If MO $\neq 2$ and $x > 0$, then w is accurate to within 4 units of the 14^{th} significant digit when $w \neq 0$. Otherwise, w is accurate to within 3 units of the 14^{th} significant digit when $w \neq 0$.

Programming. EXPLI employs the functions ALNREL, CSEVL, EXPARG, and IPMPAR. EXPLI was written at Argonne National Laboratory for the FUNPACK package of special function subroutines. EXPLI was modified by A. H. Morris.

References.

(1) Cody, W. J. and Thacher, H. C., "Rational Chebyshev Approximations for the Exponential Integral $E_1(x)$," ***Math Comp***. 22 (1968), pp. 641-649.
(2) _____, "Chebyshev Approximations for the Exponential Integral Ei(x)," ***Math Comp***. 23 (1969), pp. 289-303.

DEI(x)

The argument $x \neq 0$ is a double precision number. DEI(x) is the double precision value for Ei(x) when $x < 0$, and the double precision value for ei(x) when $x > 0$.

Remark. DEI must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $.35 \leq x \leq .4$ then the Taylor series expansion of ei(x) around x_0 is used. This expansion was obtained by A. H. Morris. If $|x| \geq 90$ then the standard asymptotic expansion for Ei and ei is applied. Otherwise, the Chebyshev expansions in the SLATEC library obtained by Wayne Fullerton (Los Alamos) are used.

Programming. DEI employs the functions DE1E and DEI0. These functions were written by A. H. Morris. The functions DCSEVL and DPMPAR are also used.

DEI1(x)

The argument $x \neq 0$ is a double precision number. DEI1(x) is the double precision value for e^{-x} Ei(x) when $x < 0$, and the double precision value for e^{-x} ei(x) when $x > 0$.

Remark. DEI1 must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $.35 \leq x \leq .4$ then the Taylor series expansion of ei(x) around x_0 is used. This expansion was obtained by A. H. Morris. If $|x| \geq 90$ then the standard asymptotic expansion for Ei and ei is applied. Otherwise, the Chebyshev expansions in the SLATEC library obtained by Wayne Fullerton (Los Alamos) are used.

Precision. DEI1(x) is accurate to within 4 units of the 28^{th} significant digit when DEI1(x) $\neq 0$.

Programming. DEI1 employs the functions DE1E and DEI0. These functions were written by A. H. Morris. The functions DCSEVL and DPMPAR are also used.

SINE AND COSINE INTEGRAL FUNCTIONS

For any complex z the sine integral and cosine integral functions $\mathrm{Si}(z)$ and $\mathrm{Cin}(z)$ are defined by

$$\mathrm{Si}(z) = \int_0^z \frac{\sin t}{t}\,\mathrm{dt}$$
$$\mathrm{Cin}(z) = \int_0^z \frac{1-\cos t}{t}\,\mathrm{dt}.$$

These are entire functions. The following functions are available for computing $\mathrm{Si}(z)$ and $\mathrm{Cin}(z)$ when z is real.

SI(x)

$\mathrm{SI}(x) = \mathrm{Si}(x)$ for all real x.

Precision. SI is accurate to within 2 units of the 14^{th} significant digit.

Programming. SI calls the function SPMPAR. SI was written by Donald E. Amos and Sharon L. Daniel (Sandia Laboratories), and modified by A. H. Morris.

CIN(x)

$\mathrm{CIN}(x) = \mathrm{Cin}(x)$ for all real x.

Precision. CIN is accurate to within 2 units of the 14^{th} significant digit.

Programming. CIN calls the function SPMPAR. CIN was written by Donald E. Amos and Sharon L. Daniel (Sandia Laboratories), and modified by A. H. Morris.

EXPONENTIAL EXPONENTIAL INTEGRAL FUNCTION

For any complex $z \neq 0$ not on the positive real axis, the exponential exponential integral function $EEi(z)$ can be defined by

$$EEi(z) = \int_{-\infty}^{z} \frac{e^{-u}}{u} Ei(u)\, du$$

where $Ei(u)$ is the exponential integral function. The following subroutine is available for computing $EEi(z)$, where $EEi(z)$ is extended to the positive real axis by letting $0 < \arg(z) \leq 2\pi$.

CALL CEXEXI(z, w)

The argument z is a nonzero complex number and w is a complex variable. When CEXEXI is called, w is assigned the value $EEi(z)$.

Precision. $Re(w)$ and $Im(w)$ are accurate to within 1 unit of the 12^{th} significant digit when $Re\big(EEi(w)\big)$ and $Im\big(EEi(w)\big)$ are not near 0.

Programming. CEXEXI was written by Allen V. Hershey and modified by A. H. Morris.

Reference. Hershey, A. V., ***Approximation of Functions by Sets of Poles***, Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

DILOGARITHM FUNCTION

For any complex z where $|\arg(1+z)| < \pi$, the dilogarithm function $L(z)$ may be defined by

$$L(z) = \int_0^z \frac{\ln(1+t)}{t}\, dt.$$

$L(z)$ is real-valued for any real $z \geq -1$, and -1 is a branch point. $L(z)$ can be extended to the negative real axis from $-\infty$ to -1 by letting $-\pi < \arg(1+z) \leq \pi$. Then for any real $x < -1$

$$L(x) = -\pi^2/6 + \int_1^{-x} \frac{\ln(t-1)}{t}\, dt + i\pi \ln(-x) \quad (i = \sqrt{-1}).$$

The function CLI is available for computing $L(z)$ when z is complex, and the function ALI is available for computing the real part of $L(z)$ when z is real.

CLI(z)

CLI is a complex-valued function where CLI(z) = $L(z)$ for all complex z. CLI must be declared in the calling program to be of type COMPLEX.

Algorithm. For $|z| \leq 1/2$ the Maclaurin series

$$L(z) = -\sum_{n \geq 1} \frac{(-z)^n}{n^2} \tag{1}$$

is used. If $|z| \geq 3$ then

$$L(z) = \pi^2/6 - L(1/z) + 1/2 \ln^2 z \tag{2}$$

is applied, and if $0 < |z+1| \leq 1/2$ then

$$L(z) = -\pi^2/6 - L(-1-z) + \ln(-z)\, \ln(1+z) \tag{3}$$

is applied. Otherwise,

$$\begin{aligned} L(z) &= -\int_0^w \frac{\lambda}{e^\lambda - 1}\, d\lambda \qquad \text{(Debye Function)} \\ &= -w + w^2/4 - \sum_{n \geq 1} \frac{B_{2n} w^{2n+1}}{(2n+1)!} \quad |w| < 2\pi \end{aligned} \tag{4}$$

is used where $w = -\ln(1+z)$ and B_{2n} are the Bernoulli numbers $B_2 = 1/6$, $B_4 = -1/30, \ldots$. In (3) we note that $\ln(-z)\ln(1+z) \to 0$ when $z \to -1$.

Programming. CLI is a modification by A. H. Morris of the subroutine CLGMCI, written by Allen V. Hershey.

Reference. Hershey, A. V., ***Approximation of Functions by Sets of Poles***, Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

ALI(x)

ALI(x) = Re[$L(x)$] for all real x.

Algorithm. Rational minimax approximations are used when $-1/2 \leq x \leq 1$. If $x > 1$ then (2) is applied, and if $-2 \leq x < -1$ or $-1 < x < -1/2$ then (3) is applied. Otherwise, if $x < -2$ then

$$\text{Re}[L(x)] = -\pi^2/3 - L(1/x) + 1/2 \ln^2(-x)$$

(which follows from (2)) is used except when $-26.63 \leq x \leq -6.97$. Since Re[$L(x_0)$] = 0 for $x_0 = -12.59517\ldots$, the Taylor series of Re[$L(x)$] around x_0 is used when $-14 < x \leq -11.1$. Otherwise, if $-11.1 < x \leq -6.97$ or $-26.63 \leq x \leq -14$ then rational minimax approximations are employed.

Precision. ALI(x) is accurate to within 2 units of the 14^{th} significant digit when $x > 0$. Otherwise, if $x < 0$ then ALI(x) is accurate to within 4 units of the 14^{th} significant digit when ALI(x) $\neq 0$.

Programmer. A. H. Morris

Reference. Morris, Robert, "The Dilogarithm Function of a Real Argument," ***Math. Comp.*** **33** (1979), pp. 778-787.

GAMMA FUNCTION

For any complex $z \neq 0, -1, -2, \ldots$ the gamma function can be defined by

$$\Gamma(z) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \frac{1}{z+n} + \int_1^{\infty} t^{z-1} e^{-t}\, dt.$$

Then $\Gamma(z)$ is a meromorphic function having simple poles at $0, -1, -2, \ldots$,and

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t}\, dt$$

for $\mathrm{Re}(z) > 0$. Also $\Gamma(z) \neq 0$ for all z. The subroutines CGAMMA and DCGAMA are available for computing $\Gamma(z)$ and $\ln \Gamma(z)$ when z is complex, and the functions GAMMA, GAMLN, DGAMMA, and DGAMLN are available for computing $\Gamma(z)$ and $\ln \Gamma(z)$ when z is real. DCGAMA, DGAMMA, and DGAMLN are double precision procedures.

CALL CGAMMA(MO, z, w)

MO is an integer, z a complex number satisfying $z \neq 0, -1, -2, \ldots$, and w a complex variable. When CGAMMA is called, w is assigned the value $\Gamma(z)$ if MO $= 0$ and the value $\ln \Gamma(z)$ if MO $\neq 0$.

Error return. If $z = 0, -1, -2, \ldots$, or if $\mathrm{Re}(z) < 0$ and $\mathrm{Re}(z)$ is too large for $\ln \Gamma(z)$ to be computed, then w is assigned the value 0.

Programming. CGAMMA calls the functions REXP, SPMPAR, and IPMPAR. CGAMMA was written by A. H. Morris.

References.

(1) Kuki, Hirondo, "Complex Gamma Function with Error Control," ***Comm. ACM*** 15 (1972), pp. 262–267.

(2) Spira, Robert, "Calculation of the Gamma Function by Stirling's Formula," ***Math Comp.*** 25 (1971), pp. 317–322.

GAMMA(x)

The argument x is a real number. If $\Gamma(x)$ can be computed then GAMMA(x) is assigned the value $\Gamma(x)$. Otherwise, if $\Gamma(x)$ cannot be computed, then GAMMA(x) is set to 0.

Algorithm. If $|x| < 15$ then x is reduced to the interval $[1, 2)$ by $\Gamma(a+1) = a\,\Gamma(a)$, and a rational minimax approximation is employed. If $x < -15$ then

(1) $$\Gamma(x) = \frac{\pi}{\sin(\pi x)\Gamma(1-x)}$$

is applied. For $x \geq 15$

$$\ln \Gamma(x) = (x - \tfrac{1}{2}) \ln x - x + \tfrac{1}{2} \ln(2\pi) + \Delta(x) \tag{2}$$

is computed where $\Delta(x)$ is a minimax approximation. The function $\Delta(x)$ is evaluated in single precision, and a double precision value is obtained for $\ln x$. This yields a double precision value for $\ln \Gamma(x)$. If $\ln \Gamma(x) = \alpha + \delta$ where α is the leading portion of $\ln \Gamma(x)$, then $\Gamma(x)$ is set to $e^{\alpha}(1 + \delta)$. This is permissible since $1 + \delta$ is the portion of the Taylor series expansion for e^{δ} that is significant.

The logarithm $\ln x$ is evaluated as follows: Let n be the largest integer less than or equal to x, and let $t = (x - n)/(x + n)$. Then $x = n(1 + t)/(1 - t)$ so that $\ln x = \ln n + \ln[(1 + t)/(1 - t)]$. Also $0 \leq t < 1/(2n)$. The function $\ln [(1 + t)/(1 - t)]$ is computed by a polynomial minimax approximation in single precision, and the value $\ln n$ is stored in double precision.

Precision. If $0 < x \leq 2$ then GAMMA(x) is accurate to within 2 units of the 14^{th} significant digit. If $x > 2$ then GAMMA(x) is accurate to within 3 units of the 14^{th} significant digit. Otherwise, GAMMA(x) is accurate to within 5 units of the 14^{th} significant digit.

Programming. GAMMA calls the functions GLOG and EXPARG. These functions were written by A. H. Morris. The functions SPMPAR and IPMPAR are also used.

GAMLN(x)

GAMLN$(x) = \ln \Gamma(x)$ for all positive real x.

Algorithm. See p. 379 and appendix D of the reference.

Precision. GAMLN(x) is accurate to within 2 units of the 14^{th} significant digit when GAMLN$(x) \neq 0$.

Reference. DiDonato, A. R. and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** 12 (1986), pp. 377–393.

Programming. GAMLN calls the function GAMLN1. These functions were written by A. H. Morris.

CALL DCGAMA(MO, Z, W)

MO is an integer, and Z and W are double precision arrays of dimension 2. It is assumed that $Z(1)$ and $Z(2)$ are the real and imaginary parts of a complex number z. If MO $= 0$ then the double precision value for $w = \Gamma(z)$ is computed. Otherwise, if MO $\neq 0$ then the double precision value for $w = \ln \Gamma(z)$ is computed. $W(1)$ and $W(2)$ contain the real and imaginary parts of w, respectively.

Error Return. If $z = 0, -1, -2, \ldots$, or if $\mathrm{Re}(z) < 0$ and $\mathrm{Re}(z)$ is too large for $\ln \Gamma(z)$ to be computed, then $W(1)$ and $W(2)$ are assigned the value 0.

Programming. DCGAMA calls the functions DREXP,DPMPAR, and IPMPAR. DCGAMA was written by A. H. Morris.

References.

(1) Kuki, Hirondo, "Complex Gamma Function with Error Control," ***Comm. ACM*** 15 (1972), pp. 262–267.

(2) Spira, Robert, "Calculation of the Gamma Function by Stirling's Formula," ***Math Comp***. 25 (1971), pp. 317–322.

DGAMMA(x)

The argument x is a double precision number. DGAMMA(x) is the double precision value for $\Gamma(x)$ when $\Gamma(x)$ can be computed. Otherwise, DGAMMA(x) is set to 0 when $\Gamma(x)$ cannot be computed.

Remark. DGAMMA must be declared in calling program to be of type DOUBLE PRECISION.

Algorithm. If $|x| \leq 20$ then x is reduced to the interval $[1,2)$ by $\Gamma(a+1) = a\Gamma(a)$, and minimax approximations are used. If $x < -20$ then (1) is applied, and if $x > 20$ then (2) is used. $\Delta(x)$ is computed by the power series corresponding to the Chebyshev expansion in the SLATEC library given by Wayne Fullerton (Los Alamos). The power series and minimax approximations were obtained by A. H. Morris.

Precision. If $0 < x \leq 2$ then DGAMMA(x) is accurate to within 1 unit of the 28^{th} significant digit. Otherwise, if $x > 2$ then DGAMMA(x) is accurate to within 1 unit of the 25^{th} significant digit.

Programming. DGAMMA calls the functions DGAM1, DPDEL, DSIN1, and DXPARG. These functions were written by A. H. Morris. The functions DPMPAR and IPMPAR are also used.

DGAMLN(x)

The argument x is a double precision positive number. DGAMLN(x) is the double precision value for $\ln \Gamma(x)$.

Remark. DGAMLN must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $.5 \leq x \leq 2.5$ then minimax approximations are used, and if $x > 10$ then (2) is applied. $\Delta(x)$ is computed by the power series corresponding to the Chebyshev expansion in the SLATEC library given by Wayne Fullerton (Los Alamos). The power series and minimax approximations were obtained by A. H. Morris.

Precision. DGAMLN(x) is accurate to within 2 units of the 28^{th} significant digit when DGAMLN(x) $\neq 0$.

Programming. DGAMLN calls the functions DPDEL, DGMLN1, DGAM1, and DLNREL. These functions were written by A. H. Morris.

DIGAMMA FUNCTION

For any complex $z \neq 0, -1, -2, \ldots$ the digamma (or psi) function $\psi(z)$ is defined by

$$\psi(z) = \Gamma'(z)/\Gamma(z)$$

where $\Gamma(z)$ is the gamma function. For real $x > 0, \psi(x)$ is an increasing function having a zero at the point $x_0 = 1.4616\ 32144\ 96836$. The subroutines CPSI and DCPSI are available for computing $\psi(z)$ when z is complex, and the functions PSI and DPSI are available for computing $\psi(z)$ when z is real. DCPSI and DPSI are double precision procedures.

CALL CPSI(z, w)

The argument z is a complex number satisfying $z \neq 0, -1, -2, \ldots$, and w is a complex variable. When CPSI is called, w is assigned the value $\psi(z)$.

Error Return. If $z = 0, -1, -2, \ldots$, or if $\mathrm{Re}(z) < 0$ and $\mathrm{Re}(z)$ is too large for $\psi(z)$ to be computed, then w is assigned the value 0.

Algorithm. If $z = x + iy$ satisfies $x \geq 0$ and $|z| \geq 6$, then the asymptotic expansion

$$\psi(z) = \ln z - \frac{1}{2z} - \sum_{m=1}^{\infty} \frac{B_{2m}}{2mz^{2m}} \tag{1}$$

is employed. Otherwise, if $x \geq 0$ then the smallest nonnegative integer n is found for which $|z + n| \geq 6$, and the relation

$$\psi(z) = -\sum_{j=0}^{n-1} \frac{1}{z+j} + \psi(z+n)$$

is applied. When $x < 0$ then

$$\psi(z) = \psi(1-z) - \pi\cot(\pi z) \tag{2}$$

is also used.

Programming. CPSI calls the functions REXP, SPMPAR, and IPMPAR. CPSI was written by A. H. Morris.

PSI(x)

The argument x is a real number. If $\psi(x)$ can be computed then PSI(x) is assigned the value $\psi(x)$. Otherwise, if $\psi(x)$ cannot be computed then PSI(x) is set to 0.

Precision. If $x > 0$ then PSI(x) is accurate to within 2 units of the 14^{th} significant digit when PSI$(x) \neq 0$.

Programming. PSI calls the functions SPMPAR and IPMPAR. PSI was written at Argonne National Laboratory for the FUNPACK package of special function subroutines. PSI was modified by A. H. Morris.

Reference. Cody, W. J., Strecok, A. J., and Thacher, H. C., "Chebyshev Approximations for the Psi Function," ***Math Comp***. 27 (1973), pp. 123–127.

CALL DCPSI(Z,W)

Z and W are double precision arrays of dimension 2. It is assumed that $Z(1)$ and $Z(2)$ are the real and imaginary parts of a complex number z. When DCPSI is called the double precision value for $w = \psi(z)$ is computed. $W(1)$ and $W(2)$ contain the real and imaginary parts of w, respectively.

Error Return. If $z = 0, -1, -2, \ldots$, or if $\mathrm{Re}(z) < 0$ and $\mathrm{Re}(z)$ is too large for $\psi(z)$ to be computed, then $W(1)$ and $W(2)$ are assigned the value 0.

Programming. DCPSI calls the functions DREXP, DPMPAR, and IPMPAR. DCPSI was written by A. H. Morris.

DPSI(x)

The argument x is a double precision number. If $\psi(x)$ can be computed then DPSI(x) is the double precision value for $\psi(x)$. Otherwise, if $\psi(x)$ cannot be computed, then DPSI(x) is set to 0.

Remark DPSI must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. If $|x| < 10$ then x is reduced to the interval $[1,2)$ by $\psi(a+1) = \frac{1}{a} + \psi(a)$, and the Chebyshev expansion in the SLATEC library given by Wayne Fullerton (Los Alamos) is used when $|x - x_0| > 2 \cdot 10^{-2}$. Otherwise, if $|x - x_0| < 2 \cdot 10^{-2}$ then the Taylor series around the zero x_0 is used. The coefficients for the Taylor series were obtained by A. H. Morris. If $x < -10$ then (2) is applied, and if $x \geq 10$ then $\psi(x) \sim \ln x$ is computed by the power series corresponding to the Chebyshev expansion in the SLATEC library given by Wayne Fullerton. The power series was obtained by A. H. Morris.

Precision. If $0 < x \leq 1$ or $x \geq 2$ then DPSI(x) is accurate to within 2 units of the 28^{th} significant digit. Otherwise, if $1 < x < 2$ then DPSI(x) is accurate to within 5 units of the 28^{th} significant digit when DPSI$(x) \neq 0$.

Programming. DPSI calls the functions DCSEVL, DPSI0, DPMPAR, and IPMPAR. DPSI was written by A. H. Morris.

DERIVATIVES OF THE DIGAMMA FUNCTION

The following subroutine is available for computing the digamma function $\psi(x)$ and its derivatives for real $x > 0$.

CALL PSIDF$(x, n, m, W, \text{IERR})$

Let $w_0 = -\psi(x)$ and $w_n = \dfrac{(-1)^{n+1}}{n!}\psi^{(n)}(x)$ where $\psi^{(n)}(x)$ is the n^{th} derivative of $\psi(x)$ $(n = 1, 2, \ldots)$.

The argument x is a positive real number, n and m are integers where $n \geq 0$ and $m \geq 1$, and W is an array of dimension m or larger. When PSIDF is called, w_{n+i-1} is computed and stored in $W(i)$ for $i = 1, \ldots, m$.

IERR is an integer variable that reports the status of the results. When PSIDF terminates, IERR has one of the following values:

IERR = 0 The values $w_n, \ldots, w_{n+m-i}$ were obtained and stored in W.
IERR = 1 (Input error) $x \leq 0, n < 0$, or $m < 1$.
IERR = 2 (Overflow) Either x is too small or $n + m - 1$ too large.
IERR = 3 (Underflow) Either x is too large or $n + m - 1$ too large.
IERR = 4 This setting can occur only if $n + m - 1 > 100$. When it occurs, w_{n+m-1} cannot be computed since $n + m - 1$ is too large for x. (However, w_{n+m-1} may be computable for larger values of x.)

Remark. Even though PSIDF may be used for computing $w_0 = -\psi(x)$, its primary purpose is for computing the scaled derivatives $w_n (n \geq 1)$ of $\psi(x)$. For $x \geq 3$, w_0 is accurate to within 2 units of the 14^{th} significant digit. However, far less accuracy is frequently obtained when $x < 3$. In the worst case, all relative accuracy is lost for w_0 when x is sufficiently near the zero $x_0 = 1.4616\ldots$ of $\psi(x)$. For $x < 3$, the function PSI should be used for computing $\psi(x)$.

Precision. For $1 \leq n \leq 30$ and $x \leq 1000$, w_n is accurate to within 1.5 units of the 12^{th} significant digit.

Programming. PSIDF employs the functions IPMPAR, SPMPAR, EPSLN, and EXPARG. PSIDF is a modification by A. H. Morris of the subroutine PSIFN, written by Donald E. Amos (Sandia Laboratories).

Reference Amos, D. E., "Algorithm 610. A Portable Fortran Subroutine for Derivatives of the Psi Function," *ACM Trans. Math Software* **9** (1983), pp. 494–502.

INCOMPLETE GAMMA RATIO FUNCTIONS

For $a > 0$ and $x \geq 0$ let $P(a,x)$ and $Q(a,x)$ denote the functions defined by

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt$$

$$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1}\, dt.$$

Then $0 \leq P(a,x) \leq 1$ and $P(a,x) + Q(a,x) = 1$. Also, $P(a,x) \to 1$ and $Q(a,x) \to 0$ for $x > 0$ when $a \to 0$. Hence, we may define $P(0,x) = 1$ and $Q(0,x) = 0$ for $x > 0$. The subroutines GRATIO and DGRAT are available for computing $P(a,x)$ and $Q(a,x)$, and the auxiliary functions RCOMP and DRCOMP are provided for computing $e^{-x}x^a/\Gamma(a)$.

CALL GRATIO(a, x, P, Q, i)

It is assumed that $a \geq 0$ and $x \geq 0$, where a and x are not both 0. P and Q are variables. GRATIO assigns P the value $P(a,x)$ and Q the value $Q(a,x)$. The argument i may be any integer. This argument specifies the desired accuracy of the results. If $i = 0$ then the user is requesting as much accuracy as possible (up to 14 significant digits). Otherwise, if $i = 1$ then accuracy is requested to within 1 unit of the 6^{th} significant digit, and if $i \neq 0, 1$ then the accuracy is requested to within 1 unit of the 3^{rd} significant digit.

Error Return. P is assigned the value 2 when a or x is negative, when $a = x = 0$, or when $P(a,x)$ and $Q(a,x)$ are indeterminant. $P(a,x)$ and $Q(a,x)$ are indeterminant when $x \approx a$ and a is exceedingly large. On the CDC 6000–7000 series computers this occurs when $|x/a - 1| \leq 10^{-14}$ and $a \geq 6.6\text{E}25$.

Programming. GRATIO calls the functions ERF, ERFC1, REXP, RLOG, RCOMP, GAM1, GAMMA, and SPMPAR. GAMMA employs the functions GLOG, EXPARG, and IPMPAR. GRATIO was written by A. H. Morris.

Reference. DiDonato, A. R. and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math. Software*** **12** (1986), pp. 377–393.

RCOMP(a, x)

RCOMP$(a,x) = e^{-x}x^a/\Gamma(a)$ for $a > 0$ and $x \geq 0$.

Algorithm. See page 378 of the reference.

Programming. RCOMP employs the functions EXPARG, GAMMA, GAM1, GLOG, and RLOG. These functions were written by A. H. Morris. The functions SPMPAR and IPMPAR are also used.

Reference. DiDonato, A. R. and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and their Inverse," ***ACM Trans. Math Software* 12** (1986), pp. 377–393.

CALL DGRAT$(a, x, P, Q,$IERR)

The arguments a and x are nonnegative double precision numbers, where a and x are not both 0. P and Q are double precision variables, and IERR an integer variable. When DGRAT is called, if $P(a,x)$ and $Q(a,x)$ can be computed then IERR is set to 0 and P and Q are assigned the double precision values for $P(a,x)$ and $Q(a,x)$.

Error Return. If $P(a,x)$ and $Q(a,x)$ cannot be computed then P is set to 2 and IERR is assigned one of the following values:

IERR = 1 a or x is negative.
IERR = 2 $a = x = 0$.
IERR = 3 $P(a,x)$ and $Q(a,x)$ are indeterminant. This setting occurs only when $x \approx a$ and a is exceedingly large.

Algorithm. A modification of the algorithm given in the reference is used. The functions $c_k(z)$ appearing in the Temme Expansion on page 381 are computed by minimax approximations designed by A. H. Morris.

Programming. DGRAT employs the subroutines DGR29 and DGR17, and the functions IPMPAR, DPMPAR, DXPARG, DREXP, DRLOG, DSIN1, DERF, DERFC1, DERFC0, DGAMMA, DGAM1, DRCOMP, and DPDEL. DGRAT was written by A. H. Morris.

Reference. DiDonato, A.R. and Morris, A.H.,"Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software* 12** (1986), pp. 377-393.

DRCOMP(a, x)

The arguments a and x are double precision numbers where $a > 0$ and $x \geq 0$. DRCOMP(a,x) is the double precision value for $e^{-x}x^a/\Gamma(a)$.

Remark. DRCOMP must be declared to be of type DOUBLE PRECISION in the calling program.

Programming. DRCOMP employs the functions IPMPAR, DPMPAR, DXPARG, DRLOG, DSIN1, DGAMMA, DGAM1, and DPDEL. DRCOMP was written by A. H. Morris.

INVERSE INCOMPLETE GAMMA RATIO FUNCTION

For $a > 0$ and $x \geq 0$ let $P(a,x)$ and $Q(a,x)$ denote the incomplete gamma ratio functions defined by

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt$$
$$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1}\, dt.$$

Then $0 \leq P(a,x) \leq 1$ and $P(a,x) + Q(a,x) = 1$. If we are given a, p, and q where $a > 0, 0 \leq p \leq 1$, and $p + q = 1$, then the subroutines GAMINV and DGINV are available for obtaining the value $x \geq 0$ for which $P(a,x) = p$ and $Q(a,x) = q$.

CALL GAMINV$(a, X, x_0, p, q, \text{IND})$

X is a variable. If $p = 0$ then X is assigned the value 0, and if $q = 0$ then X is set to the largest floating point number available. Otherwise, if $p \neq 0$ and $q \neq 0$ then GAMINV attempts to obtain a solution x for $P(a,x) = p$ and $Q(a,x) = q$ that is correct to at least 10 significant digits.[1] If a solution is obtained then it is stored in X. The solution is normally computed by Schroder iteration. The argument x_0 is an optional initial approximation for x. If the user does not wish to supply an initial approximation then set $x_0 \leq 0$.

IND is a variable that reports the status of the results. When GAMINV terminates, IND has one of the following values:

IND = 0 The solution was obtained. Iteration was not used.
IND $\geq$ 1 The solution was obtained. IND iterations were performed.
IND = −2 (Input error) $a \leq 0$.
IND = −3 No solution was obtained. The ratio Q/a is too large.
IND = −4 (Input error) $p < 0$, $q < 0$, or $p + q \neq 1$.
IND = −6 20 iterations were performed. The most recent value obtained for x is stored in X. This cannot occur if $x_0 \leq 0$.
IND = −7 Iteration failed. No value is given for x. This may occur when $x \approx 0$.
IND = −8 A value for x is stored in X, but the routine is not certain of its accuracy. Iteration cannot be performed in this case. If $x_0 \leq 0$ then this can occur only when $p \approx 0$ or $q \approx 0$. If $x_0 > 0$ then this can occur when $a \approx x$ and a is exceedingly large (say $a \geq 10^{20}$).

Remark. If $x_0 \leq 0$ then 3 or fewer iterations are required.

Programming. GAMINV employs the subroutines GRATIO and PNI, and functions ERF, ERFC1, ERFI, REXP, RLOG, ALNREL ,GAM MA, GLOG, GAM1, GAMLN, GAMLN1,

[1] If a k digit floating point arithmetic is being used where $k < 10$, then the routine attempts to obtain a solution that is correct to machine accuracy.

RCOMP, EXPARG, IPMPAR, and SPMPAR. GAMINV was written by A. H. Morris.

Reference. DiDonato, A.R. and Morris, A.H., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," ***ACM Trans. Math Software*** **12** (1986), pp. 377–393.

CALL DGINV$(a, X, p, q$,IND)

The arguments a, p, q are double precision numbers and X a double precision variable. If $p = 0$ then X is assigned the value 0, and if $q = 0$ then X is set to the largest double precision number available. Otherwise, if $p \neq 0$ and $q \neq 0$ then DGINV attempts to obtain a double precision solution x for $P(a, x) = p$ and $Q(a, x) = q$ that is correct to machine accuracy. If a solution is obtained then it is stored in X.

IND is a variable that reports the status of the results. When DGINV terminates, IND has one of the following values:

IND = 0 The solution was obtained. Iteration was not used.
IND $\geq$ 1 The solution was obtained. IND iterations were performed.
IND = −2 (Input error) $a \leq 0$.
IND = −3 No solution was obtained. The ratio Q/a is too large.
IND = −4 (Input error) $p < 0$, $q < 0$, or $p + q \neq 1$.
IND = −6 10 iterations were performed. The most recent value obtained for x is stored in X. (This setting should never occur.)
IND = −7 Iteration failed. No value is given for x. This may occur when $x \approx 0$.
IND = −8 A value for x is stored in X, but the routine is not certain of its accuracy. Iteration cannot be performed in this case. This can occur only when $p \approx 0$ or $q \approx 0$

Remarks. Schroder iteration is normally used. On the CDC 6000–7000 series computers 4 or fewer interations are required.

Programming. DGINV employs the subroutines GAMINV, GRATIO, PNI, DGR17, DPNI, DGR29, and DGRAT, and functions RCOMP, DRCOMP, SPMPAR, DPMPAR, EXPARG, DXPARG, REXP, DREXP, ALNREL, DLNREL, RLOG, DRLOG, DSIN1, ERF, EFRC1, DERF, DERFC1, DEFRCO, ERFI, DERFI, GAMMA, GLOG, GAM1, GAMLN, DGAM1, GAMLN1, DGAMMA, DPDEL, DGAMLN, DGMLN1, and IPMPAR. DGINV was written by A. H. Morris.

LOGARITHM OF THE BETA FUNCTION

For $a, b > 0$ the beta function $B(a, b)$ can be defined by

$$B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}\,dt.$$

From this it follows that $B(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ where $\Gamma(a)$ is the gamma function. The functions BETALN and DBETLN are available for computing ln $B(a, b)$. DBETLN is a double precision function.

BETALN(a, b)

BETALN(a, b) = ln $B(a, b)$ for $a, b > 0$.

Algorithm. See pages 19–21 of the reference.

Precision. BETALN(a, b) is accurate to within 4 units of the 14^{th} significant digit when $a, b \geq 1$ and BETALN$(a, b) \neq 0$. In particular, when $a, b \geq 15$, BETALN(a, b) is accurate to within 2 units of the 14^{th} significant digit.

Programming. BETALN employs the functions ALNREL, ALGDIV, BCORR, GAMLN, GAMLN1, and GSUMLN. These functions were written by A. H. Morris.

Reference. DiDonato, A. R. and Morris, A. H., "Algorithm 708 Significant Digit Computation of the Incomplete Beta Function Ratios," ***ACM Trans. Math Software*** **18** (1992), pp. 360–373.

DBETLN(a, b)

The arguments a and b are positive double precision numbers. DBETLN(a, b) is the double precision value for ln $B(a, b)$.

Remark. DBETLN must be declared in the calling program to be of type DOUBLE PRECISION.

Algorithm. The algorithm for ln $B(a, b)$ on pages 19–21 of the reference is used. $\Delta(x)$ is computed by the power series corresponding to the Chebyshev expansion in the SLATEC library given by Wayne Fullerton (Los Alamos). The power series was obtained by A. H. Morris

Programming. DBETLN employs the functions DLNREL, DLGDIV, DBCORR, DPDEL, DGAMLN, DGMLN1, DGAM1 and DGSMLN. These functions were written by A. H. Morris. The function DPMPAR is also used.

Reference. DiDonato, A. R. and Morris, A. H., "Algorithm 708 Significant Digit Computation of the Incomplete Beta Function Ratios," ***ACM Trans. Math Software*** 18 (1992), pp. 360–373.

INCOMPLETE BETA FUNCTION

For $a, b > 0$ and $0 \leq x \leq 1$ the incomplete beta function is defined by

$$I_x(a,b) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}\, dt$$

where $B(a,b)$ is the beta function. Then we note that $0 \leq I_x(a,b) \leq 1$ and

$$\lim_{a \to 0} I_x(a,b) = 1 \text{ for } x \neq 0$$
$$\lim_{b \to 0} I_x(a,b) = 0 \text{ for } x \neq 1.$$

These limits permit $I_x(a,b)$ to be defined to be 1 when $a = 0$ and $b \neq 0, x \neq 0$, and for $I_x(a,b)$ to be defined to be 0 when $b = 0$ and $a \neq 0, x \neq 1$. The subroutine BRATIO is available for computing $I_x(a,b)$ for arbitrary $a, b \geq 0$, and the subroutine ISUBX is available for computing $I_x(a,b)$ for the highly specialized case when a and b are integers or half-integers. Also, the auxiliary function BRCOMP is provided for computing $x^a y^b / B(a,b)$ when $0 < x < 1$ and $y = 1 - x$.

CALL BRATIO$(a, b, x, y, W,$ W1,IERR)

It is assumed that $a \geq 0, b \geq 0, 0 \leq x \leq 1$, and $y = 1 - x$. W, W1, and IERR are variables. If no input errors are detected then IERR is set to 0, W is assigned the value $I_x(a,b)$, and W1 is assigned the value $1 - I_x(a,b)$.

Error Return. When an input error is detected, then W and W1 are assigned the value 0 and IERR is set to one of the following values:

IERR = 1 if $a < 0$ or $b < 0$
IERR = 2 if $a = b = 0$
IERR = 3 if $x < 0$ or $x > 1$
IERR = 4 if $y < 0$ or $y > 1$
IERR = 5 if $x + y \neq 1$
IERR = 6 if $x = a = 0$
IERR = 7 if $y = b = 0$

Programming. BRATIO employs the subroutines BGRAT and GRAT1, and the functions ALGDIV, ALNREL, APSER, BASYM, BCORR, BETALN, BFRAC, BPSER, BRCOMP, BRCMP1, BUP, ERF, ERFC1, FPSER, GAMLN, GAMLN1, GAM1, GSUMLN, PSI, ESUM, EXPARG, REXP, and RLOG1. These subroutines and functions were written by A. H. Morris. The functions SPMPAR and DPMPAR are also used.

Reference. DiDonato, A. R. and Morris, A. H., "Algorithm 708 Significant Digit Computation of the Incomplete Beta Function Ratios," *ACM Trans. Math. Software* **18** (1993), pp. 360-373.

CALL ISUBX$(a, b, x, W,$ IERR,EPS)

It is assumed that a, b, and x satisfy the following restrictions:

(1) $a > 0, b > 0$, and $x \geq 0$
(2) $a \geq 1/2, 1/2 \leq b \leq 70$, and $x \leq 1$
(3) a and b are integers or half-integers

EPS specifies the (absolute) accuracy that is desired. W is a real variable and IERR an integer variable. When ISUBX is called, if there are no input errors then W is assigned the value $I_x(a, b)$ and IERR is assigned the value 1.

Error Return. If an error is detected then IERR is assigned one of the following values:

IERR = 2 if restrictions (1) are violated.
IERR = 3 if restrictions (2) are violated or a is too large.
IERR = 4 if restrictions (3) are violated.

Also W is assigned the value 0.

Remarks. ISUBX was designed for a maximum precision EPS $= 10^{-10}$

Programming. ISUBX employs the functions ALGDIV, ALNREL, BLND, IPMPAR, and LOGAM. ISUBX was written by A. H. Morris.

Reference. DiDonato, A. R. and Jarnagin, M. P., "The Efficient Calculation of the Incomplete Beta-function Ratio for Half-Integer Values of the Parameter a,b," ***Math Comp***. 21 (1967), pp. 652–662.

BRCOMP(a, b, x, y)

BRCOMP$(a, b, x, y) = x^a y^b / B(a, b)$ for $a, b > 0$ and $x, y \geq 0$ where $x + y = 1$.

Algorithm. See pages 19–21 of the reference.

Programming. BRCOMP employs the functions ALGDIV, ALNREL, BCORR, BETALN, GAM1, GAMLN, GAMLN1, GSUMLN, and RLOG1. These functions were written by A. H. Morris.

Reference. DiDonato, A. R. and Morris, A. H., "Algorithm 708 Significant Digit Computation of the Incomplete Beta Function Ratios," ***ACM Trans. Math Software*** 18 (1992), pp. 360–373.

BESSEL FUNCTION $J_\nu(z)$

If ν is complex then $J_\nu(z)$ is defined by

$$J_\nu(z) = \sum_{k=0}^{\infty} \frac{(-1)^k (z/2)^{\nu+2k}}{k!\,\Gamma(\nu+k+1)}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. $J_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$, and $J_\nu(z)$ is an entire function of ν for any fixed z. $J_\nu(z)$ can be extended to the negative real axis by letting $-\pi < \arg(z) \leq \pi$. If ν is an integer then $J_\nu(z)$ is also defined at 0 and is an entire function of z. The following subroutines are available for computing $J_\nu(z)$.

CALL CBSSLJ(z, ν, w)

The arguments z and ν are complex numbers and w is a complex variable. It is assumed that $z \neq 0$. When CBSSLJ is called, w is assigned the value $J_\nu(z)$.

Precision. The real and imaginary parts of w are normally accurate to 11–12 significant digits when $Re[J_\nu(z)]$ and $Im[J_\nu(z)]$ are not near 0. The only exception to this is when $Re(z) \approx 0$, $Im(\nu) \approx 0$, and $14 < |z| < 17.5 + \frac{1}{2}|\nu|^2$. Then $Re(w)$ is accurate to 11–12 significant digits, but all accuracy for the small nonzero value $Im(w)$ may be lost.

Algorithm. A modification of the Hershey procedure for applying the Maclaurin, Debye, and asymptotic expansions is used.

Programmimg. CBSSLT employs the subroutines CREC, CGAMMA, CBJM, CBDB, CBJA and functions IPMPAR, SPMPAR, CPABS, CDIV, EXPARG, REXP, SIN1, COS1, CGAM0, GAMMA, GLOG. CBSSLJ was written by Andrew H. Van Tuyl and A. H. Morris.

Reference. Hershey, A. V., ***Computation of Special Functions***, Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978, pp. 33–42.

CALL BSSLJ(z, n, w)

The argument z is a complex number, n is an integer, and w is a complex variable. When BSSLJ is called, w is assigned the value $J_n(z)$.

Precision. BSSLJ is accurate to within $5 \cdot 10^{-14}$ for real $0 < z < 35$ and $n = 0, 1$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

CALL BESJ(x, α, n, W, k)

The arguments x and α are nonnegative real numbers, n is a positive integer, and W is an array of dimension n or larger. When BESJ is called $J_{\alpha+i-1}(x)$ is computed and stored in $W(i)$ for $i = 1, \ldots, n$.

The argument k is an integer variable that is set by the routine. If all $J_{\alpha+i-1}(x)$ are computed then k is set to 0. Otherwise, k is assigned one of the following values:

$k = -1$ The argument x is negative.
$k = -2$ The argument α is negative.
$k = -3$ The requirement $n \geq 1$ is violated.
$k > 0$ The last k components of W have been set to 0 because of underflow.

Precision. For $0 < x \leq 35$ and $0 \leq \alpha \leq 1$, BESJ is accurate to within $8 \cdot 10^{-13}$.

Programming. BESJ calls the subroutines ASJY and JAIRY, and the functions GAMLN, SPMPAR, and IPMPAR. The subroutines were written by Donald E. Amos, Sharon L. Daniel, and M. Katherine Weston (Sandia Laboratories).

References.

(1) Amos, D.E., Daniel, S. L., and Weston, M. K., *CDC* 6600 ***Subroutines for Bessel Functions*** $J_\nu(x), x \geq 0, \nu \geq 0$ ***and Airy Functions*** $A_i(x), A_i'(x), -\infty < x < \infty$. Report SAND 75-0147, Sandia Laboratories, Albuquerque, New Mexico, 1975.

(2) _____, "CDC 6600 Subroutines IBESS and JBESS for Bessel Functions $I_\nu(x)$ and $J_\nu(x), x \geq 0, \nu \geq 0$," ***ACM Trans. Math Software*** **3** (1977), pp. 76-92.

BESSEL FUNCTION $Y_\nu(z)$

If ν is any complex number not an integer, then $Y_\nu(z)$ can be defined by

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. For any integer n we can also define $Y_n(z) = \lim_{\nu \to n} Y_\nu(z)$. Then for any complex $\nu, Y_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$. Also, $Y_\nu(z)$ is an entire function of ν for any fixed z. The following subroutine is available for computing $Y_\nu(z)$ when ν is an integer.

CALL BSSLY(z, n, w)

The argument z is a complex number, n is an integer, and w is a complex variable. It is assumed that $|\arg(z)| < \pi$. When BSSLY is called, w is assigned the value $Y_n(z)$.

Precision. If $.005 \leq x \leq .785$ then $Y_0(x)$ and $Y_1(x)$ are accurate to within 3 units of the 14^{th} significant digit. Otherwise, if $x > .785$ then $Y_0(x)$ and $Y_1(x)$ are accurate to within $4 \cdot 10^{-14}$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

MODIFIED BESSEL FUNCTION $I_\nu(z)$

If ν is complex then $I_\nu(z)$ is defined by

$$I_\nu(z) = \sum_{k=0}^{\infty} \frac{(z/2)^{\nu+2k}}{k!\,\Gamma(\nu+k+1)}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. $I_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$, and $I_\nu(z)$ is an entire function of ν for any fixed z. $I_\nu(z)$ can be extended to the negative real axis by letting $-\pi < \arg(z) \leq \pi$. If ν is an integer then $I_\nu(z)$ is also defined at 0 and is an entire function of z. The following subroutines are available for computing $I_\nu(z)$.

CALL CBSSLI(z, ν, w)

The arguments z and ν are complex numbers and w is a complex variable. It is assumed that $z \neq 0$. When CBSSLI is called, w is assigned the value $I_\nu(z)$.

Algorithm. The evaluation of $I_\nu(z)$ is reduced to $J_\nu(z)$ using

$$I_\nu(z) = e^{\nu\pi i/2}\, J_\nu(ze^{-\pi i/2}) \qquad -\tfrac{\pi}{2} < \arg(z) \leq \pi$$
$$I_\nu(z) = e^{-\nu\pi i/2}\, J_\nu(ze^{\pi i/2}) \qquad -\pi < \arg(z) < \tfrac{\pi}{2}.$$

Programming. CBSSLI calls the subroutine CBSSLJ and functions EXPARG, COS1, and SIN1. Also, the functions and subroutines needed for CBSSLJ are used. CBSSLI was written by Andrew H. Van Tuyl and A. H. Morris

CALL BSSLI(MO, z, n, w)

MO is an integer, z a complex number, n an integer, and w a complex variable. If MO $\neq 0$ then it is assumed that $|\arg(z)| < \pi$. When BSSLI is called, w is assigned the value $I_n(z)$ if MO $= 0$ and the value $e^{-z} I_n(z)$ if MO $\neq 0$.

Precision. BSSLI is accurate to within 5 units of the 13^{th} significant digit for real $0 < z < 35$ and $n = 0, 1, \ldots 40$.

Programmer. Allen V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

CALL BESI$(x, \alpha, \mathrm{MO}, n, W, k)$

MO may be 1 or 2. The arguments x and α are nonnegative real numbers, n is a positive integer, and W is an array of dimension n or larger. When BESI is called, if MO

= 1 then $I_{\alpha+i-1}(x)$ is computed and stored in $W(i)$ for $i = 1, \ldots, n$. Otherwise, if MO = 2 then $e^{-x} I_{\alpha+i-1}(x)$ is computed and stored in $W(i)$

The argument k is an integer variable that is set by the routine. If all $I_{\alpha+i-1}(x)$ or $e^{-x} I_{\alpha+i-1}(x)$ are computed then k is set to 0. Otherwise, k is assigned one of the following values:

k = −1 The argument x is negative.
k = −2 The argument α is negative.
k = −3 The requirement $n \geq 1$ is violated.
k = −4 MO is not 1 or 2.
k = −5 The argument x is too large for MO = 1.
k > 0 The last k components of W have been set to 0 because of underflow.

Precision. For $0 < x \leq 35$ and $0 \leq \alpha \leq 1$, or $0 < x \leq 35$ and $\alpha = 1, 2 \ldots, 40$, $I_\alpha(x)$ is accurate to within 2 units of the 12^{th} significant digit.

Programming. BESI calls the subroutine ASIK and the functions GAMLN, SPMPAR, and IPMPAR. BESI and ASIK were written by D. E. Amos and S. L. Daniel (Sandia Laboratories).

References.

(1) Amos, D. E. and Daniel, S. L., ***A CDC*** 6600 ***Subroutine for Bessel Functions*** $I_\nu(x)$, $\nu \geq 0$,$x \geq 0$. Report SAND 75-0152, Sandia Laboratories, Albuquerque, New Mexico, 1975.

(2) Amos, D. E., Daniel, S. L., and Weston, M. K., "CDC 6600 Subroutines IBESS and JBESS for Bessel Functions $I_\nu(x)$ and $J_\nu(x)$, $x \geq 0$, $\nu \geq 0$," ***ACM Trans. Math Software*** **3** (1977), pp. 76-92.

MODIFIED BESSEL FUNCTION $K_\nu(z)$

If ν is any complex number not an integer, then $K_\nu(z)$ is defined by

$$(*) \qquad K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}$$

for any $z \neq 0$ in the conplex plane cut along the negative real axis. For any integer n we can also define $K_n(z) = \lim_{\nu \to n} K_\nu(z)$. Then for any complex ν, $K_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$. Also, $K_\nu(z)$ is an entire function of ν for any fixed z. $K_\nu(z)$ can be extended to the negative real axis by letting $-\pi < \arg(z) \leq \pi$. The following subroutines are available for computing $K_\nu(z)$.

CALL CBESK(z, ν, w)

The arguments z and ν are complex numbers and w is a complex variable. It is assumed that $z \neq 0$. When CBESK is called, w is assigned the value $K_\nu(z)$.

Algorithm. $K_\nu(z)$ is computed using $(*)$, the asymptotic expansion

$$(1) \qquad \begin{aligned} K_\nu(z) &= \left(\frac{\pi}{2z}\right)^{1/2} e^{-z}\left[1 + \sum_{k\geq 1}(\nu, k)(2z)^{-k}\right] \\ (\nu, k) &= \frac{(4\nu^2 - 1)\cdots(4\nu^2 - (2k-1)^2)}{4^k k!}, \end{aligned}$$

the relation

$$(2) \qquad K_{\nu+1}(z) = \frac{2\nu}{z}K_\nu(z) + K_{\nu-1}(z)$$

along with the Miller algorithm and the power series $K_\nu(z) = \sum_{k\geq 0} c_k f_k$ and $K_{\nu+1}(z) = \frac{2}{z}\sum_{k\geq 0} c_k(p_k - k f_k)$ given in the references, and the analytic continuation formulae

$$(3) \qquad \begin{aligned} K_\nu(z) &= e^{-\nu\pi i}K_\nu(e^{-\pi i}z) - i\pi I_\nu(e^{-\pi i}z) \quad (0 < \arg(z) \leq \pi) \\ K_\nu(z) &= e^{\nu\pi i}K_\nu(e^{\pi i}z) + i\pi I_\nu(e^{\pi i}z) \qquad (-\pi < \arg(z) \leq 0). \end{aligned}$$

Precision. Frequently, 11-13 digit accuracy is obtained. For $10^{-2} < |z| < 50$, CBESK maintains accuracy to within 3 units of the 8^{th} significant digit except when $Re[K_\nu(z)]$ or $Im[K_\nu(z)]$ is near 0, or occasionally when z is on the negative real axis. Let $\tau = |\nu|/|z|$. For $50 < |z| < 100$, $Re(w)$ and $Im(w)$ are normally accurate to within 3 units of the 8^{th} significant digit when $\tau > 1$ and $\tau < 1/2$. However, for $1/2 < \tau < 1$ and ν not real, accuracy gradually decreases as $|z|$ increases.

Programming. CBESK employs the subroutines CBKA, CBKI, CBKM, CBKML, CKPS,

CGAMMA, CBSSLI, CBSSLJ, CREC and functions CDIV, CGAM0, COS0, COS1, SIN0, SIN1, CPABS, CXP, EXPARG, SPMPAR, IPMPAR. Also, the functions and subroutines needed for CBSSLJ are used. CBESK was written by Andrew H. Van Tuyl and A. H. Morris.

References.

(1) Amos, D.E., ***Computation of Bessel Functions of Complex Argument***, Report SAND 83-0086, Sandia Laboratories, Albuquerque, New Mexico, 1983.

(2) Temme, N.M., "On the Numerical Evaluation of the Modified Bessel Function of the Third Kind," ***J. Comp. Physics*** **19** (1975), pp. 324-337.

CALL CBSSLK(z, r, w)

The argument z is a nonzero complex number, r a real number, and w a complex variable. When CBSSLK is called, w is assigned the value $K_r(z)$.

Algorithm. The algorithm employed by CBESK is used. For real r, $(*)$ is not needed.

Programming. CBSSLK employs the subroutines CBSSLJ, CKA, CKM, CKML, CREC and functions CPABS, COS0, CXP, GAM1, SIN0, SPMPAR, IPMPAR. Also, the functions and subroutines needed for CBSSLJ are used. CBSSLK was written by Andrew H. Van Tuyl and modified by A. H. Morris.

CALL BSSLK(MO, z, n, w)

MO is an integer, z a complex number, n an integer, and w a complex variable. It is assumed that $|\arg(z)| < \pi$. When BSSLK is called, w is assigned the value $K_n(z)$ if MO $= 0$ and the value $e^z K_n(z)$ if MO $\neq 0$.

Precision. BSSLK is accurate to within 6 units of the 14^{th} significant digit for real z and $n = 0, 1$.

Programmer. Allen V. Hershey.

Reference. Hershey, A. V., ***Computation of Special Functions***, Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginina, 1978.

AIRY FUNCTIONS

For any series $w = \sum_{n=0}^{\infty} a_n z^n$ satisfying the differential equation $w'' = zw$, it follows that $w = a_0 f(z) + a_1 g(z)$ where

$$f(z) = 1 + \sum_{n \geq 1} \frac{1 \cdot 4 \cdots (3n-2)}{(3n)!} z^{3n}$$

$$g(z) = 1 + \sum_{n \geq 1} \frac{2 \cdot 5 \cdots (3n-1)}{(3n+1)!} z^{3n+1}.$$

In particular, the Airy functions $Ai(z)$ and $Bi(z)$ are independent solutions of $w'' = zw$ where

$$Ai(z) = c_1 f(z) - c_2 g(z)$$
$$Bi(z) = \sqrt{3}\,[c_1 f(z) + c_2 g(z)]$$

for $c_1 = 3^{-2/3}/\Gamma(2/3)$ and $c_2 = 3^{-1/3}/\Gamma(1/3)$. $Ai(z)$ and $Bi(z)$ are entire functions.

The subroutines CAI and CBI are available for computing $Ai(z)$ and $Bi(z)$ when z is complex, and the functions AI, AIE, BI, and BIE are available for computing $Ai(z)$ and $Bi(z)$ when z is real. CAI and CBI also provide the dervatives $Ai'(z)$ and $Bi'(z)$ of $Ai(z)$ and $Bi(z)$.

CALL CAI(IND,z, w, w',IERR)

IND is an integer, z a complex number, and w and w' complex variables. When CAI is called, w is assigned the value $Ai(z)$ and w' the value $Ai'(z)$ when IND = 0. Otherwise, if IND $\neq$ 0 then $w = e^{\zeta} Ai(z)$ and $w' = e^{\zeta} Ai'(z)$ where $\zeta = \frac{2}{3} z^{3/2}$.

IERR is a variable that is set by the routine. When CAI terminates, IERR has one of the following values:

IERR = 0 The desired values were obtained.
IERR = 1 z is too large for the desired values to be computed. In this case w and w' are assigned the value 0.

Precision. For IND $\neq$ 0 the real and imaginary parts of w and w' are accurate to within 2 units of the 12^{th} significant digit except near points where they vanish.

Programming. CAI employs the subroutines AIRM, AII, AIA, JA, JMC, BIM, KA, KML, IMC, and BIM. These routines were written by Andrew H Van Tuyl and modified by A. H Morris. The subroutines CAPO and CREC, and functions CPABS, EXPARG, IPMPAR, and SPMPAR are also used.

CALL CBI(IND,z, w, w',IERR)

IND is an integer, z a complex number, and w and w' complex variables. When CBI is called, w is assigned the value $Bi(z)$ and w' the value $Bi'(z)$ when IND = 0. Otherwise, if IND $\neq 0$ then

$$w = \begin{cases} e^{-\zeta} Bi(z) & \text{if } |\arg(z)| \leq \pi/3 \\ e^{\zeta} Bi(z) & \text{otherwise} \end{cases}$$

$$w = \begin{cases} e^{-\zeta} Bi'(z) & \text{if } |\arg(z)| \leq \pi/3 \\ e^{\zeta} Bi'(z) & \text{otherwise} \end{cases}$$

where $\zeta = \frac{2}{3} z^{3/2}$.

IERR is a variable that is set by the routine. When CBI terminates, IERR has one of the following values:

IERR = 0 The desired values were obtained.
IERR = 1 z is too large for the desired values to be computed. In this case w and w' are assigned the value 0.

Precision. For IND $\neq 0$ the real and imaginary parts of w and w' are accurate to within 2 units of the 12^{th} significant digit except near points where they vanish.

Programming. CBI employs the subroutines AIRM, BII, BIA, IA, IMC, BIM, JA, JMC, and BJM. These routines were written by Andrew H. Van Tuyl and modified by A. H. Morris. The subroutine CREC and functions CPABS, EXPARG, IPMPAR, and SPMPAR are also used.

AI(x)

AI$(x) = Ai(x)$ for real x.

Algorithm. Rational minimax approximations are used. If $x < -1$ then R and θ are computed where $Ai(x) = R \sin(\pi/4 + \theta)$.

Precision. For $x \geq -1$, AI(x) is accurate to within 2 units of the 12^{th} significant digit when AI$(x) \neq 0$.

Programming. AI calls the subroutine AIMP and function EXPARG. These subprograms were written by A. H. Morris. The function IPMPAR is also used.

AIE(x)

If $x \geq 0$ then AIE$(x) = e^{\zeta} Ai(x)$ where $\zeta = \frac{2}{3} x^{3/2}$. Otherwise, if $x < 0$ then AIE$(x) = Ai(x)$.

Algorithm. Rational minimax approximations are used. If $x < -1$ then R and θ are computed where $Ai(x) = R \sin(\pi/4 + \theta)$.

Precision. For $x \geq -1$, AIE(x) is accurate to within 2 units of the 14^{th} significant digit

Programming. AIE calls the subroutine AIMP. AIE and AIMP were written by A. H. Morris.

BI(x)

BI(x) = $Bi(x)$ for real x. If x is a positive value for which $Bi(x)$ is too large to be computed, then BI(x) is assigned the value 0.

Algorithm. Rational minimax approximations are used. If $x < -1$ then R and θ are computed where $Bi(x) = R\cos(\pi/4 + \theta)$.

Precision. For $x \geq -1$, BI(x) is accurate to within 2 units of the 12^{th} significant digit when BI$(x) \neq 0$.

Programming. BI calls the subroutine AIMP and function EXPARG. These subprograms were written by A. H. Morris. The function IPMPAR is also used.

BIE(x)

If $x \geq 0$ then BIE$(x) = e^{-\varsigma} Bi(x)$ where $\varsigma = \frac{2}{3}x^{3/2}$. Otherwise, if $x < 0$ then BIE(x) = $Bi(x)$.

Algorithm. Rational minimax approximations are used. If $x < -1$ then R and θ are computed where $Bi(x) = R\cos(\pi/4 + \theta)$.

Precision. For $x \geq -1$, BIE(x) is accurate to within 2 units of the 14^{th} significant digit.

Programming. BIE calls the subroutine AIMP. BIE and AIMP were written by A. H. Morris.

COMPLETE COMPLEX ELLIPTIC INTEGRALS OF THE FIRST AND SECOND KINDS

If k is complex then the complete elliptic integrals of the first and second kinds can be defined by

$$K(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 t)^{-1/2}\, dt$$
$$E(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 t)^{1/2}\, dt$$

for $|\arg(1-k^2)| < \pi$. $K(k)$ and $E(k)$ can be extended to $-\pi \le \arg(1-k^2) < \pi$. For $|k| < 1$

$$K(k) = \frac{\pi}{2} \sum_{n \ge 0} c_n k^{2n} \tag{1}$$
$$E(k) = -\frac{\pi}{2} \sum_{n \ge 0} c_n \frac{k^{2n}}{2n-1}$$

where $c_n = \left[\dfrac{(2n)!}{4^n (n!)^2}\right]^2$. Also, if $\ell^2 = 1 - k^2$ where $|\ell| \le 1$ and $-\pi \le \arg(\ell^2) < \pi$, then

$$K(k) = \frac{1}{\pi} K(\ell) \ln\frac{16}{\ell^2} - \sum_{n \ge 1} c_n \sum_{m=1}^{n} \frac{\ell^{2n}}{m(2m-1)} \tag{2}$$
$$E(k) = \frac{1}{\pi}[K(\ell) - E(\ell)] \ln\frac{16}{\ell^2} - \sum_{n \ge 1} c_n \frac{2n}{2n-1} \sum_{m=1}^{n} \frac{\ell^{2n}}{m(2m-1)} + \sum_{n \ge 0} c_n \frac{\ell^{2n}}{(2n-1)^2}.$$

The function CK is available for computing $K(k)$, and the subroutine CKE for computing $K(k)$ and $E(k)$.

CK(k, ℓ)

CK$(k, \ell) = K(k)$ for any complex k and ℓ where $k^2 + \ell^2 = 1$ and $\ell \neq 0$. CK is a complex valued function which must be declared in the calling program to be of type COMPLEX.

Error Return. CK$(k, \ell) = 0$ if $\ell = 0$ or $k^2 + \ell^2 \neq 1$.

Remarks.

(1) CK(k, ℓ) may underflow, yielding the value 0, when $|k|$ is sufficiently large.
(2) CK and the subroutine CKE employ the same algorithm for $K(k)$.

Precision. If k is real and $|k| < 1$ then the relative error of CK is less than 10^{-13}. Also, if k is purely imaginary then the relative error is less than 10^{-13}. $K(k)$ is real-valued for only these values of k. Otherwise, let $\varepsilon_k = 10^{-12}$ if $|k| < 0.8, \varepsilon_k = 2 \cdot 10^{-13}$ if $0.8 \le |k| < 2$, and $\varepsilon_k = 10^{-13}$ if $|k| \ge 2$. Then the relative errors of the real and imaginary parts of CK are less than ε_k except when underflow occurs, $|k| < 1$ and $|\arg(\pm k)| < 10^{-287}$, or $|k| < 10^{15}$ and $|\pi/2 - \arg(\pm k)| < 10^{-280}$. In the latter two cases the relative error of the real part of CK is less than ε_k, but all relative accuracy for the imaginary part may be lost.

Programming. CK calls the subroutine KL and functions ALNREL, CFLECT, KM, and SPMPAR. CK, KL, and KM were written by Andrew H. Van Tuyl and modified by A. H. Morris.

CALL CKE$(k, \ell, K, E, \text{IERR})$

The arguments k and ℓ are complex numbers where $k^2 + \ell^2 = 1$ and $\ell \neq 0$, K and E are complex variables, and IERR an integer variable. When CKE is called, if no errors are detected then IERR is set to 0, K is assigned the value $K(k)$, and E is assigned the value $E(k)$.

Error Return. IERR = 1 if $\ell = 0$ and IERR = 2 if $k^2 + \ell^2 \neq 1$. In these cases, K and E are not defined.

Algorithm. For $k = 0$ or $-\pi/2 < \arg(k) \le \pi/2$, formulae (2) are used if $|\ell| \le .55$, (1) are used if $|\ell| > .55$ and $|k| \le .55$, and approximations of the form

$$K(k) = \ell \sum_{n=1}^{N} \frac{a_n}{\ell^2 + b_n^2 k^2}$$

(3)

$$E(k) = \ell \sum_{n=1}^{N} a_n \left[1 + \frac{b_n k}{\ell} \tan^{-1} \frac{b_n k}{\ell} \right]$$

are used if $|\ell| > .55$ and $.55 < |k| < 1$. (3) are obtained from integral representations for $K(k)$ and $E(k)$ by numerical quadrature. If $|\ell| > .55, |k| > 1$, and $|k| < |\ell|$ then

$$K(k) = \ell_1 K(k_1) \qquad k_1 = \pm ik/\ell$$

(4)

$$E(k) = E(k_1)/\ell_1 \qquad \ell_1 = 1/\ell$$

are applied where the sign in k_1 is selected so that $-\pi/2 < \arg(k_1) \le \pi/2$. Otherwise, if $|\ell| < .55$, $|k| > 1$, and $|k| < |\ell|$ let $k_1 = 1/k$ and $\ell_1 = \pm i\ell/k$ where the sign is selected so that $-\pi/2 < \arg(\ell_1) \le \pi/2$. Then

$$K(k) = k_1 [K(k_1) \pm iK(\ell_1)]$$

(5)

$$E(k) = \frac{1}{k_1} [E(k_1) - \ell_1^2 K(k_1) \pm i(E(\ell_1) - k_1^2 K(\ell_1))]$$

are applied where $s = 1$ if $\mathrm{Im}(k) \geq 0$ and $s = -1$ if $\mathrm{Im}(k) < 0$. If $\arg(k) > \pi/2$ or $\arg(k) \leq -\pi/2$, then $K(k) = K(-k)$ and $E(k) = E(-k)$ are applied.

Precision. If k is real and $|k| < 1$, or k is purely imaginary, then the relative error of E is less than 10^{-13}. $E(k)$ is real-valued for only these values of k. Otherwise, let $\varepsilon_k = 10^{-12}$ if $|k| < 2$ and $\varepsilon_k = 10^{-13}$ if $|k| \geq 2$. Then the relative errors of the real and imaginary parts of E are less than ε_k except when underflow occurs, $|k| < 1$ and $|\arg(\pm k)| < 10^{-280}$, or $|k| < 10^{15}$ and $|\pi/2 - \arg(\pm k)| < 10^{-280}$. In the latter two cases the relative error of the real part of E is less than ε_k, but all relative accuracy for the imaginary part may be lost.

Programming. CKE calls the subroutines EKL and EKM, and the functions ALNREL, ATN, CF1ECT, and SPMPAR. CKE was written by Andrew H. Van Tuyl.

REAL ELLIPTIC INTEGRALS OF THE FIRST AND SECOND KINDS

If $0 \le \phi \le \pi/2$, then the elliptic integrals of the first and second kinds are defined by

$$F(\phi,k) = \int_0^{\phi} (1 - k^2 \sin^2 t)^{-1/2}\, dt$$

$$E(\phi,k) = \int_0^{\phi} (1 - k^2 \sin^2 t)^{1/2}\, dt$$

for any real k where $k^2 \le 1$ and $1 - k^2 \sin^2\phi \ne 0$. Alternatively, we may consider

$$R_F(a,b,c) = \frac{1}{2}\int_0^{\infty} [(t+a)(t+b)(t+c)]^{-1/2}\, dt$$

where a, b, c are nonnegative and at most one of them is 0, and

$$R_D(a,b,c) = \frac{3}{2}\int_0^{\infty} (t+a)^{-1/2}(t+b)^{-1/2}(t+c)^{-3/2}\, dt$$

where a and b are nonnegative such that $a + b > 0$, and c is positive. If $a \le b \le c$ and $a < c$ then

$$R_F(a,b,c) = \frac{c^{-1/2}}{\sin\phi} F(\phi,k)$$

$$R_D(a,b,c) = \frac{3c^{-3/2}}{k^2 \sin^3\phi}\left[F(\phi,k) - E(\phi,k)\right]$$

where $\cos^2\phi = a/c$ and $k^2 = (c-b)/(c-a)$. If $\phi = \pi/2$ then the integrals $F(\phi,k)$ and $E(\phi,k)$ are said to be ***complete***. Otherwise, if $\phi < \pi/2$ then the integrals are said to be ***incomplete***. The subroutines ELLPI, RFVAL, RDVAL, DELLPI, DRFVAL, and DRDVAL are available for computing $F(\phi,k)$, $E(\phi,k)$, $R_F(a,b,c)$ and $R_D(a,b,c)$. DELLPI, DRFVAL, and DRDVAL are double precision routines.

CALL ELLPI$(\phi, \psi, k, \ell, F, E, \text{IERR})$

The arguments ϕ, ψ, k, ℓ are real numbers which satisfy $\phi \ge 0$, $\psi \ge 0$, $\phi + \psi \le \pi/2$, and $k^2 + \ell^2 \le 1$. Also, if $\psi = 0$ then it is assumed that $\ell \ne 0$. F, E, and IERR are variables. When ELLPI is called, if no input errors are detected then IERR is set to 0, F is assigned the value $F(\phi,k)$, and E is assigned the value $E(\phi,k)$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1 $\phi < 0$ or $\psi < 0$
IERR = 2 $|k| > 1$ or $|\ell| > 1$
IERR = 3 $\psi = 0$ and $\ell = 0$

Precision. ELLPI is accurate to within 4 units of the 14^{th} significant digit.

Programming. ELLPI calls the functions ALNREL and CPABS. ELLPI was written by Allen V. Hershey and modified by A. H. Morris.

Reference. DiDonato, A. R. and Hershey, A. V., "New Formulas for Computing Incomplete Elliptic Integrals of the First and Second Kind," ***JACM*** 6 (1959), pp. 515-526.

CALL RFVAL$(a, b, c, w, \text{IERR})$

The arguments a, b, c are nonnegative real numbers, only one of which can be 0. IERR and w are variables. When RFVAL is called, if no input errors are detected then IERR is set to 0 and w is assigned the value $R_F(a, b, c)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, or c is negative.
IERR = 2 Either $a + b, a + c$, or $b + c$ is too small.
IERR = 3 Either a, b, or c is too large.

Precision. RFVAL is accurate to within 4 units of the 14^{th} significant digit.

Programming. RFVAL was written by B. C. Carlson and Elaine M. Notis (Iowa State University), and modified by A. H. Morris. The function SPMPAR is used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication," ***Numerische Mathematik*** **33** (1979), pp. 1–16.
(2) _____ and Notis, E. M., "Algorithm 577. Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software*** **7** (1981), pp. 398-403.

CALL RDVAL$(a, b, c, w, \text{IERR})$

The arguments a and b are nonnegative real numbers where $a + b > 0$, and c is a positive real number. IERR and w are variables. When RDVAL is called, if no input errors are detected then IERR is set to 0 and w is assigned the value $R_D(a, b, c)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, or c is negative.
IERR = 2 Either $a + b$ or c is too small.
IERR = 3 Either a, b, or c is too large.

Precison. RDVAL is accurate to within 4 units of the 14^{th} significant digit.

Programming. RDVAL was written by B. C. Carlson and Elaine M. Notis (Iowa State University), and modified by A. H. Morris. The function SPMPAR is used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication," ***Numerische Mathematik* 33** (1979), pp. 1–16.

(2) _____ and Notis, E. M., "Algorithm 577, Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software* 7** (1981), pp. 398–403.

CALL DELLPI$(\phi, \psi, k, \ell, F, E,$ IERR)

The arguments ϕ, ψ, k, ℓ are double precision numbers where $\phi \geq 0, \psi \geq 0, \phi + \psi = \pi/2$, and $k^2 + \ell^2 = 1$. Also, if $\psi = 0$ then it is assumed that $\ell \neq 0$. F and E are double precision variables, and IERR is an integer variable. When DELLPI is called, if no input errors are detected then IERR is set to 0, F is assigned the double precision value for $F(\phi, k)$, and E is assigned the double precision value for $E(\phi, k)$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1 $\phi < 0$ or $\psi < 0$
IERR = 2 $|k| > 1$ or $|\ell| > 1$
IERR = 3 $\psi = 0$ and $\ell = 0$

Precision. DELLPI is accurate to within 5 units of the 28^{th} significant digit.

Programming. DELLPI employs the functions DCPABS, DLNREL, and DPMPAR. DELLPI was written by Allen V. Hershey and modified by A. H. Morris.

Reference. DiDonato, A. R. and Hershey, A. V.,"New Formulas for Computing Incomplete Elliptic Integrals of the First and Second Kind," ***JACM* 6** (1959), pp. 515–526.

CALL DRFVAL$(a, b, c, w,$ IERR)

The arguments a, b, c are nonnegative double precision numbers, only one of which can be 0. IERR is an integer variable and w a double precision variable. When DRFVAL is called, if no input errors are detected then IERR is set to 0 and w is assigned the double precision value for $R_F(a, b, c)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, or c is negative.
IERR = 2 Either $a + b, a + c$, or $b + c$ is too small.
IERR = 3 Either a, b, or c is too large.

Programming. DRFVAL was written by B. C. Carlson and Elaine M. Notis (Iowa State University), and modified by A. H. Morris. The function DPMPAR is used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication," ***Numerische Mathematik* 33** (1979), pp. 1–16.

(2) _____ and Notis, E. M., "Algorithm 577, Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software*** **7** (1981), pp. 398–403.

CALL DRDVAL$(a, b, c, w, \text{IERR})$

The arguments a and b are nonnegative double precision numbers where $a + b > 0$, and c is a positive double precision number. IERR is an integer variable and w a double precision variable. When DRDVAL is called, if no input errors are detected then IERR is set to 0 and w is assigned the double precision value for $R_D(a, b, c)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, or c is negative.
IERR = 2 Either $a + b$ or c is too small.
IERR = 3 Either a, b, or c is too large.

Programming. DRDVAL was written by B. C. Carlson and Elaine M. Notis (Iowa State University), and modified by A. H. Morris. The function DPMPAR is used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication," ***Numerische Mathematik*** **33** (1979), pp. 1–16.
(2) _____ and Notis, E. M., "Algorithm 577, Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software*** **7** (1981), pp. 398–403.

REAL ELLIPTIC INTEGRALS OF THE THIRD KIND

For any $0 \leq \phi \leq \pi/2$ the elliptic integral $\Pi(\phi, n, k)$ is defined by

$$\Pi(\phi, n, k) = \int_0^{\phi} (1 - n\sin^2\theta)^{-1}(1 - k^2\sin^2\theta)^{-\frac{1}{2}} d\theta$$

where n is any real number such that $1 - n\sin^2\phi \neq 0$, and k any real number such that $k^2 \leq 1$ and $1 - k^2\sin^2\phi \neq 0$. Alternatively, for any $r \neq 0$ we may consider

$$R_J(a, b, c, r) = \frac{3}{2}\int_0^{\infty} (t + r)^{-1}[(t + a)(t + b)(t + c)]^{-\frac{1}{3}} dt$$

where a, b, c are nonnegative and at most one of them is 0. If $a \leq b \leq c$ and $a < c$ then

$$R_J(a, b, c, r) = \frac{3c^{-\frac{3}{2}}}{n\sin^3\phi}[\Pi(\phi, n, k) - F(\phi, k)]$$

where $F(\phi, k)$ is the elliptic integral of the first kind, $\cos^2\phi = a/c, k^2 = (c - b)/(c - a)$, and $n = (c - r)/(c - a)$. If $\phi = \pi/2$ then the elliptic integral $\Pi(\phi, n, k)$ is said to be ***complete***. Otherwise, if $\phi < \pi/2$ then the integral is said to be ***incomplete***. The subroutines EPI, RJVAL, DEPI, and DRJVAL are available for computing $\Pi(\phi, n, k)$ and $R_J(a, b, c, r)$. DEPI and DRJVAL are double precision routines.

CALL EPI$(\phi, \psi, k^2, \ell^2, n, m, w,$IERR$)$

The arguments $\phi, \psi, k^2, \ell^2, n, m$ are real numbers where $\phi \geq 0, \psi \geq 0, \phi + \psi = \pi/2$, $k^2 + \ell^2 = 1$, $|n| \leq 1$, and $n + m = 1$. Also, if $\psi = 0$ then it is assumed that $\ell^2 \neq 0$ and $m \neq 0$. IERR and w are variables. When EPI is called, if no input errors are detected then IERR is set to 0 and w is assigned the value $\Pi(\phi, n, k)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either ϕ or ψ is negative, or $\phi + \psi \neq \pi/2$.
IERR = 2 Either $|n| > 1$ or $n + m \neq 1$.
IERR = 3 Either k^2 or ℓ^2 is negative, or $k^2 + \ell^2 \neq 1$.
IERR = 4 Either ψ and m are too close to 0, or ψ and ℓ^2 are too close to 0.

Precision. EPI is accurate to within 4 units of the 14^{th} significant digit.

Programming. EPI employs the subroutines RFVAL, RJVAL, RCVALI and function SPMPAR. EPI was written by A.H. Morris.

CALL RJVAL$(a, b, c, r, w,$IERR$)$

The arguments a, b, c are nonnegative real numbers, only one of which can be 0, and r is a positive real number. IERR and w are variables. When RJVAL is called, if no input

errors are detected then IERR is set to 0 and w is assigned the value $R_J(a,b,c,r)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, c, or r is negative.
IERR = 2 Either $a+b, a+c, b+c$, or r is too small.
IERR = 3 Either a, b, c, or r is too large.

Precision. RJVAL is accurate to within 4 units of the 14^{th} significant digit.

Programming. RJVAL calls the subroutine RCVAL1. These subroutines were written by B.C. Carlson and Elaine M. Notis (Iowa State University), and modified by A.H. Morris. The function SPMPAR is also used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication." ***Numerische Mathematik* 33** (1979), pp. 1--16.
(2) _____ and Notis, E. M., "Algorithm 577,Algorithms for Incomplete Elliptic Integrals." ***ACM Trans. Math Software* 7** (1981), pp. 398–403.

CALL DEPI$(\phi, \psi, k^2, \ell^2, n, m, w,$IERR$)$

The arguments $\phi, \psi, k^2, \ell^2, n, m$ are double precision numbers where $\phi \geq 0, \psi \geq 0, \phi + \psi = \pi/2, k^2 + \ell^2 = 1, |n| \leq 1$, and $n + m = 1$. Also, if $\psi = 0$ then it is assumed that $\ell^2 \neq 0$ and $m \neq 0$. IERR is an integer variable and w a double precision variable. When DEPI is called, if no input errors are detected then IERR is set to 0 and w is assigned the double precision value for $\Pi(\phi, n, k)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either ϕ or ψ is negative, or $\phi + \psi \neq \pi/2$.
IERR = 2 Either $|n| > 1$ or $n + m \neq 1$.
IERR = 3 Either k^2 or ℓ^2 is negative, or $k^2 + \ell^2 \neq 1$.
IERR = 4 Either ψ and m are too close to 0, or ψ and ℓ^2 are too close to 0.

Programming. DEPI employs the subroutines DRFVAL, DRJVAL, DRCVL1 and function DPMPAR. DEPI was written by A.H. Morris.

CALL DRJVAL$(a, b, c, r, w,$IERR$)$

The arguments a, b, c are nonnegative double precision numbers, only one of which can be 0, and r is a positive double precision number. IERR is an integer variable and w a double precision variable. When DRJVAL is called, if no input errors are detected then IERR is set to 0 and w is assigned the double precision value for $R_J(a,b,c,r)$.

Error Return. If an input error is detected then IERR has one of the following values:

IERR = 1 Either a, b, c, or r is negative.
IERR = 2 Either $a+b, a+c, b+c$, or r is too small.

IERR = 3 Either a, b, c, or r is too large.

Programming. DRJVAL calls the subroutine DRCVL1. These subroutines were written by B.C. Carlson and Elaine M. Notis (Iowa State University), and modified by A.H. Morris. The function DPMPAR is also used.

References.

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication." ***Numerische Mathematik* 33** (1979), pp. 1–16.

(2) _____ and Notis, E. M., "Algorithm 577,Algorithms for Incomplete Elliptic Integrals." ***ACM Trans. Math Software* 7** (1981), pp. 398–403.

JACOBIAN ELLIPTIC FUNCTIONS

For any complex number $k \neq 0, \pm 1$ the elliptic function $sn(z,k)$ may be defined as the meromorphic function $w(z)$ that satisfies

$$\left(\frac{dw}{dz}\right)^2 = (1-w^2)(1-k^2w^2) \tag{1}$$

$$w(0) = 0,\ w'(0) = 1.$$

If $k = 0$ then $sn(z,0) = \sin z$ satisfies (1), and if $k = \pm 1$ then $sn(z,k) = \tanh z$ satisfies (1). Alternatively, $sn(z,k) = \sin\phi$ where $\phi(z)$ satisfies

$$\left(\frac{d\phi}{dz}\right)^2 = 1 - k^2\sin^2\phi \tag{2}$$

$$\phi(0) = 0,\ \phi'(0) = 1.$$

The elliptic functions $cn(z,k)$ and $dn(z,k)$ may be defined by

$$cn(z,k) = \sqrt{1 - sn(z,k)^2}$$
$$dn(z,k) = \sqrt{1 - k^2 sn(z,k)^2}$$

where the roots take the value 1 for $z = 0$. In particular, if $k = 0$ then $cn(z,0) = \cos z$ and $dn(z,0) = 1$, and if $k = \pm 1$ then $cn(z,k) = dn(z,k) = 1/\cosh z$. The subroutines ELLPF and ELPFC1 are available for computing $sn(z,k), cn(z,k)$, and $dn(z,k)$ when k is a real value such that $|k| \leq 1$. ELLPF may be used when z is real and ELPFC1 when z is complex.

CALL ELLPF(u, k, ℓ, S, C, D,IERR)

It is assumed that u, k, and ℓ are real numbers where $k^2 + \ell^2 = 1$. S, C, and D are real variables. When ELLPF is called, S, C, and D are assigned the values $S = sn(u,k)$, $C = cn(u,k)$, and $D = dn(u,k)$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0 The elliptic functions were computed.
IERR = 1 (Input error) $k^2 + \ell^2 \neq 1$.
IERR = 2 u is too large for k.

When IERR ≥ 1, no computation is performed.

Precision. Let $K(k)$ be the complete elliptic integral of the first kind. For $|k| \leq .99995$ the relative errors of $sn(u,k)$ and $dn(u,k)$ are less than 10^{-12} when $0 \leq u \leq K(k)$, and the relative error of $cn(u,k)$ is less than 10^{-12} when $0 \leq u \leq .97K(k)$.

Algorithm. Let $K = K(k)$ be the complete elliptic integral of the first kind. For $0 \le u \le K/2$ (when $\ell \neq 0$), the Maclaurin expansion

$$sn(u,k) = u - \frac{(1+k^2)u^3}{3!} + \frac{(1+14k^2+k^4)u^5}{5!} - \cdots$$

is employed when $u \le .01$. Otherwise, if $u > .01$ let $K' = K(\ell)$, $q = \exp(-\pi K'/K)$, and $r = \exp(-\pi K/K')$. Then

$$sn(u,k) = \frac{2\pi}{kK} \sum_{n \ge 0} \frac{q^{n+\frac{1}{2}}}{1-q^{2n+1}} \sin \frac{(2n+1)\pi u}{2K}$$

is used when $k \le \ell$ and

$$sn(u,k) = \frac{\pi}{2kK'} \left[\tanh \frac{\pi u}{2K'} + 4 \sum_{n \ge 1} \frac{(-1)^n r^{2n}}{1+r^{2n}} \sinh \frac{2\pi u}{K'} \right]$$

is used when $k > \ell$. The functions $cn(u,k)$ and $dn(u,k)$ are obtained from

$$sn(u,k)^2 + cn(u,k)^2 = 1$$
$$dn(u,k)^2 + k^2 sn(u,k)^2 = 1.$$

For $K/2 < u \le K$ the identities

$$sn(u,k) = cn(v,k)/dn(v,k)$$
$$cn(u,k) = |\ell| sn(v,k)/dn(v,k)$$
$$dn(u,k) = |\ell|/dn(v,k)$$

are applied. Here $v = K - u$.

Programming. ELLPF employs the subroutines SCD, SCDF, SCDJ, SCDM, ELLPI, SNHCSH and functions ALNREL, CPABS, SPMPAR, IPMPAR. ELLPF was written by Andrew H. Van Tuyl and modified by A. H. Morris.

CALL ELPFC1(z, k, ℓ, S, C, D,IERR)

The argument z is complex, and k and ℓ are real numbers where $k^2 + \ell^2 = 1$. S, C, and D are complex variables. When ELPFC1 is called S, C, and D are assigned the values $S = sn(z,k)$, $C = cn(z,k)$, and $D = dn(z,k)$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0 The elliptic functions were computed.
IERR = 1 (Input error) $k^2 + \ell^2 \neq 1$.
IERR = 2 z is too large for k.
IERR = 3 z is a pole for the elliptic functions.

When IERR ≥ 1, no computation is performed.

Precision. Let $x = \mathrm{Re}(z)$, $y = \mathrm{Im}(z)$, $K = K(k)$ be the elliptic integral of the first kind for k, and $K' = K(\ell)$. For $|k| < .99995$ the relative errors of the real and imaginary parts of $sn(z,k)$ are less than 10^{-12} when $0 < x \leq K$ and $0 < y \leq .992K'$, and the relative errors of the real and imaginary parts of $cn(z,k)$ and $dn(z,k)$ are less than 10^{-12} when $0 < x \leq .97K$ and $0 < y \leq .97K'$.

Algorithm. For $z = x + iy$ let

$$\begin{array}{ll} s = sn(x,k) & s_1 = sn(y,\ell) \\ c = cn(x,k) & c_1 = cn(y,\ell) \\ d = dn(x,k) & d_1 = dn(y,\ell) \end{array}$$

and $D = c_1^2 + k^2 s^2 s_1^2$. Then

$$\begin{aligned} sn(z,k) &= (sd_1 + icds_1c_1)/D \\ cn(z,k) &= (cc_1 - isds_1d_1)/D \\ dn(z,k) &= (dc_1d_1 - ik^2scs_1)/D \end{aligned}$$

are applied when $D \neq 0$.

Programming. ELPFC1 calls ELLPF, which employs the subroutines SCD, SCDF, SCDJ, SCDM, ELLPI, SNHCSH and functions ALNREL, CPABS, SPMPAR, IPMPAR. ELPFC1 was written by Andrew H. Van Tuyl.

WEIERSTRASS ELLIPTIC FUNCTION FOR THE EQUIANHARMONIC AND LEMNISCATIC CASES

Let w and w' be complex numbers where $\mathrm{Im}(w'/w) > 0$, and $w_{mn} = 2mw + 2nw'$ for all integers m, n. Then for any complex z, the Weierstrass elliptic function $\mathcal{P}(z; w, w')$ can be defined by

$$\mathcal{P}(z; w, w') = \frac{1}{z^2} + \sum{}' \left[\frac{1}{(z - w_{mn})^2} - \frac{1}{w_{mn}^2} \right]$$

where Σ' denotes the sum for all $m, n = 0, \pm 1, \pm 2, \ldots$ except $m = n = 0$. If $w = re^{i\phi}$ and $w' = r'e^{i\phi'}$ where $\phi' = \phi + \theta$ for $0 \leq \theta < 2\pi$, then the restriction $\mathrm{Im}(w'/w) > 0$ is equivalent to assuming that $0 < \theta < \pi$. $\mathcal{P}(z; w, w')$ is analytic everywhere except at the points w_{mn}, which are poles, and

$$\mathcal{P}(z + 2w; w, w') = \mathcal{P}(z; w, w')$$
$$\mathcal{P}(z + 2w'; w, w') = \mathcal{P}(z, w, w')$$

for all z. The relations

$$\mathcal{P}(-z; w, w') = \mathcal{P}(z; w, w')$$
$$\mathcal{P}(\lambda z; \lambda w, \lambda w') = \lambda^{-2}\mathcal{P}(z; w, w') \quad \lambda \neq 0$$

also hold. A somewhat surprising fact is that only the values $g_2 = 60\Sigma' w_{mn}^{-4}$ and $g_3 = 140\Sigma' w_{mn}^{-6}$ are needed for computing $\mathcal{P}(z; w, w')$ at a point z. Hence, $\mathcal{P}(z; w, w')$ is frequently denoted by $\mathcal{P}(z; g_2, g_3)$. For $\lambda \neq 0$

$$g_2(\lambda w, \lambda w') = \lambda^{-4} g_2(w, w')$$
$$g_3(\lambda w, \lambda w') = \lambda^{-6} g_3(w, w')$$

also hold. We now consider the following cases:

(1) Equianharmonic ($g_2 = 0$ and g_3 is a positive real number)

(2) Lemniscatic (g_2 is a positive real number and $g_3 = 0$)

(1) occurs when $2w = \frac{1}{2} - \frac{\sqrt{3}}{2}i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2}i$, and (2) occurs when $2w = 1$ and $2w' = i$. The following subroutines are available for computing $\mathcal{P}(z; w, w')$ and its derivative $\mathcal{P}'(z; w, w')$ for these two choices of (w, w').

CALL PEQ(z, e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = \frac{1}{2} - \frac{\sqrt{3}}{2}i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2}i$. When PEQ is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathcal{P}(z; w, w')$.

Error Return. If $z = w_{mn}$ for some m, n then IERR is assigned the value 1 and $e = 0$.

Precision. If $|\mathcal{P}(z; w, w')| \leq 1$ then the absolute error is less than $7 \cdot 10^{-13}$. Otherwise, the relative error is less than $7 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References.

(1) Eckhardt, Ulrich,"Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software*** **4** (1980), pp. 112–120.

(2) _____,"A Rational Approximation to Weierstrass' $\mathcal{P}$-Function," ***Math Comp.*** **30** (1976), pp. 818–826.

CALL PEQ1(z, e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = \frac{1}{2} - \frac{\sqrt{3}}{2}i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2}i$. When PEQ1 is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathcal{P}'(z; w, w')$.

Error Return. If $z = w_{mn}$ for some m, n then IERR is assigned the value 1 and $e = 0$.

Precision. If $|\mathcal{P}'(z; w, w')| \leq 1$ then the absolute error is less than $7 \cdot 10^{-13}$. Otherwise, the relative error is less than $7 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References.

(1) Eckhardt, Ulrich,"Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software*** **4** (1980), pp. 112–120.

(2) _____,"A Rational Approximation to Weierstrass' $\mathcal{P}$-Function," ***Math Comp.*** **30** (1976), pp. 818–826.

CALL PLEM(z, e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = 1$ and $2w' = i$. When PLEM is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathcal{P}(z; w, w')$.

Error Return. If $z = w_{mn}$ for some m, n then IERR is assigned the value 1 and $e = 0$.

Precision. If $|\mathcal{P}(z; w, w')| \leq 1$ then the absolute error is less than $6 \cdot 10^{-13}$. Otherwise, the relative error is less than $6 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References.

(1) Eckhardt, Ulrich,"Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software*** **4** (1980), pp. 112–120.

(2) _____,"A Rational Approximation to Weierstrass' $\mathcal{P}$-Function. II: The Lemniscatic Case," ***Computing (Arch. Elektron. Rechnen)*** **18** (1977), pp. 341–349.

CALL PLEM1$(z, e,$IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = 1$ and $2w' = i$. When PLEM1 is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathcal{P}'(z; w, w')$.

Error Return. If $z = w_{mn}$ for some m, n then IERR is assigned the value 1 and $e = 0$.

Precision. If $|\mathcal{P}'(z; w, w')| \leq 1$ then the absolute error is less than $6\ 10^{-13}$. Otherwise, the relative error is less than $6 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References.

(1) Eckhardt, Ulrich,"Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software*** **4** (1980), pp. 112–120.

(2) _____,"A Rational Approximation to Weierstrass' $\mathcal{P}$-Function. II: The Lemniscatic Case," ***Computing (Arch. Elektron. Rechnen)*** **18** (1977), pp. 341–349.

INTEGRAL OF THE BIVARIATE DENSITY FUNCTION OVER ARBITRARY POLYGONS AND SEMI-INFINITE ANGULAR REGIONS

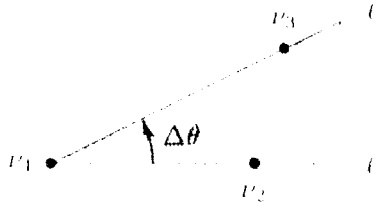
Given a sequence of points $\nu_i = (x_i, y_i)(i = 1, \ldots, n+1)$ where $n \geq 3$ and $\nu_{n+1} = \nu_1$. Let τ denote the polygon whose boundary $\partial\tau$ is a polygonal line which begins at point ν_1, traverses the points ν_i in the order that they are indexed, and is the straight line segment connecting ν_i to ν_{i+1} for each $i = 1, \ldots, n$ where $\nu_i \neq \nu_{i+1}$. Then the subroutine VALR2 is available for computing the integral

$$P(\tau) = \frac{1}{2\pi} \iint_{\tau} e^{-(x^2+y^2)/2}\, dx\, dy$$

and the associated function $A(\tau) = \iint_{\tau} dx\, dy$. If the boundary $\partial\tau$ is a simple positively (negatively) oriented closed curve, then $P(\tau)$ and $A(\tau)$ are positive (negative) and $|A(\tau)|$ = the area of τ. However, $\partial\tau$ need not be simple. It may be self-intersecting or have overlapping line segments. If $\Delta\theta_i$ is the angle between the vectors $\nu_i - \nu_{i-1}$ and $\nu_{i+1} - \nu_i$ (where $\nu_0 = \nu_n$), then it may occur that $\Delta\theta_i = \pi$ for some i, in which case a portion of the polygon may be degenerate. In general, $-\pi < \Delta\theta_i \leq \pi$ for each i where the sign of the angle is positive (negative) if the angle is measured in a counterclockwise (clockwise) direction from $\nu_i - \nu_{i-1}$ to $\nu_{i+1} - \nu_i$. VALR2 also computes the value $k(\tau) = \frac{1}{2\pi} \sum_{i=1}^{n} \Delta\theta_i$, which is an integer. If the boundary is a simple closed curve, then $k(\tau)$ is the winding number of the curve around any interior point of the polygon τ.

Alternatively, assume that we are given three points $\nu_i = (x_i, y_i)(i = 1, 2, 3)$ and let $\Delta\theta$ denote the angle between the vectors $\nu_2 - \nu_1$ and $\nu_3 - \nu_1$. In this case, assume that the angle $\Delta\theta$ is measured in a counterclockwise direction from $\nu_2 - \nu_1$ to $\nu_3 - \nu_1$, so that $0 \leq \Delta\theta < 2\pi$. Let ℓ denote the straight line beginning at point ν_1 and passing through point ν_2, and let $\tilde{\ell}$ denote the straight line beginning at ν_1 and passing through ν_3. Then the subroutine VALR2 is also available for computing $P(\tau)$ when τ is the semi-infinite angular region bounded by ℓ and $\tilde{\ell}$, and having the angle $\Delta\theta$. $0 < P(\tau) < 1$ for any angular region τ, and $P(\tau) \to 1$ when $\Delta\theta \to 2\pi$.

Anguiar region τ



CALL VALR2$(X, Y, n, P, \text{IOP}, A, \text{IND}, k)$

The argument n is either 1 or the number of points involved in defining a polygon. If $n = 1$ then it is assumed that τ is a semi-infinite angular region defined by the points $\nu_i = (x_i, y_i)(i = 1, 2, 3)$, and that X and Y are arrays containing x_1, x_2, x_3 and y_1, y_2, y_3. Otherwise, if $n \neq 1$ then it is assumed that τ is a polygon defined by the points ν_i,

(x_i, y_i) $(i = 1, \ldots, n+1)$ where $n \geq 3$ and $\nu_{n+1} = \nu_1$. In this case, X and Y are arrays containing the abscissae $x_1, \ldots, x_{n+1}$ and ordinates $y_1, \ldots, y_{n+1}$. Since $\nu_{n+1} = \nu_1$, the values x_{n+1} and y_{n+1} need not be supplied by the user. The routine automatically stores x_1 and y_1 in $X(n+1)$ and $Y(n+1)$.

P, A, and k are variables. If $n = 1$ then P is assigned the value $P(\tau)$ for the angular region τ and A is assigned the value 0. In this case, k is not defined. Otherwise, if $n \geq 3$ then P is assigned the value $P(\tau)$, A is assigned the value $A(\tau)$, and k is assigned the value $k(\tau)$ for the polygon τ.

IOP is an input argument which specifies the (relative) precision to which $P(\tau)$ is to be computed. IOP is set to 1, 2, or 3 for 3, 6, or 9 decimal digit accuracy.

IND is the variable that reports the status of the results. The routine assigns IND one of the following values:

IND = 0 The desired values were obtained.

IND = 1 (Input error) ν_1 is either equal to ν_2 or ν_3, or is too close to ν_2 or ν_3 to compute $P(\tau)$ for the angular region τ. In this case, P is set to 5.

IND = 2 The desired values were obtained. If $n = 1$ then $\Delta\theta \approx \pi$. Otherwise, if $n \geq 3$ then $|\Delta\theta_i| \approx \pi$ for some i.

IND = 3 (Input error) Either $n < 1$ or $n = 2$.

Remarks. VALR2 can be used for computing the integral of the general bivariate density function over an arbitrary polygon or semi-infinite angular region $\hat{\tau}$. Consider

$$\hat{P}(\hat{\tau}) = B \iint_{\hat{\tau}} exp\left\{ \frac{-1}{2(1-\rho^2)} \left[\left(\frac{\omega - \mu_\omega}{\sigma_\omega} \right)^2 - 2\rho \frac{(\omega - \mu_\omega)(z - \mu_z)}{\sigma_\omega \sigma_z} + \left(\frac{z - \mu_z}{\sigma_z} \right)^2 \right] \right\} d\omega\, dz$$

where $B = (1-\rho^2)^{-1/2}/(2\pi\sigma_\omega\sigma_z)$, (μ_ω, μ_z) is the mean, σ_ω and σ_z are the (nonzero) variances, and ρ is the correlation coefficient satisfying $|\rho| < 1$. Consider also

$$x = (1-\rho^2)^{-\frac{1}{2}} \left[\frac{\omega - \mu_\omega}{\sigma_\omega} - \rho \frac{z - \mu_z}{\sigma_z} \right] \quad \text{and} \quad y = \frac{z - \mu_z}{\sigma_z}.$$

Since this transformation maps straight lines into straight lines, $\hat{\tau}$ is mapped onto a polygon or angular region τ and we obtain $\hat{P}(\hat{\tau}) = P(\tau)$. Moreover, if $\hat{\tau}$ is a polygon then $A(\hat{\tau}) = \iint_{\hat{\tau}} d\omega dz = \sigma_\omega \sigma_z \sqrt{1-\rho^2}\, A(\tau)$.

Programming. VALR2 employs the functions ERF, ERFC1, and SPMPAR. VALR2 was designed by Armido R. DiDonato and Richard K. Hageman, and modified by A. H. Morris.

Reference. DiDonato, A. R., and Hageman, R. K., ***Computation of the Integral of the Bivariate Normal Distribution over Arbitrary Polygons***, Report TR 80-166, Naval Surface Weapons Center, Dahlgren, Virginia, 1980.

CIRCULAR COVERAGE FUNCTION

The subroutine CIRCV is available for computing the circular coverage function $P(R,d)$ and the generalized circular error function $V(K,c)$. V is the integral of an uncorrelated elliptical Gaussian distribution with standard deviations σ_x and σ_y over a circle of radius $K\sigma_x$ centered at the mean of the distribution. If $\sigma_x \geq \sigma_y$ then

$$V(K,c) = \frac{1}{\pi c}\int_0^K\int_0^\pi \exp\left\{\frac{-r^2}{4c^2}\left[1+c^2+(1-c^2)\cos\theta\right]\right\} r\,d\theta\,dr$$

where $c = \sigma_y/\sigma_x$. P is the integral of a circular Gaussian distribution with common standard deviation σ over a circle of radius $R\sigma$ whose center is offset a distance $d\sigma$ from the mean of the distribution.

$$P(R,d) = \frac{1}{2\pi}\int_0^R\int_0^{2\pi} \exp\left\{-\frac{1}{2}\left[(d+r\cos\theta)^2+r^2\sin^2\theta\right]\right\} r\,d\theta\,dr$$

Alternatively,

$$V(K,c) = \frac{1}{c}\int_0^K e^{-Br^2} I_0(Ar^2) r\,dr \qquad (c \neq 0)$$

$$P(R,d) = e^{-d^2/2}\int_0^R e^{-r^2/2} I_0(rd) r\,dr$$

where $A = (1-c^2)/(4c^2)$, $B = (1+c^2)/(4c^2)$, and I_0 is the modified Bessel function of the first kind and order 0. Since $V(K,c) \to \mathrm{erf}(K/\sqrt{2})$ when $c \to 0$, we may define $V(K,0) = \mathrm{erf}(K/\sqrt{2})$.

CALL CIRCV(x,a,i,w,IERR)

The argument i may be any integer. If $i = 0$ then the arguments x and a are assumed to have the values $x = K$ and $a = c$ where $K > 0$ and $0 \leq c \leq 1$. Otherwise, if $i \neq 0$ then $x = R$ and $a = d$ where $R > 0$ and $d \geq 0$.

IERR and w are variables. When CIRCV is called, if no input errors are detected then IERR is assigned the value 0. Also, $w = V(K,c)$ if $i = 0$ and $w = P(R,d)$ if $i \neq 0$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1 if $x > 0$ is not satisfied.

IERR = 2 if $0 \leq c \leq 1$ or $d \geq 0$ is not satisfied.

When either of these errors is detected, w is assigned the value -1.

Precision If $i = 0$ then w is accurate to within 1 unit of the 13^{th} significant digit when $V(K,c)$ is not near 0. Otherwise, if $i \neq 0$ then w is accurate to within 1 unit of the 11^{th} significant digit when $P(R,d)$ is not near 0.

Algorithm. If $AK^2 \leq 14$ then $V(K,c) = \sum_{n\geq 0} T_n$ is employed where

$$S_0 = \frac{1}{2cB} e^{-BK^2} \qquad T_0 = \frac{1}{2cB}\left[1 - e^{-BK^2}\right]$$
$$S_n = \left(\frac{AK^2}{2n}\right) S_{n-1} \qquad T_n = \frac{2n-1}{2n}\frac{A^2}{B^2} T_{n-1} - \left(\frac{AK^2}{2n} + \frac{A}{B}\right)\frac{AK^2}{2n} S_{n-1} \qquad (n > 1).$$

Otherwise, if $AK^2 > 14$ then the asymptotic expansion $V(K,c) = 1 - e^{-K^2/2}\sum_{n\geq 1} M_n$ is used where

$$M_1 = \frac{1}{\delta} e^{K^2/2}\mathrm{erfc}\left(\frac{K}{\sqrt{2}}\right) \qquad x_1 = \frac{1}{\sqrt{2\pi}}\frac{2}{K\delta} \qquad \delta = \sqrt{1-c^2}$$
$$M_{n+1} = \frac{2n-1}{2n}\frac{c^2}{1-c^2}(x_n - M_n) \qquad x_{n+1} = \frac{2n-1}{2n}\frac{2n-1}{4AK^2} x_n \qquad (n \geq 1).$$

If $R < 1.7$ or $Rd \leq 16$ then $P(R,d) = \sum_{n\geq 0} T_n$ is employed where

$$T_n = e^{-\delta}\frac{\delta^n}{n!}\left[1 - e^{-\gamma}\sum_{i=0}^{n}\frac{\gamma^i}{i!}\right] \qquad \delta = d^2/2 \qquad \gamma = R^2/2.$$

Otherwise, if $Rd > 16$ and $R > 1.7$ then the asymptotic expansion

$$P(R,d) = \begin{cases} \frac{1}{2}e^{-z^2}|s_1 - s_2| & \text{if } R < d \\ |1 - \frac{1}{2}e^{-z^2}(s_1 + s_2)| & \text{if } R > d \end{cases}$$

is used where $z = |R-d|/\sqrt{2}$, $s_1 = \frac{R+d}{2\sqrt{Rd}}\sum_{n\geq 1} M_n$, $s_2 = \frac{1}{\sqrt{2Rd}}\sum_{n\geq 1} x_n$, and

$$x_1 = \frac{1}{\sqrt{\pi}} \qquad M_1 = e^{z^2}\mathrm{erfc}(z)$$
$$x_{n+1} = \frac{2n-1}{2n}\frac{2n-1}{4Rd} x_n \qquad M_{n+1} = \frac{2n-1}{2n}\frac{z}{2Rd}(x_n - zM_n) \qquad (n \geq 1).$$

Programming. CIRCV employs the functions ERF, ERFC1, ERFCR, EXPARG, GAMMA, GAM1, GLOG, RCOMP, REXP, RLOG and subroutines ERFC0, GRATIO. The functions SPMPAR and IPMPAR are also used. CIRCV was written by Armido R. DiDonato and modified by A. H. Morris.

References.

(1) DiDonato, A. R., ***Five Statistical Programs in BASIC for Desktop Computers***, Report NSWC TR 83-13, Naval Surface Weapons Center, Dahlgren, Virginia, 1982.

(2) and Jarnagin, M. P., ***A Method for Computing the Generalized Circular Error Function and the Circular Coverage Function***, Report 1768, Naval Weapons Laboratory, Dahlgren, Virginia, 1962.

ELLIPTICAL COVERAGE FUNCTION

The subroutine PKILL is available for evaluating the integral of an uncorrelated elliptical Gaussian distribution over the area A of a circle $(x-h)^2+(y-k)^2=R^2$. The probability to be computed is given by

$$P(R,\sigma_x,\sigma_y,h,k)=\frac{1}{2\pi\sigma_x\sigma_y}\iint_A \exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)\right]dx\,dy$$

where σ_x is the standard deviation in the x direction and σ_y is the standard deviation in the y direction. Then

$$\begin{aligned}
P(R,\sigma_x,\sigma_y,h,k) &= P(R,\sigma_y,\sigma_x,k,h),\\
P(R,\sigma_x,\sigma_y,h,k) &= P(R,\sigma_x,\sigma_y,|h|,|k|),\ \text{and}\\
P(R,\sigma_x,\sigma_y,h,k) &= P(aR,a\sigma_x,a\sigma_y,ah,ak)
\end{aligned}$$

for any $a>0$. Also $P=0$ when $R=0$.

CALL PKILL$(R,\sigma_x,\sigma_y,h,k,P)$

R,σ_x,σ_y,h,k are real numbers and P is a variable. It is assumed that $R\geq 0$, $\sigma_x>0$, and $\sigma_y>0$. When PKILL is called, P is assigned the value $P(R,\sigma_x,\sigma_y,h,k)$.

Error Return. $P=-10^{10}$ if $\sigma_x\leq 0$ or $\sigma_y\leq 0$.

Precision. P is accurate to within 1 unit of the 6^{th} significant digit when $P\geq 10^{-20}$ and $|h|,|k|,\sqrt{h^2+k^2}$ are not near R.

Programming. PKILL employs the subroutine GRATIO and functions SPMPAR, ERF, IPMPAR, EPSLN, EXPARG, REXP, RLOG, ERFC, ERFC1, AERF, GAMMA, GAM1, GLOG. PKILL was written by Armido R. DiDonato.

Reference. DiDonato, A. R., ***Significant Digit Computation of the Elliptical Coverage Function***, Report NSWC TR 90 513, Naval Surface Warfare Center, Dahlgren, Virginia, 1990.

COPYING POLYNOMIALS

If $p(x) = \sum_{j=0}^{m-1} a_j x^j$ and the coefficients a_j are stored in an array A, then the following subroutines are available for copying the first n coefficients a_j into an array B.

CALL PLCOPY(A, ka, m, B, kb, n)
CALL DPCOPY(A, ka, m, B, kb, n)

A and B are arrays. PLCOPY is used if A and B are real arrays, and DPCOPY is used if A and B are double precision arrays.

The arguments m, n, ka, kb are positive integers. The coefficients a_j are assumed to be stored in A where $A(1 + j \cdot ka) = a_j$ for $j = 0, 1, \ldots, m - 1$. The routine stores the first n coefficients a_j in B where $B(1 + j \cdot kb) = a_j$ for $j = 0, 1, \ldots, n - 1$.

Note. If $n > m$ then $B(1 + j \cdot kb) = 0$ for $j \geq m$.

Programmer. A. H. Morris.

ADDITION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x) + q(x) = \sum_j c_j x^j$.

CALL PADD$(A, ka, \ell, B, kb, m, C, kc, n)$
CALL DPADD$(A, ka, \ell, B, kb, m, C, kc, n)$

A, B and C are arrays. PADD is used if A, B and C are real arrays and DPADD is used if A, B and C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j \cdot ka) = a_j \quad (j = 0, 1, \ldots, \ell - 1)$
$B(1 + j \cdot kb) = b_j \quad (j = 0, 1, \ldots, m - 1).$

The routine stores the first n coefficients c_j of $p(x) + q(x)$ in C where $C(1 + j \cdot kc) = c_j$ for $j = 0, 1, \ldots, n - 1$.

Remarks. The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris.

SUBTRACTION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x) - q(x) = \sum_j c_j x^j$.

CALL PSUBT($A, ka, \ell, B, kb, m, C, kc, n$)
CALL DPSUBT($A, ka, \ell, B, kb, m, C, kc, n$)

A, B and C are arr PSUBT is used if A, B and C are real arrays and DPSUBT is used if A, B and C are ble precision arrays.

The arguments ℓ, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored nd B where

$A(1 + j \cdot ka)$ $(j = 0, 1, \ldots, \ell - 1)$
$B(1 + j \cdot kb)$ $(j = 0, 1, \ldots, m - 1)$.

The routine stores the s n coefficients c_j of $p(x) - q(x)$ in C where $C(1 + j \cdot kc) = c_j$ for $j = 0, 1, \ldots, n - 1$.

Remarks. The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris.

MULTIPLICATION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x)q(x) = \sum_j c_j x^j$.

CALL PMULT$(A, ka, \ell, B, kb, m, C, kc, n)$
CALL DPMULT$(A, ka, \ell, B, kb, m, C, kc, n)$

A, B and C are arrays. PMULT is used if A, B and C are real arrays and DPMULT is used if A, B and C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j \cdot ka) = a_j \quad (j = 0, 1, \ldots, \ell - 1)$
$B(1 + j \cdot kb) = b_j \quad (j = 0, 1, \ldots, m - 1).$

The routine stores the first n coefficients c_j of $p(x)q(x)$ in C where $C(1 + j \cdot kc) = c_j$ for $j = 0, 1, \ldots, n - 1$.

Remarks. It is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris.

DIVISION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ where $b_0 \neq 0$, then the following subroutines are available for computing the first n coefficients of the series $p(x)/q(x) = \sum_j c_j x^j$.

CALL PDIV$(A, ka, \ell, B, kb, m, C, kc, n,$IERR$)$
CALL DPDIV$(A, ka, \ell, B, kb, m, C, kc, n,$IERR$)$

A, B and C are arrays. PDIV is used if A, B and C are real arrays and DPDIV is used if A, B and C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j \cdot ka) = a_j \quad (j = 0, 1, \ldots, \ell - 1)$
$B(1 + j \cdot kb) = b_j \quad (j = 0, 1, \ldots, m - 1).$

IERR is a variable. When the routine is called, if $b_0 \neq 0$ then IERR is assigned the value 0 and the first n coefficients c_j of $p(x)/q(x)$ are stored in C where $C(1 + j \cdot kc) = c_j$ for $j = 0, 1, \ldots, n - 1$.

Error Return. IERR = 1 if $b_0 = 0$. In this case, no computation is performed.

Remarks. It is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris.

REAL POWERS OF POLYNOMIALS

If r is real and $p(x) = \sum_{j=0}^{m-1} a_j x^j$ where $a_0 > 0$, then the following subroutines are available for computing the first n coefficients of the series $p(x)^r = \sum_j b_j x^j$.

CALL PLPWR$(r, A, ka, m, B, kb, n,$IERR$)$
CALL DPLPWR$(r, A, ka, m, B, kb, n,$IERR$)$

A and B are arrays. PLPWR is used if A and B are real arrays and r a real number, and DPLPWR is used if A and B are double precision arrays and r a double precision number.

The arguments m, n, ka, kb are positive integers. The coefficients a_j are assumed to be stored in A where $A(1 + j \cdot ka) = a_j$ for $j = 0, 1, \ldots, m-1$. IERR is a variable. When the routine is called, if $a_0 > 0$ then IERR is assigned the value 0 and the first n coefficients b_j of $p(x)^r$ are stored in B where $B(1 + j \cdot kb) = b_j$ for $j = 0, 1, \ldots, n-1$.

Error Return. IERR = 1 if $a_0 \leq 0$. In this case, no computation is performed.

Remark. It is assumed that the arrays A and B do not overlap.

Algorithm. If $q = p^r$ then $pq' = rqp'$ where p' and q' are the derivatives of p and q. Consequently, $b_j = \frac{1}{j a_0} \sum_{i=1}^{j} (ri + i - j) a_i b_{j-i}$ is used for $j \geq 1$. Also $b_0 = a_0^r$.

Programmer. A. H. Morris.

INVERSES OF POWER SERIES

Given an analytic function $w = f(z) = \sum_{i \geq 1} a_i z^i$ where $f(0) = 0$. Then the inverse function $z = f^{-1}(w)$ exists when $a_1 \neq 0$, and $f^{-1}(w) = \sum_{i \geq 1} d_i w^i$. The subroutines PINV and DPINV are available for obtaining the coefficients d_i when the coefficients a_i are real. DPINV is a double precision routine.

CALL PINV(A, D, n,WK)
CALL DPINV(A, D, n,WK)

If PINV is called then A, D, and WK are real arrays. Otherwise, if DPINV is called then A, D, and WK are double precision arrays.

It is assumed that $n \geq 2$. A is an array containing the coefficients $a_1, \ldots, a_n$ and D is an array of dimension n. When PINV or DPINV is called, the coefficients $d_1, \ldots, d_n$ are computed and stored in D.

WK is an array of dimension $n(n+1)/2$ or larger. WK is a work space for the routine.

Programmer. A. H. Morris.

Reference. Chang, Feng-cheng,"Power Series Unification and Reversion," ***Applied Math and Computation*** **23** (1987), pp. 7–23.

DERIVATIVES AND INTEGRALS OF POLYNOMIALS

Let $f(x) = \sum_{i=0}^{n-1} a_i x^i$ be a polynomial with real coefficients a_i. The polynomial can be differentiated and integrated by the following subroutine:

CALL MPLNMV(MO,x_0, n, A, w)

A is an array containing the coefficients a_i where $A(i) = a_{i-1}$ for $i = 1, \ldots, n$. The argument x_0 is an arbitrary real number and w is a variable. MO may have the values $-1, 0, 1, 2$. When MPLNMV is called w is assigned the value:

$$w = \begin{cases} \int_0^{x_0} f(x)\,dx & \text{if MO} = -1 \\ f(x_0) & \text{if MO} = 0 \\ f'(x_0) & \text{if MO} = 1 \\ f''(x_0) & \text{if MO} = 2 \end{cases}$$

Programmer. Allen V. Hershey.

EVALUATION OF CHEBYSHEV EXPANSIONS

For any complex number z and integer $n = 0, 1, \ldots$ let

$$T_0(z) = 1, \quad T_1(z) = z$$

$$T_{n+2}(z) = 2zT_{n+1}(z) - T_n(z).$$

Then $T_n(z)$ is a polynomial of degree n having the leading coefficient 2^{n-1} when $n \geq 1$. Also $T_n(t) = \cos(n\theta)$ when $t = \cos\theta (0 \leq \theta \leq \pi)$, so that $|T_n(t)| \leq 1$ for real t where $|t| \leq 1$. The polynomials $T_n(z)$ are called the *Chebyshev polynomials (of the first kind)*. If $f(x) = a_0/2 + \sum_{i=1}^{n-1} a_i T_i(x)$ where a_i is real, then the following functions are available for computing $f(x)$ when x is real.

CSEVL(x, A, n)

It is assumed that $n \geq 1$ and that A is an array containing $a_0, a_1, \ldots, a_{n-1}$ where $A(i) = a_{i-1} (i = 1, \ldots, n)$. Then CSEVL$(x, A, n) = f(x)$ for any real x.

Programmer. A. H. Morris.

DCSEVL(x, A, n)

It is assumed that $n \geq 1$ and A is a double precision array containing $a_0, a_1, \ldots, a_{n-1}$ where $A(i) = a_{i-1} (i = 1, \ldots, n)$. Then for any double precision value x, DCSEVL(x, A, n) is the double precision value of $f(x)$.

Remark. DCSEVL must be declared in the calling program to be of type DOUBLE PRECISION.

Programmer. A. H. Morris.

LAGRANGE POLYNOMIALS

Let $a_1, \ldots, a_n$ be n distinct real numbers. Then the i^{th} ***Lagrange polynomial*** is defined by

$$\phi_i(x) = \frac{(x-a_1)(x-a_2)\cdots(x-a_{i-1})(x-a_{i+1})\cdots(x-a_n)}{(a_i-a_1)(a_i-a_2)\cdots(a_i-a_{i-1})(a_i-a_{i+1})\cdots(a_i-a_n)}$$

for $i = 1, 2, \ldots, n$. The Lagrange polynomials have the property that $\phi_i(a_i) = 1, \phi_i(a_j) = 0$ for $j \neq i$, and $p(x) = \sum_{i=1}^{n} p(a_i)\phi_i(x)$ for any polynomial of degree $n-1$. For convenience,

$$d_i = (a_i-a_1)(a_i-a_2)\cdots(a_i-a_{i-1})(a_i-a_{i+1})\cdots(a_i-a_n)$$

is called the ***normalization divisor*** of $\phi_i(x)$. The following subroutines are available for computing the Lagrange polynomials and their normalization divisors.

CALL LGRNGN(A, n, D)

A and D are arrays of dimension n. The arguments $a_1, \ldots, a_n$ are given in the array A. The normalization divisors $d_1, \ldots, d_n$ are computed by the routine and stored in D.

Programmer. Allen V. Hershey.

CALL LGRNGV(MO,n, x_0, A, D, F,DF,DDF)

A and D are arrays of dimension n. The arguments $a_1, \ldots, a_n$ are given in A and the normalization divisors $d_1, \ldots, d_n$ are given in D. The argument x_0 is an arbitrary real number and F,DF,DDF are arrays of dimension n.

The argument MO may take the values $0, 1, 2$. If MO $= 0$ then the polynomials $\phi_i(x)$ are evaluated at x_0 and the values $\phi_i(x_0)$ stored in F. If MO $= 1$ then the function $\phi_i(x)$ and its derivative $\phi_i'(x)$ are computed at x_0. In this case, $\phi_i(x_0)$ is stored in $F(i)$ and $\phi_i'(x_0)$ is stored in DF(i) for $i = 1, \ldots, n$. Similarly, if MO $= 2$ then the function $\phi_i(x)$ and its first and second derivatives are computed at x_0. The values $\phi_i(x_0)$ are stored in F, the first derivatives are stored in DF, and the second derivatives are stored in DDF.

Note. If MO $= 0$ then DF and DDF are ignored by the routine. Similarly, if MO $= 1$ then DDF is ignored.

Programmer. Allen V. Hershey.

CALL LGRNGX(A, n, C)

A is an array of dimension n and C an array of dimension $n \times (n+1)$. The arguments $a_1, \ldots, a_n$ are given in A. The purpose of the routine is to compute the coefficients $c_{i,j}$ of the Lagrange polynomials

$$\phi_j(x) \quad \sum_{k=0}^{n-1} c_{k+1,j} x^k.$$

When LGRNGX is called, the coefficients of $\phi_j(x)$ are stored in the j^{th} column of C for $j \leq n$. Also, the first n coefficients of the polynomial

$$g(x) = (x - a_1) \cdots (x - a_n)$$

are stored in the $(n+1)^{st}$ column of C.

Programmer. Allen V. Hershey.

ORTHOGONAL POLYNOMIALS ON FINITE SETS

Let $u_1, \ldots, u_n$ be n distinct real numbers. For any real-valued functions f, g defined on the points u_i let $(f,g) = \sum_{i=1}^{n} f(u_i)g(u_i)$. Then (f,g) is an inner product when f and g are regarded as functions defined only on u_i. Thus, an orthonormal set of polynomials $\{\phi_0, \phi_1, \ldots, \phi_{n-1}\}$ exists where the degree of ϕ_j is j for $j < n$. The polynomials ϕ_j are defined recursively by

$$\phi_{j+1}(u) = \frac{1}{a_j}\left[(u - b_j)\phi_j(u) - a_{j-1}\phi_{j-1}(u)\right]$$

where $a_j = (\phi_{j+1}, u\phi_j)$ and $b_j = (\phi_j, u\phi_j)$. Here it is assumed that $\phi_{-1} = a_{-1} = 0$ and $\phi_0(u) = 1/\sqrt{n}$. The following subroutines are available for computing these polynomials.

CALL ORTHOS(U, m, P, n, R)

U is an array containing the values $u_1, \ldots, u_n$ and m is an integer such that $1 \le m \le n$. P is an array of dimension $n \times m$ and R an array of dimension $2m - 2$. When ORTHOS is called, $\phi_{j-1}(u_i)$ is computed and stored in $P(i,j)$ for $i \le n$ and $j \le m$. Also the coefficients $a_0, b_0, a_1, b_1, \ldots, a_{m-2}, b_{m-2}$ are stored in R.

Programmer. Allen V. Hershey.

CALL ORTHOV(MO,n, u, R, m, F,DF,DDF)

The argument u is a real number and m an integer such that $1 \le m < n$. R is an array containing the coefficients $a_0, b_0, a_1, b_1, \ldots, a_{m-2}, b_{m-2}$ and F,DF,DDF are arrays of dimension m.

MO may take the values 0, 1, 2. If MO = 0 then $\phi_0, \phi_1, \ldots, \phi_{m-1}$ are evaluated at u and the values $\phi_{j-1}(u)$ stored in F. If MO = 1 then ϕ_{j-1} and its derivative ϕ'_{j-1} are computed at u. In this case, $\phi_{j-1}(u)$ is stored in $F(j)$ and $\phi'_{j-1}(u)$ is stored in DF(j) for $j = 1, \ldots, m$. Similarly, if MO = 2 then ϕ_{j-1} and its first and second derivatives are evaluated at u. The values $\phi_{j-1}(u)$ are stored in F, the first derivatives are stored in DF, and the second derivatives are stored in DDF.

Note. If MO = 0 then DF and DDF are ignored by the routine. Similarly, if MO = 1 then DDF is ignored.

Programmer. Allen V. Hershey

CALL ORTHOX(n, R, m, C)

The argument m is an integer such that $1 < m < n$. R is an array containing the coefficients $a_0, b_0, a_1, b_1, \ldots, a_{m-2}, b_{m-2}$ and C an array of dimension $m \times m$. The purpose

ZEROS OF CONTINUOUS FUNCTIONS

Let $F(x)$ be a continuous real-valued function defined for $a \leq x \leq b$, and assume that $F(a)$ and $F(b)$ have opposite signs. Then the following functions are available for finding a point x in the interval $[a,b]$ for which $F(x) = 0$.

ZEROIN(F, a, b,AERR,RERR)

$F(x)$ is a user defined function whose arguments and values are assumed to be real numbers. ZEROIN returns a value x in the interval $[a,b]$ for which $F(x) = 0$.

AERR and RERR are the absolute and relative tolerances to be satisfied (AERR ≥ 0 and RERR ≥ 0). One may set RERR $= 10^{-k}$ if it is desired that x be accurate to k significant decimal digits. If RERR $= 0$ then it is assumed that machine accuracy is desired.

Remark. F must be declared in the calling program to be of type EXTERNAL.

References.

(1) Brent, Richard, ***Algorithms for Minimization without Derivatives***, Prentice-Hall, 1973.

(2) Forsythe, G.E., Malcolm, M.A., and Moler, C.B., ***Computer Methods for Mathematical Computations***, Prentice-Hall, 1977.

Programming. ZEROIN is a slightly modified translation of the ALGOL 60 procedure ZERO given in reference (1). The code was distributed by G. E. Forsythe, M. A. Malcolm, and C. B. Moler (University of New Mexico), and modified by A. H. Morris. The function SPMPAR is used.

DZERO(F, a, b,AERR,RERR)

The arguments a, b, AERR, and RERR are double precision numbers, and $F(x)$ is a user defined function whose arguments and values are assumed to be double precision numbers. DZERO returns a double precision value x in the interval $[a,b]$ for which $F(x)$ 0.

AERR and RERR are the absolute and relative tolerances to be satisfied (AERR 0 and RERR 0). One may set RERR 10 ^{k} if it is desired that x be accurate to k significant decimal digits. If RERR 0 then it is assumed that machine accuracy is desired.

Remarks. F must be declared in the calling program to be of type DOUBLE PRECISION and EXTERNAL, and DZERO must be declared to be of type DOUBLE PRECISION.

References.

(1) Brent, Richard, ***Algorithms for Minimization without Derivatives***, Prentice-Hall, 1973.

(2) Forsythe, G E , Malcolm, M A , and Moler, C B , ***Computer Methods for Mathematical Computations***, Prentice-Hall, 1977

Programming. DZERO is a slightly modified translation of the ALGOL 60 procedure ZERO given in reference (1). The code was distributed by G. E. Forsythe, M. A. Malcolm, and C. B. Moler (University of New Mexico), and modified by A. H. Morris. The function DPMPAR is used.

SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS

Let $f_i(x) = 0 \quad (i = 1, ..., n)$ denote a system of n equations in n unknowns where $x = (x_1, \ldots, x_n)$. Assume that each $f_i(x)$ is differentiable and that an initial guess $a = (a_1, \ldots, a_n)$ to a solution of the equations is given. Then the following subroutine is available for solving the equations to within a specified tolerance.

CALL HBRD(F, n, X,FVEC,EPS,TOL,INFO,WK,ℓ)

X and FVEC are arrays of dimension n or larger. On input X contains the starting point $a = (a_1, \ldots, a_n)$. When HBRD terminates, X contains the final estimate $x = (x_1, \ldots, x_n)$ of the solution vector and FVEC contains the values of the functions $f_1, \ldots, f_n$ at the output point in X.

The argument F is the name of a user defined subroutine that has the format:

CALL $F(n, X$,FVEC,IFLAG)

Here X and FVEC are arrays of dimension n and IFLAG is an integer variable. The array X contains a point $x = (x_1, \ldots, x_n)$. Normally F will evaluate the functions $f_1, \ldots, f_n$ at this point and store the results in FVEC. However, if x does not lie in the domain of $f_1, \ldots, f_n$ then this cannot be done. In this case, the argument IFLAG (which will have been assigned a nonnegative value by HBRD) should be reset by F to a negative value. This will signal HBRD to terminate. F must be declared in the calling program to be of type EXTERNAL.

EPS is an input argument which specifies the relative accuracy of F. If it is estimated that the subroutine F produces results accurate to k significant decimal digits then one may set EPS $= 10^{-k}$. It is required that EPS ≥ 0. If EPS $= 0$ then it is assumed that F produces results accurate to machine precision.

TOL is an input argument which specifies the desired accuracy of the solution. The Euclidean norm $||x|| = \sqrt{\Sigma_i x_i^2}$ is employed. If $\bar{x}$ denotes an actual solution of the equations, then HBRD terminates when an iterate x is generated for which it is estimated that $||x - \bar{x}|| \leq \text{TOL} \cdot ||x||$ is satisfied. It is required that TOL ≥ 0. In order for the convergence test to work properly, it is recommended that TOL always be smaller than 10^{-5}.

WK is an array of dimension ℓ that is used for a work space. It is assumed that the argument ℓ is greater than or equal to $n(3n + 13)/2$.

INFO is an integer variable that reports the status of the results. When HBRD terminates, INFO has one of the following values:

INFO < 0 This occurs when the user terminates the execution of HBRD by resetting the argument IFLAG in the subroutine F to a negative value. Then INFO = the negative value of IFLAG.

INFO $= 0$ (Input Error) $n < 1$, EPS < 0, TOL < 0, or $\ell < n(3n + 13)/2$.

INFO $= 1$ A solution having the desired accuracy was obtained.

INFO = 2 The number of calls to the subroutine F has reached or exceeded $200(n+1)$.

INFO = 3 TOL is too small. No further improvement in the accuracy of x is possible.

INFO = 4 The routine is making very poor progress.

When HBRD terminates, if INFO $\neq 0$ then X contains the final iterate that was generated. Also, if INFO ≥ 1 then FVEC contains the values of the functions $f_1, \ldots, f_n$ at this iterate. If INFO = 2 then it may be helpful to continue the procedure by recalling HBRD with the current point in X as the new starting point. However, this is not advisable when INFO = 4. This setting can arise when $x = 0$ is a solution or when entrapment occurs. HBRD searches for a solution to the equations by minimizing $\sum_i f_i(x)^2$. In doing this, it can become trapped in a region where the minimum does not correspond to a solution of the equations. This is what occurs when the equations have no solution. When entrapment occurs and the equations are known to have a solution, then it is recommended that the user try a different starting point.

Scaling. If the convergence criterion $||x - \bar{x}|| \leq \text{TOL} \cdot ||\bar{x}||$ is satisfied and $\text{TOL} = 10^{-k}$, then the larger components of the final iterate x may be accurate to k significant digits but not the smaller components. For example, if $\text{TOL} = 10^{-5}$ and $x = (1.2, .34\text{E-}4)$, then 1.2 may be accurate to 5 significant digits while .34E-4 is accurate to only 1 significant digit. If it is suspected that the smaller components do not have acceptable accuracy, then it is recommended that the variables in the original problem be rescaled and the problem rerun.

Algorithm. A modified form of the hybrid Powell procedure is employed.

Programming. HBRD is a slightly modified version of the MINPACK–1 subroutine HYBRD1. The MINPACK–1 subroutines HYBRD, ENORM, DOGLEG, FDJAC1, QFORM, QRFAC, R1MPYQ, and R1UPDT are employed. The subroutines were written by Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom (Argonne National Laboratory). The function SPMPAR is also used.

References.

(1) More, J. J., Garbow, B.S., and Hillstrom, K. E., ***User Guide for MINPACK–1***, Argonne Nation Laboratory Report ANL–80–74, Argonne, Illinois, 1980.

(2) Powell, M.J.D., A Hybrid Method for Nonlinear Equations," ***Numerical Methods for Nonlinear Algebraic Equations***, P. Rabinowitz (ed.), Gordon and Breach, London, 1970.

SOLUTIONS OF QUADRATIC, CUBIC, AND QUARTIC EQUATIONS

Given a polynomial $a_0 + a_1 z + \cdots + a_n z^n$ with real coefficients where $a_n \neq 0$ and $n = 2, 3$ or 4. The following subroutines are available for computing the roots $z_1, \ldots, z_n$ of the polynomial.

CALL QDCRT(A, Z)
CALL CBCRT(A, Z)
CALL QTCRT(A, Z)

It is assumed that A is a real array and Z a complex array. QDCRT is used if $n = 2$, CBCRT is used if $n = 3$, and QTCRT is used if $n = 4$. A is the array of coefficients where $A(k) = a_{k-1}$ for $k = 1, 2, \ldots, n+1$, and Z is an array of dimension n. When the appropriate subroutine is called, the roots $z_1, \ldots, z_n$ are stored in Z. The real roots precede the complex roots. The real roots are ordered so that $|z_j| \leq |z_{j+1}|$. The complex roots are unordered except that complex conjugate pairs of roots appear consecutively with the root having the positive imaginary part being first.

Programming. QTCRT calls the subroutines CBCRT and AORD, and CBCRT calls the subroutine QDCRT and function CBRT. The routines were written by A. H. Morris and CBCRT was modified by Wm. Davis (NSWC). The function SPMPAR is also used.

CALL DQDCRT(A,ZR,ZI)
CALL DCBCRT(A,ZR,ZI)
CALL DQTCRT(A,ZR,ZI)

It is assumed that A,ZR, and ZI are double precision arrays. DQDCRT is used if $n = 2$, DCBCRT is used if $n = 3$, and DQTCRT is used if $n = 4$. A is the array of coefficients where $A(k) = a_{k-1}$ for $k = 1, 2, \ldots, n+1$, and ZR and ZI are arrays of dimension n. When the appropriate subroutine is called, the real parts of the roots $z_1, \ldots, z_n$ are stored in ZR and the imaginary parts are stored in ZI. The real roots precede the complex roots. The real roots are ordered so that $|z_j| \leq |z_{j+1}|$. The complex roots are unordered except that complex conjugate pairs of roots appear consecutively with the root having the positive imaginary part being first.

Programming. DQTCRT calls the routines DAORD, DCSQRT, and DCBCRT. DCBCRT calls the subroutine DQDCRT and function DCBRT. The routines were written by A. H. Morris. The function DPMPAR is also used.

DOUBLE PRECISION ROOTS OF POLYNOMIALS

Given a polynomial $a_0 + a_1 z + \cdots + a_n z^n$ of degree $n \geq 1$. The subroutines DRPOLY and DCPOLY are available for obtaining the roots $z_1, \ldots, z_n$ of the polynomial. DRPOLY may be used if the coefficients a_j are real, and DCPOLY is applicable if the coefficients are complex. These subroutines perform the calculations in double precision.

CALL DRPOLY(A, n,ZR,ZI,NUM,WK,DWK)

A is a double precision array containing the coefficients where $A(j) = a_{n-j+1}$ for $j = 1, \ldots, n+1$. ZR and ZI are double precision arrays of dimension n or larger, and NUM is an integer variable. When DRPOLY is called, if no errors are detected then NUM = the number of roots that are obtained. If NUM ≥ 1 then the real parts of the roots are stored in ZR(j) and the imaginary parts in ZI(j) for $j = 1, \ldots,$NUM. The roots are unordered except that complex conjugate pairs of roots appear consecutively with the positive imaginary part being first.

WK is a real array of dimension $n + 1$ or larger, and DWK is a double precision array of dimension $6(n + 1)$ or larger. WK and DWK are work spaces for the routine.

Error Return. NUM $= -1$ if $n < 1$ or $a_n = 0$.

Programming. DRPOLY employs the subroutines DRPLY1,FXSHFR,QUADIT,REALIT, CALCSC, NEXTK, NEWEST, QUADSD, and QUADPL. These routines exchange information in a labeled common block named GLOBAL. The routines were written by M. A. Jenkins (Queen's University, Kingston, Ontario), and modified by A. H. Morris. The functions SPMPAR, DPMPAR, and IPMPAR are also used.

References.

(1) Jenkins, M. A.,"Zeros of a Real Polynomial," ***ACM Trans. Math Software*** **1** (1975), pp. 178–189.

(2) Jenkins, M. A. and Traub, J. F.,"A Three-Stage Algorithm for Real Polynomials using Quadratic Iterations," ***SIAM J. Numerical Analysis*** **7** (1970), pp. 545–566.

CALL DCPOLY(AR,AI,n,ZR,ZI,NUM,DWK)

AR and AI are double precision arrays containing the real and imaginary parts of the coefficients where AR(j) = Re(a_{n-j+1}) and AI(j) = Im(a_{n-j+1}) for $j = 1, \ldots, n+1$. ZR and ZI are double precision arrays of dimension n or larger, and NUM is an integer variable. When DCPOLY is called, if no errors are detected then NUM = the number of roots that are obtained. If NUM ≥ 1 then the real parts of the roots are stored in ZR(j) and the imaginary parts in ZI(j) for $j = 1, \ldots,$NUM.

DWK is a double precision array of dimension $10(n + 1)$ or larger that is a work space for the routine.

Error Return. NUM $= -1$ if $n < 1$ or $a_n = 0$.

Programming. DCPOLY employs the routines DCPLY1, CAUCHY, NOSHFT, FXSHFT, VRSHFT, CALCT, NEXTH, POLYEV, CDIVID, and the functions SCALCP, ERREV. These routines and functions were written by M.A. Jenkins (Queen's University, Kingston, Ontario) and J. F. Traub (Bell Laboratories), and modified by A. H. Morris. The functions DPMPAR, IPMPAR, and DCPABS are also used.

References.

(1) Jenkins, M. A. and Traub, J. F., "Algorithm 419, Zeros of a Complex Polynomial," ***Comm. ACM*** **15** (1972), pp. 97–99.

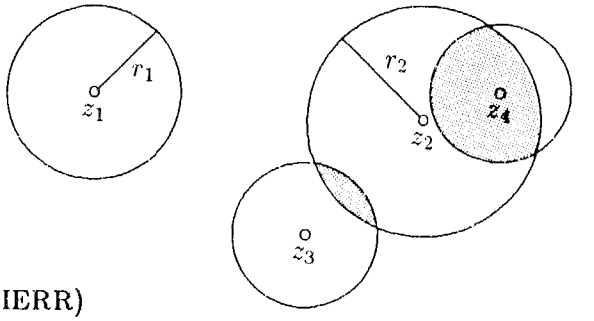
(2) _____, "A Three-Stage Variable-Shift Iteration for Polynomial Zeros and its Relation to Generalized Rayleigh Iteration," ***Numer. Math*** **14** (1970), pp. 252–263.

ACCURACY OF THE ROOTS OF POLYNOMIALS

Let $z_1, \ldots, z_n$ be approximations for the roots of a real or complex polynomial $p(z) = a_0 + a_1 z + \cdots + a_n z^n$ of degree $n \geq 1$. Then for each z_i, the subroutines RBND and CBND obtain the radius r_i of a disk $D_i = \{z : |z - z_i| \leq r_i\}$ centered at z_i which contains a true zero of the polynomial. The radius r_i is an upper bound on the absolute error of the approximation z_i.

For each z_i, the subroutines also compute the number k_i of disks D_j (including the disk D_i) which overlap with D_i. The value k_i is the number of roots of $p(z)$ that are clustered near z_i. If $k_i = 1$ then z_i approximates a simple root.

Example. In the figure $k_1 = 1$, $k_2 = 3$, $k_3 = 2$, and $k_4 = 2$.



CALL RBND$(n, A, Z, R, \text{RERR}, K, \text{IERR})$
CALL CBND$(n, A, Z, R, \text{RERR}, K, \text{IERR})$

A is a real array if RBND is used, and A is a complex array if CBND is used. A contains the coefficients $a_0, a_1, \ldots, a_n$ where $A(i) = a_{i-1}$ $(i = 1, \ldots, n+1)$, and Z is a complex array containing the approximate roots $z_1, \ldots, z_n$.

IERR is an integer variable, R a real array of dimension n or larger, and K an integer array of dimension n or larger. When RBND or CBND is called, if no input errors are detected then IERR is assigned the value 0, the radii $r_1, \ldots, r_n$ are computed and stored in R, and the values $k_1, \ldots, k_n$ are computed and stored in K.

RERR is a real array of dimension n or larger. If $z_i = 0$ then RERR(i) is set to -1 by the routine, and if $z_i \neq 0$ then RERR(i) = the estimated maximum relative error of z_i.

Error Return. IERR = 1 when $n < 1$ and IERR = 2 when $a_n = 0$. In these cases no computation is performed.

Programming. RBND and CBND employ the functions CPABS and SPMPAR. RBND and CBND were written by Carl B. Bailey and modified by William R. Gavin (Sandia Laboratories). The formats of the subroutines were modified by A. H. Morris.

COPYING VECTORS

A copy of a vector $X = (x_1, \ldots, x_n)$ can be made and stored in the array Y by the following subroutines:

CALL SCOPY(n, X, kx, Y, ky)
CALL DCOPY(n, X, kx, Y, ky)
CALL CCOPY(n, X, kx, Y, ky)

The argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of Y.

SCOPY is used if X and Y are real arrays, DCOPY is used if X and Y are double precision arrays, and CCOPY is used if X and Y are complex arrays. When any of these routines is called, if $n \leq 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then the components x_i of X are stored in Y according to the spacing specified by the ky parameter.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The routines were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

INTERCHANGING VECTORS

The components of two vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ can be interchanged by the following subroutines:

CALL SSWAP(n, X, kx, Y, ky)
CALL DSWAP(n, X, kx, Y, ky)
CALL CSWAP(n, X, kx, Y, ky)

The argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of Y.

SSWAP is used if X and Y are real arrays, DSWAP is used if X and Y are double precision arrays, and CSWAP is used if X and Y are complex arrays. When any of these routines is called, if $n \leq 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then the components x_i and y_i are interchanged for $i = 1, \ldots, n$. Thus, when the routine terminates $X = (y_1, \ldots, y_n)$ and $Y = (x_1, \ldots, x_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The routines were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

PLANAR ROTATION OF VECTORS

Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be vectors and c and s be real numbers such that $c^2 + s^2 = 1$. X and Y can be replaced with $cX + sY$ and $-sX + cY$ by the following subroutines:

CALL SROT(n, X, kx, Y, ky, c, s)
CALL DROT(n, X, kx, Y, ky, c, s)
CALL CSROT(n, X, kx, Y, ky, c, s)

The argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of Y.

SROT is used if X and Y are real arrays, DROT is used if X and Y are double precision arrays, and CSROT is used if X and Y are complex arrays. The arguments c and s are real numbers when SROT and CSROT are used, and are double precision numbers when DROT is used. When any of these routines is called, if $n \leq 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then the components x_i and y_i are replaced with $cx_i + sy_i$ and $-sx_i + cy_i$ for $i = 1, \ldots, n$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The routines were written by Jack Dongarra (Argonne National Laboratory)

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

MODIFIED GIVENS ROTATIONS

If a_1 and a_2 are real numbers where $r^2 = a_1^2 + a_2^2 \neq 0$, then there is an orthogonal matrix $G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ such that $G \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$. In this case $c = a_1/r$ and $s = a_2/r$. The matrix G represents what is called a ***Givens rotation***. Given $A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$, the matrix G is uniquely defined up to the sign of r.

Alternatively, assume that instead of having A we are given $D = \begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix}$ and $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ where $d_i > 0$ $(i = 1, 2)$ and $A = D^{1/2}B$. Then there exist matrices $\tilde{D} = \begin{pmatrix} \tilde{d}_1 & 0 \\ 0 & \tilde{d}_2 \end{pmatrix}$ and $H = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$ where $\tilde{d}_1$ and $\tilde{d}_2$ are positive values and

$$(1) \qquad \begin{aligned} HB &= \begin{pmatrix} \tilde{b}_1 \\ 0 \end{pmatrix} \\ GD^{1/2} &= \sigma \tilde{D}^{1/2} H \qquad (\sigma = \pm 1). \end{aligned}$$

In this case, $GA = GD^{1/2}B = \sigma \tilde{D}^{1/2}(HB)$.

If $d_1 b_1^2 > d_2 b_2^2$ then H and $\tilde{D}$ can be defined by

$$(2) \qquad \begin{aligned} h_{11} &= 1 & h_{12} &= \frac{d_2 b_2}{d_1 b_1} \\ h_{21} &= -\frac{b_2}{b_1} & h_{22} &= 1 \\ u &= 1 - h_{12} h_{21} & & \\ \tilde{d}_1 &= d_1/u & \tilde{d}_2 &= d_2/u, \end{aligned}$$

in which case $\tilde{b}_1 = ub_1$ and $\sigma = \operatorname{sgn}(c)$. Otherwise, if $d_1 b_1^2 \leq d_2 b_2^2$ then H and $\tilde{D}$ can be defined by

$$(3) \qquad \begin{aligned} h_{11} &= \frac{d_1 b_1}{d_2 b_2} & h_{12} &= 1 \\ h_{21} &= -1 & h_{22} &= \frac{b_1}{b_2} \\ u &= 1 + h_{11} h_{22} & & \\ \tilde{d}_1 &= d_2/u & \tilde{d}_2 &= d_1/u, \end{aligned}$$

in which case $\tilde{b}_1 = ub_2$ and $\sigma = \operatorname{sgn}(s)$.

Any H which satisfies (1) is called a ***modified Givens rotation matrix***. H is frequently preferred to G since its use may require fewer multiplications, and square roots are avoided. For convenience, if $b_2 = 0$ or $d_2 = 0$ then H is defined to be the identity matrix and $\tilde{D} = D$. Otherwise, if $b_1 = 0$ or $d_1 = 0$ then formulas (3) are applied.

It is normally assumed that $d_1 \geq 0$. However, there are applications where it is convenient to let $d_2 < 0$. If $d_1 > 0$ and $d_2 < 0$ then $\tilde{D}$ and H can be defined by (2) or (3) depending on whether $|d_1 b_1^2| > |d_2 b_2^2|$ or $|d_1 b_1^2| \leq |d_2 b_2^2|$. If (3) are used and $u \neq 0$, then $\tilde{d}_1 < 0$, which is not compatible with the requirement $d_1 \geq 0$. Consequently, $d_2 < 0$ is permitted only when $|d_1 b_1^2| > |d_2 b_2^2|$, and we note that $0 < u < 1$. Since u can be near 0, $\tilde{D}$ and H may have to be rescaled to avoid overflow and underflow. Rescaling may also be needed when $d_2 > 0$ (in this case $1 \leq u \leq 2$).

The subroutines SROTMG and DROTMG are available for obtaining $\tilde{D}$ and H, and the subroutines SROTM and DROTM are available for computing $H \begin{pmatrix} x \\ y \end{pmatrix}$ for any $\begin{pmatrix} x \\ y \end{pmatrix}$.

CALL SROTMG(D_1, D_2, B_1, B_2, Q)
CALL DROTMG(D_1, D_2, B_1, B_2, Q)

D_1, D_2, B_1, B_2 are real variables and Q a real array if SROTMG is used, and D_1, D_2, B_1, B_2 are double precision variables and Q a double precision array if DROTMG is used. Q is an array of dimension 5.

It is assumed that $D_i = d_i$ and $B_i = b_i$ $(i = 1, 2)$ on input and that $d_1 \geq 0$. The argument d_2 may be negative or nonnegative. If $d_2 < 0$ then it is assumed that $|d_1 b_1^2| > |d_2 b_2^2|$. When SROTMG or DROTMG is called, D_1 and D_2 are reset to $\tilde{d}_1$ and $\tilde{d}_2$, and B_1 is reset to $\tilde{b}_1$. B_2 is not modified, and H is stored in Q as follows.

$Q(1)$ is an indicator which specifies the elements of H that are stored in Q. If $Q(1) = -1$ then all the elements are stored and

$$Q(2) = h_{11} \qquad Q(4) = h_{12}$$
$$Q(3) = h_{21} \qquad Q(5) = h_{22}.$$

Otherwise, $Q(1)$ has one of the following values:

Q(1) = 0 H is defined by (2). $Q(3) = h_{21}$ and $Q(4) = h_{12}$. $Q(2)$ and $Q(5)$ are not defined.

Q(1) = 1 H is defined by (3). $Q(2) = h_{11}$ and $Q(5) = h_{22}$. $Q(3)$ and $Q(4)$ are not defined.

Q(1) = 2 H is the identity matrix. $Q(2), \ldots, Q(5)$ are not defined.

Error Return. If $d_1 < 0$ or $|d_1 b_1^2| < |d_2 b_2^2|$ when $d_2 < 0$, then $\tilde{D}$ and H are set to the zero matrix and $\tilde{b}_1 = 0$. In this case, $Q(1) = -1$ and $Q(i) = 0$ for $i = 2, \ldots, 5$.

Programming. These subroutines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. SROTMG and DROTMG were modified by A. H. Morris.

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

CALL SROTM(n, X, kx, Y, ky, Q)
CALL DROTM(n, X, kx, Y, ky, Q)

Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be vectors, and Q be the array obtained by SROTMG or DROTMG. If H is the modified Givens rotation matrix stored in Q, then SROTM and DROTM are available for computing $\begin{pmatrix} \bar{x}_i \\ \bar{y}_i \end{pmatrix} = H \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ $(i = 1, \ldots, n)$.

The argument kx is an integer which specifies the interval between succesive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of Y.

SROTM is used if X, Y, and Q are real arrays, and DROTM is used if X, Y, and Q are double precision arrays. When SROTM or DROTM is called, if $n \leq 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then the components x_i and y_i are replaced with $\bar{x}_i$ and $\bar{y}_i$ for $i = 1, \ldots, n$.

Programming. These subroutines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh.

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

DOT PRODUCTS OF VECTORS

The following functions are available for computing the sums $\sum_i x_i y_i$ and $\sum_i \bar{x}_i y_i$ where $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ are real or complex vectors.

SDOT(n, X, kx, Y, ky)
DDOT(n, X, kx, Y, ky)
CDOTC(n, X, kx, Y, ky)
CDOTU(n, X, kx, Y, ky)

The argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of Y.

SDOT is used if X and Y are real arrays, and DDOT is used if X and Y are double precision arrays. SDOT is a real function and DDOT is a double precision function. When either of these two functions is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then the function is assigned the value $\sum_{i=1}^{n} x_i y_i$.

CDOTC and CDOTU are used if X and Y are complex arrays. CDOTC and CDOTU are complex functions. When either of these two functions is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then CDOTC(n, X, kx, Y, ky) is assigned the value $\sum_{i=1}^{n} \bar{x}_i y_i$ and CDOTU (n, X, kx, Y, ky) is assigned the value $\sum_{i=1}^{n} x_i y_i$.

Remark. DDOT must be declared in the calling program to be of type DOUBLE PRECISION, and CDOTC and CDOTV must be declared to be of type COMPLEX.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The functions were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Trans. Math Software* **5** (1979), pp. 308–323.

SCALING VECTORS

If a is a real or complex number and $X = (x_1, \ldots, x_n)$ a vector, then the vector X can be replaced with the vector aX by the following subroutines:

CALL SSCAL(n, a, X, kx)
CALL DSCAL(n, a, X, kx)
CALL CSCAL(n, a, X, kx)
CALL CSSCAL(n, a, X, kx)

The argument kx is a positive integer. It is assumed that the component x_i is stored in $X(1 + (i-1)kx)$ for $i = 1, \ldots, n$.

SSCAL is used if a is a real number and X a real array, DSCAL is used if a is a double precision number and X a double precision array, CSCAL is used if a is a complex number and X a complex array, and CSSCAL is used if a is a real number and X a complex array. When any of these routines is called, if $n \leq 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then each x_i is replaced with ax_i. Thus when the routine terminates $X = (ax_1, \ldots, ax_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The routines were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Trans. Math Software* **5** (1979), pp. 308–323.

VECTOR ADDITION

If a is a real or complex number and $X = (x_1, \ldots, x_n)$ a vector, then any vector $Y = (y_1, \ldots, y_n)$ can be replaced with the vector $aX + Y$ by the following subroutines:

CALL SAXPY(n, a, X, kx, Y, ky)
CALL DAXPY(n, a, X, kx, Y, ky)
CALL CAXPY(n, a, X, kx, Y, ky)

The argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geq 0$ then it is assumed that x_i is stored in $X(1+(i-1)kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1+(n-i)|kx|)$. Similarly, the argument ky specifies the spacing of the components of the vector Y.

SAXPY is used if a is a real number and X, Y are real arrays, DAXPY is used if a is a double precision number and X, Y are double precision arrays, and CAXPY is used if a is a complex number and X, Y are complex arrays. When any of these routines is called, if $n \leq 0$ or $a = 0$ then the routine immediately terminates. Otherwise, if $n \geq 1$ then y_i is replaced with $ax_i + y_i$ for $i = 1, \ldots, n$. Thus when the routine terminates $Y = (ax_1 + y_1, \ldots, ax_n + y_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The routines were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Trans. Math Software* **5** (1979), pp. 308–323.

L_1 NORM OF A VECTOR

The following functions are available for computing the L_1 norm of a real vector or a modified L_1 norm of a complex vector.

SASUM(n, X, kx)
DASUM(n, X, kx)
SCASUM(n, X, kx)

$X = (x_1, \ldots, x_n)$ is a vector and kx a positive integer. It is assumed that x_i is stored in $X(1 + (i-1)kx)$ for $i = 1, \ldots, n$.

SASUM is used if X is a real array and DASUM is used if X is a double precision array. SASUM is a real function and DASUM a double precision function. When either of these functions is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then the function is assigned the value $\sum_{i=1}^{n} |x_i|$.

SCASUM is used if X is a complex array. SCASUM is a real function. When SCASUM is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then SCASUM(n, X, kx) is assigned the value $\sum_{i=1}^{n} [|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)|]$.

Remarks.

(1) DASUM must be declared in the calling program to be of type DOUBLE PRECISION.
(2) SCASUM(n, X, kx) is the norm of the complex vector $X = (x_1, \ldots, x_n)$ when X is regarded as a real vector of dimension $2n$. This norm is frequently preferred over the standard L_1 norm $\sum_{i=1}^{n} |x_i|$ since it takes less time to compute.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The functions were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Trans. Math Software* 5 (1979), pp. 308–323.

L_2 NORM OF A VECTOR

The following functions are available for computing the L_2 norm of a real or complex vector.

SNRM2(n, X, kx)
DNRM2(n, X, kx)
SCNRM2(n, X, kx)

$X = (x_1, \ldots, x_n)$ is a vector and kx a positive integer. It is assumed that x_i is stored in $X(1 + (i-1)kx)$ for $i = 1, \ldots, n$.

SNRM2 is used if X is a real array, DNRM2 is used if X is a double precision array, and SCNRM2 is used if X is a complex array. SNRM2 and SCNRM2 are real functions and DNRM2 a double precision function. When any of these functions is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then the function is assigned the value $\left[\sum_{i=1}^{n} |x_i|^2\right]^{1/2}$.

Remark. The function DNRM2 must be declared in the calling program to be of type DOUBLE PRECISION.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The functions were written by Charles Lawson (Jet Propulsion Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

L_∞ NORM OF A VECTOR

The following functions are available for finding the largest component x_i of a vector $X = (x_1, \ldots, x_n)$.

ISAMAX(n, X, kx)
IDAMAX(n, X, kx)
ICAMAX(n, X, kx)

The argument kx is a positive integer. It is assumed that the component x_i is stored in $X(1 + (i-1)kx)$ for $i = 1, \ldots, n$.

ISAMAX is used if X is a real array and IDAMAX is used if X is a double precision array. ISAMAX and IDAMAX are integer functions. When either of these functions is called, if $n \leq 0$ then the function is assigned the value 0. Otherwise, if $n \geq 1$ then the function is assigned the value i where i is the smallest index such that $|x_i| = \max\{|x_j| : j = 1, \ldots, n\}$.

ICAMAX is also an integer function. It is used when X is a complex array. If $n \leq 0$ then ICAMAX(n, X, kx) is assigned the value 0. Otherwise, if $n \geq 1$ then the function is assigned the value i where i is the smallest index such that $|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)| = \max\{|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| : j = 1, \ldots, n\}$.

Note. The mapping $X \to \max\{|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| : j = 1, \ldots, n\}$ is the L_∞ norm of the complex vector $X = (x_1, \ldots, x_n)$ when X is regarded as a real $n \times 2$ matrix. This norm is frequently preferred over the standard L_∞ norm $\max\{|x_j| : j = 1, \ldots, n\}$ since it takes less time to compute.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The functions were written by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," ***ACM Trans. Math Software*** **5** (1979), pp. 308–323.

PACKING AND UNPACKING SYMMETRIC MATRICES

An $n \times n$ symmetric matrix $A = (a_{ij})$ can be represented by either its lower triangular elements

$$\begin{pmatrix} \underline{a_{11}} & a_{12} & a_{13} & \cdots \\ \underline{a_{21}} & \underline{a_{22}} & a_{23} & \cdots \\ \underline{a_{31}} & \underline{a_{32}} & \underline{a_{33}} & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}$$

or its upper triangular elements. If the lower triangular elements are used, then the packed form for the matrix is an array of dimension $n(n+1)/2$ containing the elements $a_{11}a_{21}a_{22}a_{31}a_{32}a_{33}\cdots$ Similarly, if the upper triangular elements are used then the packed form for the matrix is an array containing $a_{11}a_{12}\cdots a_{1n}a_{22}a_{23}\cdots a_{2n}\cdots$ ***Currently the lower triangular elements are used for packing symmetric matrices.*** The following subroutines are available for packing and unpacking real symmetric matrices.

CALL MCVFS(A, ka, n, B)
CALL DMCVFS(A, ka, n, B)

A is an $n \times n$ symmetric matrix and B an array whose dimension is equal to or greater than $n(n+1)/2$. The routines store the packed form of A in B. MCVFS is used if A and B are real arrays and DMCVFS is used if A and B are double precision arrays. The input argument ka has the following value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geq n$.

Note. A and B may begin in the same location.

Programmer. A. H. Morris.

CALL MCVSF(A, ka, n, B)
CALL DMCVSF(A, ka, n, B)

B is an array containing the elements of a packed $n \times n$ symmetric matrix and A is an array of dimension $ka \times n$ where $ka \geq n$. The routines unpack B and store the results in A. MCVSF is used if A and B are real arrays and DMCVSF is used if A and B are double precision arrays.

Note. A and B may begin in the same location.

Programmer. A. H. Morris.

CONVERSION OF REAL MATRICES TO AND FROM DOUBLE PRECISION FORM

The subroutines MCVRD and MCVDR are available for converting real matrices to and from double precision form.

CALL MCVRD(m, n, A, ka, B, kb)

A is a real $m \times n$ matrix and B a double precision 2-dimensional array. MCVRD stores the matrix in double precision form in the array B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

CALL MCVDR(m, n, A, ka, B, kb)

A is a double precision $m \times n$ matrix and B a real 2-dimensional array. MCVDR stores the matrix in single precision form in the array B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

STORAGE OF REAL MATRICES IN THE COMPLEX MATRIX FORMAT

The following subroutine is available for storing a real matrix A in the complex matrix format.

CALL MCVRC(m, n, A, ka, B, kb)

A is a real $m \times n$ matrix and B a complex 2-dimensional array. MCVRC stores the matrix in complex form in the array B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

THE REAL AND IMAGINARY PARTS OF A COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\mathrm{Re}(A) = (\mathrm{Re}(a_{ij}))$ and $\mathrm{Im}(A) = (\mathrm{Im}(a_{ij}))$ denote the real and imaginary parts of A. The following subroutines are available for obtaining $\mathrm{Re}(A)$ and $\mathrm{Im}(A)$.

CALL CMREAL(m, n, A, ka, B, kb)

A is a complex $m \times n$ matrix and B a real 2-dimensional array. CMREAL obtains $\mathrm{Re}(A)$ and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

CALL CMIMAG(m, n, A, ka, B, kb)

A is a complex $m \times n$ matrix and B a real 2-dimensional array. CMIMAG obtains $\mathrm{Im}(A)$ and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

COPYING MATRICES

The following subroutines are available for copying matrices.

CALL MCOPY(m, n, A, ka, B, kb)

A is a real $m \times n$ matrix and B a real 2-dimensional array. MCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

CALL SMCOPY(n, A, B)

A is a real packed $n \times n$ symmetric matrix and B a real array whose dimension is equal to or greater than $n(n+1)/2$. SMCOPY makes a copy of the packed symmetric matrix A and stores it in B.

Programmer. A. H. Morris.

CALL DMCOPY(m, n, A, ka, B, kb)

A is a double precision $m \times n$ matrix and B a double precision 2-dimensional array. DMCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

CALL CMCOPY(m, n, A, ka, B, kb)

A is a complex $m \times n$ matrix and B a complex 2-dimensional array. CMCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Programmer. A. H. Morris.

COMPUTATION OF THE CONJUGATE OF A COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\overline{A} = (\overline{a}_{ij})$ denote the conjugate of A. The following subroutine is available for computing the conjugate matrix $\overline{A}$.

CALL CMCONJ(m, n, A, ka, B, kb)

A is a complex $m \times n$ matrix and B a complex 2-dimensional array. CMCONJ computes $\overline{A}$ and stores the results in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq m$, and that B contains at least n columns.

Remark. A and B may reference the same storage area when $ka = kb$.

Programmer. A. H. Morris.

TRANSPOSING MATRICES

The subroutines TPOSE, DTPOSE, CTPOSE and TIP, DTIP, CTIP are available for transposing a matrix A. TPOSE, DTPOSE, and CTPOSE are used if the results are to be stored in a separate storage area B. TIP, DTIP, and CTIP are used if the results are to be stored in A (i.e., if the transposition is to be done in place).

CALL TPOSE(m, n, A, ka, B, kb)
CALL DTPOSE(m, n, A, ka, B, kb)
CALL CTPOSE(m, n, A, ka, B, kb)

TPOSE is called if A is a real matrix and B a real array, DTPOSE is called if A is a double precision matrix and B a double precision array, and CTPOSE is called if A is a complex matrix and B a complex array.

A is a matrix having m rows and n columns, and B a 2-dimensional arrary. The routine transposes A and stores the results in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
It is assumed that $ka \geq m$ and $kb \geq n$, and that B contains at least n columns.

Programmer. A. H. Morris.

CALL TIP$(A, m, n, \text{MOVE}, k, \text{MDIM})$
CALL DTIP$(A, m, n, \text{MOVE}, k, \text{MDIM})$
CALL CTIP$(A, m, n, \text{MOVE}, k, \text{MDIM})$

TIP is called if A is a real array, DTIP is called if A is a double precision array, and CTIP is called if A is a complex array.

A is an array of dimension mn which contains an $m \times n$ matrix to be transposed. The routine transposes the matrix and stores the results in A. If $m = n$ then the arguments MOVE, k, MDIM are ignored.

If $m \neq n$ then k may be any integer. If $k \leq 0$ then MOVE is ignored. Otherwise, if $k \geq 1$ then MOVE is assumed to be an array of dimension k. MOVE is a storage area for saving information that may help speed up the transposition process. If no information is saved then TIP, DTIP, and CTIP may run 2 10 times slower than TPOSE, DTPOSE, and CTPOSE. However, the use of a storage area may or may not actually speed up the transposition process. This depends entirely on the values of m and n. MDIM is a variable that is set by the routine. After the routine terminates, MDIM will be the estimated optimum setting for k for the current values of m and n.

Programming. The routines TIP, DTIP, and CTIP employ the subroutine INFCTR. The routines were written by Norman Brenner (Dept. of Earth and Planetary Sciences, Massachusetts Institute of Technology), and modified by A. H. Morris.

Reference. Brenner, Norman, "Algorithm 467. Matrix Transposition in Place," *Comm. ACM* **16** (1973), pp. 692–694.

COMPUTING ADJOINTS OF COMPLEX MATRICES

If $A = (a_{ij})$ then let $A^* = (\overline{a}_{ji})$ denote the adjcint of A. The following subroutines are available for computing A^*.

CALL CMADJ(m, n, A, ka, B, kb)

A is a complex $m \times n$ matrix and B a complex 2-dimensional array. CMADJ computes A^* and stores the results in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geq m$ and $kb \geq n$, and that B contains at least n columns.

Remark. CMADJ combines the following operations:

CALL CTPOSE(m, n, A, ka, B, kb)

CALL CMCONJ(n, m, B, kb, B, kb)

It is assumed that A and B are different storage areas.

Programmer. A. H. Morris.

CALL CTRANS(ka, n, A)

A is a complex array of dimension $ka \times n$ which contains an $n \times n$ matrix. CTRANS computes the adjoint of the matrix and stores the results in A. It is assumed that $ka \geq n$.

Programmer. George J. Davis (Georgia State University)

MATRIX ADDITION

The matrix sum $C = A + B$ can be computed by the following subroutines:

CALL MADD$(m, n, A, ka, B, kb, C, kc)$

A and B are real $m \times n$ matrices and C a real 2-dimensional array. MADD computes $A+B$ and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C contains at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

CALL SMADD(n, A, B, C)

A and B are real packed $n \times n$ symmetric matrices and C is a real array whose dimension is equal to or greater than $n(n+1)/2$. SMADD computes $A+B$, which is also a symmetric matrix, and stores the results in packed form in C.

The array C may begin in the same location as A or B. If C begins in the same location as A then the result C will overwrite the input data A. Similarly, B will be overwritten if C begins in the same location as B. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris.

CALL DMADD$(m, n, A, ka, B, kb, C, kc)$

A and B are double precision $m \times n$ matrices and C a double precision 2-dimensional array. DMADD computes $A + B$ and stores the results in C. The arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C contains at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

CALL CMADD$(m, n, A, ka, B, kb, C, kc)$

A and B are complex $m \times n$ matrices and C a complex 2-dimensional array. CMADD computes $A + B$ and stores the results in C. The arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C contains at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

MATRIX SUBTRACTION

The matrix difference $C = A - B$ can be computed by the following subroutines:

CALL MSUBT$(m, n, A, ka, B, kb, C, kc)$

A and B are real $m \times n$ matrices and C a real 2-dimensional array. MSUBT computes $A - B$ and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C contains at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

CALL SMSUBT(n, A, B, C)

A and B are real packed $n \times n$ symmetric matrices and C is a real array whose dimension is equal to or greater than $n(n+1)/2$. SMSUBT computes $A - B$, which is also a symmetric matrix, and stores the results in packed form in C.

The array C may begin in the same location as A or B. If C begins in the same location as A then the result C will overwrite the input data A. Similarly, B will be overwritten if C begins in the same location as B. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris.

CALL DMSUBT$(m, n, A, ka, B, kb, C, kc)$

A and B are double precision $m \times n$ matrices and C a double precision 2-dimensional array. DMSUBT computes $A - B$ and stores the results in C. The arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C contains at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

CALL CMSUBT$(m, n, A, ka, B, kb, C, kc)$

A and B are complex $m \times n$ matrices and C a complex 2-dimensional array. CMSUBT computes $A - B$ and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq m$, $kc \geq m$, and that C has at least n columns.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that $kc = ka$. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Programmer. A. H. Morris.

MATRIX MULTIPLICATION

The matrix product $C = AB$ may be computed by the subroutines MTMS, DMTMS, CMTMS or MPROD, DMPROD, CMPROD. MTMS, DMTMS, and CMTMS can be used if the storage area for C does not overlap with the storage areas for A and B. Otherwise, if the components of C are to be stored in the storage area for A or B, then MPROD, DMPROD, or CMPROD must be used.

CALL MTMS$(m, n, \ell, A, ka, B, kb, C, kc)$
CALL DMTMS$(m, n, \ell, A, ka, B, kb, C, kc)$
CALL CMTMS$(m, n, \ell, A, ka, B, kb, C, kc)$

MTMS is used if A and B are real matrices and C a real array, DMTMS is used if A and B are double precision matrices and C a double precision array, and CMTMS is used if A and B are complex matrices and C a complex array.

A is a matrix having m rows and n columns, B a matrix having n rows and ℓ columns, and C a 2-dimensional array. The routine computes the product AB and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq n$, $kc \geq m$, and that C contains at least ℓ columns.

Remark. It is assumed that the storage area for C is separate from the storage areas for A and B.

Programmer. A. H. Morris.

CALL MPROD$(m, n, \ell, A, ka, B, kb, C, kc, \mathrm{WK})$
CALL DMPROD$(m, n, \ell, A, ka, B, kb, C, kc, \mathrm{WK})$
CALL CMPROD$(m, n, \ell, A, ka, B, kb, C, kc, \mathrm{WK})$

MPROD is used if A and B are real matrices and C and WK are real arrays, DMPROD is used if A and B are double precision matrices and C and WK are double precision arrays, and CMPROD is used if A and B are complex matrices and C and WK are complex arrays.

A is a matrix having m rows and n columns, B a matrix having n rows and ℓ columns, and C a 2-dimensional array. The routine computes the product AB and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geq m$, $kb \geq n$, $kc \geq m$, and that C contains at least ℓ columns.

WK is an array that serves as a temporary storage area. The matrix C may begin in the same location as A or B. If C begins in the same location as A, then it is assumed that $kc = ka$ and that the dimension of WK is equal to or greater than ℓ. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that $kc = kb$ and that the dimension of WK is equal to or greater than m. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case, the array WK is not referenced.

Remark. If C begins in the same location as A or B, then it is assumed that the storage areas for A and B are distinct storage areas.

Programming. MPROD employs the subroutines RLOC and YCHG, DMPROD employs the subroutines DLOC and DYCHG, and CMPROD employs the subroutines CLOC and CYCHG. The routines were written by A. H. Morris.

PRODUCT OF A PACKED SYMMETRIC MATRIX AND A VECTOR

Let A denote a packed symmetric matrix of order n and x a column vector of dimension n where $n \geq 1$. Then the following subroutines are available for computing the product Ax.

CALL SVPRD(A, n, x, y)
CALL DSVPRD(A, n, x, y)

The argument y is an array of dimension n. SVPRD is called when A, x, y are real arrays and DSVPRD is called when A, x, y are double precision arrays. When either of these routines is called, Ax is computed and stored in y.

Programmer. A. H. Morris.

TRANSPOSE MATRIX PRODUCTS

If A^t denotes the transpose of A, then the matrix product $C = A^t B$ can be computed by the following subroutine:

CALL TMPROD$(m, n, \ell, A, ka, B, kb, C, kc)$

A is a real matrix having m rows and n columns, B a real matrix having m rows and ℓ columns, and C a real 2-dimensional array. TMPROD computes $A^t B$ and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

kc = the number of rows in the dimension statement for C in the calling program

Here it is assumed that $ka \geq m$, $kb \geq m$, $kc \geq n$, and that C contains at least ℓ columns. Also it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Note. All inner products $\sum_k a_{ki} b_{kj}$ are computed in double precision and the results stored in single precision.

Programmer. A. H. Morris.

SYMMETRIC MATRIX PRODUCTS

If A^t denotes the transpose of A, then the matrix product A^tA can be computed and its packed form stored in the array B by the following subroutine:

CALL SMPROD(m, n, A, ka, B)

A is a real $m \times n$ matrix and B a real array whose dimension is equal to or greater than $n(n+1)/2$. SMPROD computes A^tA and stores its packed form in B. The input argument ka has the following value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geq m$.

Note. All inner products $\sum_k a_{ki}a_{kj}$ are computed in double precision and the results stored in single precision.

Programmer. A. H. Morris.

KRONECKER PRODUCT OF MATRICES

If A is an $m \times n$ matrix and B a $k \times \ell$ matrix, then the ***Kronecker product*** $A \otimes B$ is defined by

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}.$$

From this definition we obtain:

(1) (Transpose Equality) $(A \otimes B)^t = A^t \otimes B^t$.
(2) $(A \otimes B) \otimes E = A \otimes (B \otimes E)$ for any matrix E.
(3) $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ if C is a matrix having n rows and D a matrix having ℓ rows.

If A and B are complex square matrices of orders m and k respectively, then from the Jordan canonical forms of A and B the determinant equality $\det(A \otimes B) = (\det A)^k (\det B)^m$ can be verified. Moreover, if A and B are nonsingular then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ from (3).

The following subroutines are available for computing $C = A \otimes B$.

CALL KPROD$(A, ka, m, n, B, kb, k, \ell, C, kc)$
CALL DKPROD$(A, ka, m, n, B, kb, k, \ell, C, kc)$
CALL CKPROD$(A, ka, m, n, B, kb, k, \ell, C, kc)$

It is assumed that A is an $m \times n$ matrix, B a $k \times \ell$ matrix, and C a 2-dimensional array. KPROD is used if A, B, C are real arrays, DKPROD is used if A, B, C are double precision arrays, and CKPROD is used if A, B, C are complex arrays. When the routine is called, $A \otimes B$ is computed and stored in C.

The arguments *ka*, *kb*, and *kc* have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that $ka \geq m$, $kb \geq k$, $kc \geq mk$, and that C contains at least $n\ell$ columns.

Remark. It is assumed that the array C does not overlap with A or B.

Programmer. A. H. Morris.

RANK OF A REAL MATRIX

The following subroutines are available for obtaining lower and upper bounds for the rank of real matrix.

CALL RNK(A, ka, m, n,RERR,AERR,k, k_0,WK,IWK)
CALL DRNK(A, ka, m, n,RERR,AERR,k, k_0,WK,IWK)

A and WK are real arrays and RERR and AERR real values if RNK is used. A and WK are double precision arrays and RERR and AERR double precision values if DRNK is used.

A is an $m \times n$ matrix where $m, n \geq 1$, and ka is the number of rows in the dimension statement for A in the calling program. It is required that $ka \geq m$. A is destroyed by the routine.

RERR is an argument which specifies that relative accuracy of the data in A. If it is estimated that the elements in A are accurate to μ significant digits then one may set RERR $= 10^{-\mu}$. It is required that RERR ≥ 0. If RERR $= 0$ then it is assumed that the elements in A are accurate to machine precision.

AERR is an argument which specifies the maximum absolute uncertainty of the data in A. For example, if it is estimated that the elements a_{ij} of A have the relative accuracy RERR except when $|a_{ij}| < 10^{-30}$, then one may set AERR $= 10^{-30}$. It is required that AERR ≥ 0.

The arguments k and k_0 are variables. When RNK or DRNK is called, the rank of A is bounded from above and below using the tolerances RERR and AERR. The variable k is set to the upper bound and k_0 to the lower bound.

WK is an array of dimension $5 \cdot \min\{m, n\}$ or larger, and IWK an array of dimension $m + n$ or larger. WK and IWK are work spaces for the routines.

Remarks. Normally $k_0 = k$, in which case k is the rank of A. However, if $k_0 \neq k$ then it is recommended that the least squares routine HFTI2 or DHFT2 be used for obtaining the most appropriate value for the rank. When HFTI2 or DHFT2 is used, B may be selected to be the $m \times m$ identity matrix, and the size of the elements of the solution matrix X should be considered (in addition to the values of the residual norms). Frequently, the lower bound k_0 will be found to be the most appropriate value for the rank.

Programming. RNK employs the subroutines U11LS, U11US, ISWAP, SSWAP, SAXPY, SSCAL and functions SDOT, SNRM2, ISAMAX, SPMPAR, IPMPAR. DRNK employs the subroutines DU11LS, DU11US, ISWAP, DSWAP, DAXPY, DSCAL and functions DDOT, DNRM2, IDAMAX, DPMPAR, IPMPAR. RNK and DRNK were written by A. H. Morris, and U11LS, U11US, DU11LS, DU11US, were written by T. Manteuffel (Los Alamos).

Reference. Manteuffel, T., ***An Interval Analysis Approach to Rank Determination***

in Linear Least Squares Problems, Report SAND 80–0655, Sandia Laboratories, Albuquerque, New Mexico, 1980.

INVERTING GENERAL REAL MATRICES AND SOLVING GENERAL SYSTEMS OF REAL LINEAR EQUATIONS

The subroutines CROUT, KROUT, MSLV, MSLV1, NPIVOT, DMSLV, and DMSLV1 are available for inverting real matrices A and solving systems of real linear equations $AX = B$. CROUT, KROUT, MSLV, MSLV1, and NPIVOT solve single precision problems, and DMSLV and DMSLV1 solve double precision problems.

All the routines except NPIVOT are general-purpose, employing partial pivot Gauss elimination. NPIVOT can only occasionally be used since it uses Gauss-Jordan elimination with no pivot search. Normally CROUT and KROUT produce the same results, and MSLV and MSLV1 produce the same results. Since many of the calculations are performed in double precision in CROUT and KROUT, these subroutines will be slower than MSLV and MSLV1, but they may be more accurate.

CALL CROUT(MO, n, m, A, ka, B, kb, D, INDEX,TEMP)

A is a real matrix of order n where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a real matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

D is a real variable. When CROUT is called, D is assigned the value $\det(A)$ where $\det(A)$ is the determinant of A. If D is found to have the value 0 then the routine immediately terminates.

INDEX is an array of dimension $n - 1$ or larger that is used by the routine for keeping track of the row interchanges that are made. If MO $\neq 0$ then INDEX is ignored.

TEMP is a real array of dimension n or larger that is a work space for the routine. If MO $\neq 0$ then TEMP is ignored.

Remarks.

(1) KROUT should be used instead of CROUT when the determinant D is not needed. Underflow or overflow in the calculation of D may cause CROUT to terminate prematurely.

(2) The matrix A is destroyed.

Algorithm. The Crout procedure is used. All inner products are computed in double precision and the results returned in single precision. Partial pivoting is performed.

Programming. CROUT calls the subroutine CROUT0. These routines were written by A. H. Morris.

CALL KROUT(MO, n, m, A, ka, B, kb,IERR,INDEX,TEMP)

A is a real matrix of order n where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a real matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

INDEX is an array of dimension $n - 1$ or larger that is used by the routine for keeping track of the row interchanges that are made. If MO $\neq 0$ then INDEX is ignored.

TEMP is a real array of dimension n or larger that is a work space for the routine. If MO $\neq 0$ then TEMP is ignored.

Error Return. IERR is a variable that reports the status of the results. When the routine terminates IERR has one of the following values:

IERR $=$ 0 The requested results were obtained.
IERR $= -1$ Either n, ka, or kb is incorrect. In this case, A and B have not been modified.
IERR $= k$ The k^{th} column of A has been reduced to a column containing only zeros.

When an error is detected, the routine immediately terminates.

Remark. The matrix A is destroyed.

Algorithm. The CROUT procedure is used. All inner products are computed in double precision and the results returned in single precision. Partial pivoting is performed.

Programming. KROUT calls the subroutine KROUT0. These routines were written by A. H. Morris.

CALL NPIVOT(n, m, A, ka, B, kb, D,IERR)

A is a real matrix of order n where $n \geq 1$. When NPIVOT is called the inverse of A is computed and stored in A.

The argument m is an integer. If $m \geq 1$ then B is a real matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then kb is ignored.

D is a real variable. On input D must be assigned a value by the user. If the input value is r, then when NPIVOT terminates $D = rd$ where d is the determinant of A.

Error Return IERR is an integer variable. If inversion is successful then IERR is assigned the value 0. Otherwise, IERR = 1 if NPIVOT cannot invert the matrix.

Algorithm. The Gauss-Jordan procedure is used. However, no pivot searching is done. NPIVOT terminates (with IERR set to 1) whenever a zero pivot element is encountered.

Remarks. Since pivot search is frequently needed to invert a matrix, and since pivot search is normally required to obtain accurate results, ***NPIVOT should not be used except on those occasions when pivot search is known to be superfluous.***

Programmer. A. H. Morris.

CALL MSLV(MO, n, m, A, ka, B, kb, D, RCOND,IERR,IWK,WK)
CALL DMSLV(MO, n, m, A, ka, B, kb, D, RCOND,IERR,IWK,WK)

A is a matrix of order n where $n \geq 1$. If MO = 0 then the inverse of A is computed and stored in A. If MO $\neq$ 0 then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

D is an array of dimension 2. When either routine is called the determinant $\det(A)$ of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leq |d| < 10$ and k an integer, then d is stored in $D(1)$ and the exponent k is stored in floating point form in $D(2)$.

RCOND is a variable. When either routine is called, the routine makes an estimate c of the condition number of the matrix A (relative to the L_1 norm). RCOND is assigned the value $1/c$.

IWK is an array of dimension n or larger that is used by the routines for keeping track

of the row interchanges that are made. WK is an array of dimension n or larger that is used as a work space.

Remarks.

(1) If MSLV is called then it is assumed that A and B are real matrices, D and WK real arrays, and RCOND a real variable. Otherwise, if DMSLV is called then it is assumed that A and B are double precision matrices, D and WK double precision arrays, and RCOND a double precision variable.

(2) RCOND satisfies $0 \leq \text{RCOND} \leq 1$. If RCOND $\approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if MSLV is used to invert a matrix in the 14 digit CDC single precision arithmetic and RCOND = .4E–3, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If RCOND ≈ 1 then one should expect the results to be almost as accurate as the original data A. However, if RCOND ≈ 0 then one should expect the results to be nonsense.

(3) The matrix A is destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that $1 +$ RCOND > 1, then IERR is set to 0 and the problem is solved. Otherwise, if $1+$RCOND $= 1$ then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant will not have been computed.

Algorithm. The partial pivot Gauss elimination procedure is used.

Programming. MSLV and DMSLV are driver routines for the LINPACK subroutines SGECO, SGEFA, SGESL, SGEDI and DGECO, DGEFA, DGESL, DGEDI. The subroutines were coded by Cleve Moler (University of New Mexico). The subroutines employ the vector routines SSWAP, SDOT, SSCAL, SAXPY, SASUM, ISAMAX and DSWAP, DDOT, DSCAL, DAXPY, DASUM, IDAMAX.

References.

(1) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

(2) Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Journal of Numerical Analysis*** **16** (1979), pp. 368–375.

CALL MSLV1(MO,n,m,A,ka,B,kb,IERR,IWK,WK)
CALL DMSLV1(MO,n,m,A,ka,B,kb,IERR,IWK,WK)

A is a matrix of order n where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m > 1$ then B is a matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m < 0$ then there are no equations to be solved. In this case the

argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

IWK is an array of dimension n or larger that is used by the routines for keeping track of the row interchanges that are made. WK is an array of dimension n or larger that is used for a work space. If MO $\neq 0$ then WK is ignored.

Remarks.

(1) If MSLV1 is called then it is assumed that A and B are real matrices and WK a real array. Otherwise, if DMSLV1 is called then it is assumed that A and B are double precision matrices and WK a double precision array.
(2) The matrix A is destroyed.

Error Return. IERR is a variable that reports the status of the results. When the routine terminates IERR has one of the following values:

IERR = 0 The requested results were obtained.
IERR = -1 Either n, ka, or kb is incorrect. In this case, A and B have not been modified.
IERR = k The k^{th} column of A has been reduced to a column containing only zeros.

Algorithm. The partial pivot Gauss elimination procedure is used.

Programming. MSLV1 and DMSLV1 are driver routines for the LINPACK subroutines SGEFA, SGESL, SGEDI and DGEFA, DGESL, DGEDI. The subroutines were coded by Cleve Moler (University of New Mexico). The subroutines employ the vector routines SSWAP, SDOT, SSCAL, SAXPY, SASUM, ISAMAX and DSWAP, DDOT, DSCAL, DAXPY, DASUM, IDAMAX.

References.

(1) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.
(2) Cline, A. K., Moler, C. B., Stewart, G W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Journal of Numerical Analysis*** **16** (1979), pp. 368–375.

SOLUTION OF REAL EQUATIONS WITH ITERATIVE IMPROVEMENT

Given a real $n \times n$ matrix A and column vector b. The following subroutine is available for solving the equation $Ax = b$. Iterative improvement is performed to compute the solution x to machine accuracy.

CALL SLVMP(MO,n, A, ka, b, X,WK,IWK,IND)

MO is an input argument which specifies if SLVMP is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional real array of dimension $ka \times n$ containing the matrix A, b a real vector of dimension n, and X a real array of dimension n. When SLVMP is called, $Ax = b$ is solved and the solution stored in X. A and b are not modified by the routine.

WK is a real array of dimension n^2+n or larger, and IWK an integer array of dimension n or larger. These arrays are for internal use by the routine. On an initial call to SLVMP, an LU decomposition is obtained for A and stored in WK and IWK. Then the equation $Ax = b$ is solved.

IND is an integer variable that reports the status of the results. On an initial call to SLVMP, when the routine terminates IND has one of the following values:

IND = 0 The solution X was obtained to machine accuracy.
IND = 1 X was obtained, but not to machine accuracy.
IND = $-k$ The k^{th} column of A was reduced to a column containing only zeros.

When IND = $-k$, no solution is obtained.

After an initial call to SLVMP, if IND = 0 or 1 on output, then the routine may be called to solve a new set of equations $Ax = \tilde{b}$ without having to redecompose the matrix A. In this case, the input argument MO may be set to any nonzero value. When MO $\neq 0$ it is assumed that only b has been modified. The routine employs the LU decomposition obtained on the initial call to SLVMP to solve the new set of equations $Ax = b$. On output X will contain the solution to the new set of equations. As before, A and b are not modified by the routine.

If SLVMP is recalled with MO $\neq 0$, then when the routine terminates IND has one of the following values:

IND = 0 The solution X was obtained to machine accuracy.
IND = 1 X was obtained, but not to machine accuracy.

Programming. SLVMP calls the subroutine LUIMP. These routines were written by A. H. Morris. The subroutines MCOPY, SGEFA, SGESL, SSCAL, SAXPY and functions SPMPAR, SDOT, ISAMAX are also employed.

SOLUTION OF ALMOST BLOCK DIAGONAL SYSTEMS OF LINEAR EQUATIONS

Consider a system $Ax = b$ of linear equations where A is an $n \times n$ matrix having the block structure

$$A = \begin{pmatrix} \boxed{A_1} & & & 0 \\ & \boxed{A_2} & & \\ & & \boxed{A_3} & \\ & & & \ddots \\ 0 & & & \boxed{A_m} \end{pmatrix}$$

Here it is assumed that A_i is an $r_i \times c_i$ matrix for $i = 1, \ldots, m$, and that A_i and A_{i+1} may have $\delta_i \geq 0$ columns in common for $i < m$. Thus $\sum_{i=1}^{m} r_i = n$ and block A_i begins in column $\sum_{k=1}^{i-1} (c_k - \delta_k) + 1$ for $i \geq 2$. It is also assumed that three successive blocks A_{i-1}, A_i, A_{i+1} do not have columns in common. Thus $\delta_{i-1} + \delta_i \leq c_i$ for $i = 2, \ldots, m-1$. If $m \geq 2$ then the following subroutines are available for solving $Ax = b$.

CALL ARCECO(n, S,MTR,m,IWK,b, X,IND)

S is an array of dimension $\sum_{i=1}^{m} r_i c_i$ or larger. On input S contains the blocks $A_1, \ldots, A_m$ of the matrix A. The blocks are stored in sequence. A_1 is stored in the first $r_1 c_1$ locations of S, A_2 is stored in the next $r_2 c_2$ locations, etc. For each A_i, the columns of A_i are stored in sequence in S.

Example.

$$\text{If} \quad A = \begin{pmatrix} \boxed{\begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{matrix}} & 0 & 0 \\ 0 & \boxed{\begin{matrix} a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{matrix}} & 0 \\ 0 & 0 & \boxed{a_{55}} \end{pmatrix}$$

then $n = 5$, $m = 3$, $\delta_1 = 2$, and $\delta_2 = 0$. Also, S is an array containing the data $a_{11}, a_{21}, a_{12}, a_{22}, a_{13}, a_{23}, a_{32}, a_{42}, a_{33}, a_{43}, a_{34}, a_{44}, a_{55}$.

MTR is an integer matrix of dimension $3 \times m$ containing the block information of the matrix A:

$\text{MTR}(1, i) = r_i \qquad (i = 1, \ldots, m)$
$\text{MTR}(2, i) = c_i \qquad (i = 1, \ldots, m)$
$\text{MTR}(3, i) = \delta_i \qquad (i = 1, \ldots, m-1)$

For convenience, the routine sets $\text{MTR}(3, m) = 0$.

X is an array of dimension n or larger. When ARCECO is called, A is decomposed and then the equations $Ax = b$ are solved. The decomposition of A is stored in S, overwriting the initial input data A, and the solution x is stored in X. The vector b is destroyed by the routine.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the decomposition of A are stored in IWK.

IND is a variable that reports the status of the results. When ARCECO terminates, IND has one of the following values:

IND = 0 The system of equations was solved.
IND = 1 (Input Error) Either n, m, or MTR is incorrect, or three successive blocks A_{i-1}, A_i, A_{i+1} of A have columns in common.
IND = −1 A is a singular matrix. The equations cannot be solved.

Usage. After a call to ARCECO, if IND = 0 on output then the subroutine ARCESL (see below) may be called to solve a new set of equations $Ax = \tilde{b}$ without having to redecompose the matrix A. ARCESL employs the decomposition of A obtained by ARCECO.

Algorithm. A modification of the alternate row and column elimination procedure by Varah is used.

Programming. ARCECO employs the routines ARCEDC, ARCEPR, ARCEPC, ARCESL, ARCEFS, ARCEFM, ARCEFE, ARCEBS, ARCEBM, and ARCEBE. These routines were developed by J. C. Diaz (Mobil Research and Development Corp., Farmers Branch, Texas), G. Fairweather (University of Kentucky), and P. Keast (University of Toronto).

Reference. Diaz, J. C., Fairweather, G., and Keast, P., "FORTRAN Packages for Solving Certain Almost Block Diagonal Linear Systems by Modified Alternate Row and Column Elimination," ***ACM Trans. Math Software*** **9** (1983), pp. 358–375.

CALL ARCESL(S,MTR,m,IWK,b,X)

The argument m is the number of blocks $A_1, \ldots, A_m$ in the matrix A. ARCESL may be used only when IND = 0 on output from ARCECO. In this case, S contains the decomposition of A obtained by ARCECO and IWK contains the pivot indices of the decomposition. The argument b is a vector of dimension n, and X an array of dimension n or larger. When ARCESL is called, the equations $Ax = b$ are solved and the solution stored in X. The vector b is destroyed by the routine.

Programming. ARCESL calls the subroutines ARCEFS, ARCEFM, ARCEFE, ARCEBS, ARCEBM, and ARCEBE. These routines were developed by J. C. Diaz (Mobil Research and Development Corp.,Farmers Branch, Texas), G. Fairweather (University of Kentucky), and P. Keast (University of Toronto).

SOLUTION OF ALMOST BLOCK TRIDIAGONAL SYSTEMS OF LINEAR EQUATIONS

Consider a system $Tx = b$ of linear equations where T is a square matrix having the block structure

$$T = \begin{pmatrix} A_1 & B_1 & C_1 & & \\ C_2 & A_2 & B_2 & & 0 \\ & C_3 & A_3 & B_3 & \\ 0 & \ddots & \ddots & \ddots & \\ & & C_{n-1} & A_{n-1} & B_{n-1} \\ & & B_n & C_n & A_n \end{pmatrix}.$$

Here it is assumed that $A_k, B_k, C_k \quad (k = 1, \ldots, n)$ are $m \times m$ matrices, and that b is a column vector of dimension mn. If $n \geq 4$ then the following subroutine is available for solving $Tx = b$.

CALL BTSLV(MO,m, n, A, B, C, X,IP,IND)

MO is an input argument which specifies if BTSLV is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A, B, C are 3-dimensional arrays of dimension $m \times m \times n$ where the (i,j)-th elements of the matrices A_k, B_k, C_k are stored in $A(i,j,k)$, $B(i,j,k)$, $C(i,j,k)$ for $k = 1, \ldots, n$. A, B, C are modified by the routine.

X is an array of dimension mn or larger. On input, the vector b is stored in X. When BTSLV is called, if a solution x is obtained for $Tx = b$ then the solution is stored in X.

IP is an array of dimension mn or larger that is used by the routine for listing the row interchanges that are made.

On an initial call to the routine, a block LU decomposition is performed on T, the results of which are stored in A, B, C. This decomposition involves row interchanges only within the diagonal blocks A_k; i.e., no row interchanges are performed between rows of different blocks A_k and A_ℓ. Thus it may occur that the decomposition of a nonsingular matrix T cannot be completed. IND is a variable that reports the status of the results. When BTSLV terminates, IND has one of the following values:

IND = 0 T was decomposed and the equations $Tx = b$ solved.
IND = −1 (Input Error) Either $m < 1$ or $n < 4$.
IND = k The decomposition process failed in the k^{th} diagonal block. The routine cannot solve the equations in their present form.

After an initial call to BTSLV, if IND = 0 then the routine may be recalled with MO $\neq 0$ and a new b. When MO $\neq 0$, then it is assumed that A, B, C, and IP have not been modified and that X contains the new b. The routine retrieves from A, B, C, and IP the block decomposition that was obtained on the initial call to BTSLV, and solves the new

system of equations $Tx = b$. The solution is stored in X. In this case, IND is not referenced by the routine.

Programming. BTSLV employs the subroutines DECBT, SOLBT, DEC, and SOL. These subroutines were written by Alan C. Hindmarsh (Lawrence Livermore Laboratory).

Reference. Hindmarsh, A. C., ***Solution of Block-Tridiagonal Systems of Linear Algebraic Equations***, Report UCID-30150, Lawrence Livermore Laboratory, 1977.

INVERTING SYMMETRIC REAL MATRICES AND SOLVING SYMMETRIC SYSTEMS OF REAL LINEAR EQUATIONS

The subroutines SMSLV and DSMSLV are available for inverting symmetric real matrices A and solving systems of real linear equations $AX = B$. SMSLV handles single precision problems and DSMSLV handles double precision problems. It is assumed that the matrix A is in packed form. If the inverse of A is computed, then the inverse is a symmetric matrix which will be stored in packed form.

Note. All eigenvalues of a real symmetric matrix A are real. The ***inertia*** of A is the ordered triple (π, ν, ς) where π is the number of positive eigenvalues of A, ν the number of negative eigenvalues of A, and ς the number of zero eigenvalues of A. Thus, if n is the order of A then $\pi + \nu + \varsigma = n$. Also A is positive definite (positive semi-definite, negative definite, negative semi-definite) if $\pi = n$ ($\nu = 0, \nu = n, \pi = 0$).

CALL SMSLV(MO,n,m,A,B,kb,D,RCOND,INERT,IERR,IPVT,WK)
CALL DSMSLV(MO,n,m,A,B,kb,D,RCOND,INERT,IERR,IPVT,WK)

A is an array of dimension $n(n+1)/2$ containing the elements of a packed $n \times n$ symmetric matrix where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A in packed form. If MO $\neq 0$ then the inverse of A is not computed.

The argument m is an integer. If $m \geq 1$ then B is a matrix having n rows and m columns. In this case $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case B is ignored.

The argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then kb is ignored.

D is an array of dimension 2. When either routine is called the determinant $\det(A)$ of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leq |d| < 10$ and k an integer, then d is stored in $D(1)$ and the exponent k is stored in floating point form in $D(2)$.

RCOND is a variable. When either routine is called, the routine makes an estimate c of the condition number of the matrix A (relative to the L_1 norm). RCOND is assigned the value $1/c$.

INERT is an integer array of dimension 3. When either routine is called the inertia of the matrix A is computed. INERT(1) is set to the number of positive eigenvalues of A, INERT(2) is set to the number of negative eigenvalues and INERT(3) is set to the number of zero eigenvalues.

IPVT is an integer array of dimension n or larger that is used by the routines for keeping track of the row and column interchanges that are made. WK is array of dimension n or larger that is used as a work space.

Remarks.

(1) If SMSLV is called then it is assumed that A and B are real matrices, D and WK are real arrays, and RCOND is a real variable. Otherwise, if DSMSLV is called then it is assumed that A and B are double precision matrices, D and WK are double precision arrays, and RCOND is a double precision variable.

(2) RCOND satisfies $0 \leq \text{RCOND} \leq 1$. If RCOND $\approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if SMSLV is used to invert a matrix in the 14 digit CDC single precision arithmetic and RCOND = .4E−3, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If RCOND ≈ 1 then one should expect the results to be almost as accurate as the original data A. However, if RCOND ≈ 0 then one should expect the results to be nonsense.

(3) The data in A is destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that 1 + RCOND > 1, then IERR is set to 0 and the problem is solved. Otherwise, if 1 + RCOND = 1 then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant and inertia will not have been computed.

Algorithm. The diagonal pivoting factorization procedure is used. Partial pivoting is employed.

Precision. SMSLV and the more general routine MSLV have approximately the same accuracy, and DSMSLV and DMSLV have approximately the same accuracy.

Efficiency. Even though SMSLV performs approximately half the number of multiplications and divisions as MSLV, normally one can expect SMSLV to take about 70-80% of the time required by MSLV. However, for sparse matrices SMSLV may be 20-30% slower than MSLV. Similar comments hold for DSMSLV and DMSLV.

Programming. SMSLV and DSMSLV are driver routines for the LINPACK subroutines SSPCO, SSPFA, SSPSL, SSPDI and DSPCO, DSPFA, DSPSL, DSPDI. SSPCO and DSPCO were written by Cleve Moler (University of New Mexico). The remaining LINPACK subroutines were written by James Bunch (University of California, San Diego). The subroutines employ the vector routines SCOPY, SSWAP, SDOT, SSCAL, SAXPY, SASUM, ISAMAX and DSWAP, DDOT, DSCAL, DAXPY, DASUM, IDAMAX.

References.

(1) Bunch, J. R. and Parlett, B. N., "Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations," *SIAM J. Numerical Analysis* **8** (1971), pp. 639–655.

(2) Bunch, J. R., "Analysis of the Diagonal Pivoting Method," *SIAM J. Numerical Analysis* **8** (1971), pp. 656–680.

(3) Bunch, J. R., Kaufman, L., and Parlett, B.N., "Decomposition of a Symmetric Matrix," ***Numerische Mathematik*** **27** (1976), pp. 95–109.
(4) Bunch, J., and Kaufman, L., "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems," ***Math. Comp.*** **31** (1977), pp. 163–179.
(5) Cline, A. K., Moler, C. B., Stewart, G. W , and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Numerical Analysis*** **16** (1979), pp. 368–375.
(6) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

INVERTING POSITIVE DEFINITE SYMMETRIC MATRICES AND SOLVING POSITIVE DEFINITE SYMMETRIC SYSTEMS OF LINEAR EQUATIONS

The subroutines PCHOL and DPCHOL are available for inverting positive definite symmetric real matrices A and solving systems of real linear equations $AX = B$. PCHOL handles single precision problems and DPCHOL handles double precision problems. It is assumed that the matrix A is in packed form. If the inverse of A is computed then the inverse is a symmetric matrix which will be stored in packed form.

CALL PCHOL(MO,n, m, A, B,kb,IERR)
CALL DPCHOL(MO,n, m, A, B,kb,IERR)

A is an array of dimension $n(n+1)/2$ or larger containing the elements of a packed $n \times n$ positive definite symmetric matrix where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A in packed form. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a matrix having n rows and m columns. In this case $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case B is ignored.

The argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then kb is ignored.

Remarks.

(1) If PCHOL is called then it is assumed that A and B are real arrays, and if DPCHOL is called then it is assumed that A and B are double precision arrays.
(2) The data in A is destroyed.

Error Return. IERR is an integer variable. If A is positive definite then IERR is set to 0 and the problem is solved. Otherwise, IERR $= k$ if the leading $k \times k$ submatrix of A is not positive definite.

Algorithm. The Cholesky procedure is used.

Precision. The results obtained by PCHOL and DPCHOL are occasionally less accurate (up to 1 significant digit) than the results obtained by SMSLV and DSMSLV.

Programming. PCHOL and DPCHOL are driver routines for the LINPACK subroutines SPPFA, SPPSL, SPPDI and DPPFA, DPPSL, DPPDI. These subroutines were written by Cleve Moler (University of New Mexico). The functions SDOT, DDOT and subroutines SAXPY, SSCAL, DAXPY, DSCAL are also used.

Reference. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

SOLUTION OF TOEPLITZ SYSTEMS OF LINEAR EQUATIONS

An $n \times n$ Toeplitz matrix is a matrix of the form

$$\begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \cdots & a_{-n+2} \\ a_2 & a_1 & a_0 & \cdots & a_{-n+3} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{pmatrix}.$$

For convenience, we denote this matrix by A_n for $n \geq 1$. If A_n is a real matrix where $a_0 \neq 0$ and b is a real column vector, then the subroutines TOPLX and DTOPLX are available for solving the system of equations $A_n x = b$. TOPLX is a single precision routine and DTOPLX a double precision routine.

CALL TOPLX$(A, b, x, n, G, H,$IERR)
CALL DTOPLX$(A, b, x, n, G, H,$IERR)

TOPLX is used if A, b, x, G, H are real arrays, and DTOPLX is used if A, b, x, G, H are double precision arrays.

A is an array of dimension $2n-1$ or larger containing the coefficients $A(j) = a_{j-n}$ $(j = 1, \ldots, 2n-1)$ of the matrix A_n. The argument x is an array of dimension n or larger, and IERR is an integer variable. When the routine is called, if $A_n x = b$ is solved then IERR is set to 0 and the solution is stored in x. A and b are not modified by the routine.

G and H are arrays of dimension n or larger that are work spaces for the routine.

Error Return. IERR = 1 if the equations cannot be solved by the routine.

Remarks. The Levinson bordering procedure is used. This procedure is inductive, beginning with the solution $x_1 = b_1/a_0$ of the equation $a_0 x_1 = b_1$. Given a solution for $\Sigma_{j=1}^{N} a_{i-j} x_j = b_i$ $(i = 1, \ldots, N)$ where $N < n$, then a solution for $\Sigma_{j=1}^{N+1} a_{i-j} \bar{x}_j = b_i$ $(i = 1, \ldots, N+1)$ is obtained where $\bar{x}_1, \ldots, \bar{x}_{N+1}$ is computed from $x_1, \ldots, x_N$. This procedure fails when some submatrix A_N is singular (e.g., when $a_0 = 0$). Also, the procedure may yield poor results when some A_N is exceedingly ill-conditioned. In such situations one must use a more general equation solver. In TOPLX and DTOPLX $4(n-1)^2$ floating additions and $4n(n-1)$ integer/floating multiplications and divisions are used when $n \geq 2$. Consequently, these routines are considerably more efficient than general equation solvers such as KROUT and DMSLV, but more restrictive and frequently less accurate.

Programming. TOPLX and DTOPLX are modifications by A. H. Morris of the subroutine TOEPLZ, written by George Rybicki.

Reference. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., ***Numerical Recipes: The Art of Scientific Computing***, Cambridge University Press, 1986, pp. 47–52.

INVERTING GENERAL COMPLEX MATRICES AND SOLVING GENERAL SYSTEMS OF COMPLEX LINEAR EQUATIONS

The subroutines CMSLV, CMSLV1, and DCMSLV are available for inverting complex matrices and solving systems of complex linear equations. CMSLV and CMSLV1 solve single precision problems and DCMSLV solves double precision problems.

CALL CMSLV(MO,n, m, A, ka, B, kb, D,RCOND,IERR,IPVT,WK)

A is a complex matrix of order n where $n \geq 1$. If MO $= 0$ then the inverse of A is computed and stored in A. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a complex matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and the argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

D is a complex array of dimension 2. When CMSLV is called the determinant $\det(A)$ of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leq |\mathrm{Re}(d)| + |\mathrm{Im}(d)| < 10$ and k an integer, then d is stored in $D(1)$ and the exponent k is stored as a complex number in $D(2)$.

RCOND is a real variable. When CMSLV is called, the routine makes an estimate c of the condition number of the matrix A (relative to the modified L_1 norm where each absolute value $|z|$ is replaced with $|\mathrm{Re}(z)| + |\mathrm{Im}(z)|$). RCOND is assigned the value $1/c$.

IPVT is an integer array of dimension n or larger that is used by the routine for keeping track of the row interchanges that are made. WK is a complex array of dimension n or larger that is used as a work space.

Remarks.

(1) RCOND satisfies $0 \leq$ RCOND ≤ 1. If RCOND $\approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if CMSLV is used to invert a matrix in the 14 digit CDC single precision arithmetic and RCOND $= 4E-3$, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If RCOND ≈ 1 then one should expect the results to be almost as accurate as the original data A. However, if RCOND ≈ 0 then one should expect the results to be nonsense.

(2) The matrix A is always destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that $1 +$

RCOND > 1, then IERR is set to 0 and the problem is solved. Otherwise, if 1+RCOND = 1 then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant will not have been computed.

Algorithm. The partial pivot Gauss elimination procedure is used. The pivots a_{kj} are selected so that $|\mathrm{Re}(a_{kj})| + |\mathrm{Im}(a_{kj})| = \max\{|\mathrm{Re}(a_{ij})| + |\mathrm{Im}(a_{ij})| : i = j, \ldots, n\}$.

Programming. CMSLV calls the LINPACK subroutines CGECO, CGEFA, CGESL, and CGEDI. These subroutines were written by Cleve Moler (University of New Mexico). The subroutines CSWAP, CSCAL, CSSCAL, CAXPY and functions CDOTC, SCASUM, ICAMAX are also used.

References.

(1) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

(2) Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Journal of Numerical Analysis*** **16** (1979), pp. 368–375.

CALL CMSLV1(MO,n, m, A, ka, B, kb,IERR,IPVT,WK)

A is a complex matrix of order n where $n \geq 1$. If MO = 0 then the inverse of A is computed and stored in A. If MO $\neq$ 0 then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then B is a complex matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $n \leq 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and the argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leq 0$ then the argument kb is ignored.

IPVT is an integer array of dimension n or larger that is used by the routines for keeping track of the row interchanges that are made.

WK is a complex array of dimension n or larger that is a work space for the routine. If MO $\neq$ 0 then WK is ignored.

Error Return. IERR is a variable that reports the status of the results. When CMSLV1 terminates IERR has one of the following values:

IERR = 0 The requested results were obtained.

IERR = 1 Either n, ka, or kb is incorrect. In this case, A and B have not been modified.

IERR = k The k^{th} column of A has been reduced to a column containing zeros. The requested results cannot be obtained.

Remarks.

(1) The matrix A is destroyed.
(2) CMSLV and CMSLV1 produce the same results for X and the inverse of A.

Algorithm. The partial pivot Gauss elimination procedure is used. The pivots a_{kj} are selected so that $|\mathrm{Re}(a_{kj})| + |\mathrm{Im}(a_{kj})| = \max\{|\mathrm{Re}(a_{ij})| + |\mathrm{Im}(a_{ij})| : i = j, \ldots, n\}$.

Programming. CMSLV1 calls the LINPACK subroutines CGEFA, CGESL, and CGEDI. These subroutines were written by Cleve Moler (University of New Mexico). The subroutines CSWAP, CSCAL, CAXPY and functions CDOTC, ICAMAX are also used.

Reference. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, SIAM, 1979.

CALL DCMSLV(MO,n,m,AR,AI,ka,BR,BI,kb,IERR,IPVT,WK)

AR and AI are double precision matrices of order $n \geq 1$. AR and AI are the real and imaginary parts of the complex matrix A whose inverse is to be computed or for which $AX = B$ is to be solved. If MO $= 0$ then the inverse of the complex matrix is computed and the results stored in AR and AI. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geq 1$ then BR and BI are double precision matrices having n rows and m columns. In this case, BR and BI are the real and imaginary parts of the complex matrix B for which $AX = B$ is to be solved. When DCMSLV is called, the real and imaginary parts of the solution X are computed and stored in BR and BI. If $m \leq 0$ then there are no equations to be solved. In this case BR and BI are ignored.

The argument ka is the number of rows in the dimension statements for AR and AI in the calling program, and the argument kb is the number of rows in the dimension statements for BR and BI in the calling program. If $m \leq 0$ then the argument kb is ignored.

IPVT is an integer array of dimension n or larger that is used by the routine for keeping track of the row interchanges that are made.

WK is a double precision array of dimension $2n$ or larger that is a work space for the routine. If MO $\neq 0$ then WK is ignored.

Error Return. IERR is a variable that reports the status of the results. When DCMSLV terminates IERR has one of the following values:

IERR $= 0$	The requested results were obtained.
IERR $= 1$	Either n, ka, or kb is incorrect. In this case, AR, AI, BR, BI have not been modified.
IERR $= k$	The k^{th} columns of AR and AI have been reduced to columns containing zeros. The requested results were not obtained.

Remark. The matrices AR and AI are destroyed.

Algorithm. The partial pivot Gauss elimination procedure is used. The pivots A_{kj} are selected so that $|\mathrm{Re}(a_{kj})| + |\mathrm{Im}(a_{kj})| = \max\{|\mathrm{Re}(a_{ij})| + |\mathrm{Im}(a_{ij})| : i = j, \ldots, n\}$.

Programming. DCMSLV calls the subroutines DCFACT, DCSOL, and DCMINV. These routines were written by A. H. Morris. The functions CDIVID and DPMPAR are also used.

SOLUTION OF COMPLEX EQUATIONS WITH ITERATIVE IMPROVEMENT

Given a complex $n \times n$ matrix A and a complex column vector b. The following routine is available for solving the equation $Ax = b$. Iterative improvement is performed to compute the solution x to machine accuracy.

CALL CSLVMP(MO,n, A, ka, b, X,WK,IWK,IND)

MO is an input argument which specifies if CSLVMP is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional complex array of dimension $ka \times n$ containing the matrix A, b a complex vector of dimension n, and X a complex array of dimension n. When CSLVMP is called, $Ax = b$ is solved and the solution stored in X. A and b are not modified by the routine.

WK is a complex array of dimension $n^2 + n$ or larger, and IWK an integer array of dimension n or larger. These arrays are for internal use by the routine. On an initial call to CSLVMP, an LU decomposition is obtained for A and stored in WK and IWK. Then the equation $Ax = b$ is solved.

IND is an integer variable that reports the status of the results. On an initial call to CSLVMP, when the routine terminates IND has one of the following values:

IND = 0 The solution X was obtained to machine accuracy.
IND = 1 X was obtained, but not to machine accuracy.
IND = $-k$ The k^{th} column of A was reduced to a column containing only zeros. In this case no solution can be obtained.

After an initial call to CSLVMP, if IND = 0 or 1 on output, then the routine may be called to solve a new set of equations $Ax = \tilde{b}$ without having to redecompose the matrix A. In this case, the input argument MO may be set to any nonzero value. When MO $\neq 0$ it is assumed that only b has been modified. The routine employs the LU decomposition obtained on the initial call to CSLVMP to solve the new set of equations $Ax = b$. On output X will contain the solution to the new set of equations. As before, A and b are not modified by the routine.

If CSLVMP is recalled with MO $\neq 0$, then when the routine terminates IND has one of the following values:

IND = 0 The solution X was obtained to machine accuracy.
IND = 1 X was obtained, but not to machine accuracy.

Programming CSLVMP calls the subroutine CLUIMI. These routines were written by A. H. Morris. The subroutines CMCOPY, CGEFA, CGESL, CSCAL, CAXPY and functions SPMPAR, CDOTC, ICAMAX are also employed.

SINGULAR VALUE DECOMPOSITION OF A MATRIX

If A is a complex $m \times n$ matrix then there exists an $m \times m$ unitary matrix U and an $n \times n$ unitary matrix V such that $D = U^*AV$ is a diagonal matrix[1]. Let $d_1, \ldots, d_k$ be the diagonal elements of D where $k = \min\{m, n\}$. Then U and V can be selected so that the diagonal elements are real numbers and $d_1 \geq d_2 \geq \cdots \geq d_k \geq 0$. The nonnegative diagonal elements d_i are unique, and if A is a real matrix then U and V can be chosen to be real orthogonal matrices. The decomposition $D = U^*AV$ is called the ***singular value decomposition of A***. The elements $d_1, \ldots, d_k$ are the ***singular values*** of A, the columns of U are ***left singular vectors***, and the columns of V are ***right singular vectors***.

Remark. For $m > n$, $D = \begin{pmatrix} D_1 \\ 0 \end{pmatrix}$ where $D_1 = \text{diag}(d_1, \ldots, d_n)$. Consequently, if U is partitioned into $U = (U_1, U_2)$ where U_1 has n columns, then it follows that $A = UDV^* = U_1 D_1 V^*$. The decomposition $A = U_1 D_1 V^*$ is frequently also called the singular value decomposition, and in many applications it suffices.

The following subroutines are available for finding the singular value decomposition $D = U^*AV$ of a matrix A.

CALL SSVDC($A, ka, m, n, D, E, U, ku, V, kv$,WORK,JOB,INFO)
CALL DSVDC($A, ka, m, n, D, E, U, ku, V, kv$,WORK,JOB,INFO)
CALL CSVDC($A, ka, m, n, D, E, U, ku, V, kv$,WORK,JOB,INFO)

A is a 2-dimensional array of dimension $ka \times n$ containing the $m \times n$ matrix whose singular value decomposition is to be computed. D is an array of dimension $\min\{m+1, n\}$. When any of the routines is called, the singular values of A are computed and stored in descending order of magnitude in $D(1), \ldots, D(k)$ where $k = \min\{m, n\}$.

JOB is an integer that controls the computation of the singular vectors. It is assumed that JOB $= I \cdot 10 + J$ when $I, J = 0, 1, \ldots, 9$. I and J have the following meaning.

- $I = 0$ Do not compute the left singular vectors.
- $I = 1$ Compute all m left singular vectors.
- $I > 1$ Compute the first $\min\{m, n\}$ left singular vectors. (Here we compute the decomposition $A = U_1 D_1 V^*$.)
- $J = 0$ Do not compute the right singular vectors.
- $J > 0$ Compute the right singular vectors.

U is a 2-dimensional array which contains the left singular vectors that are requested, and ku is the number of rows in the dimension statement for U in the calling program. It is assumed that $ku \geq m$. If no left singular vectors are requested (i.e., if JOB < 10) then U is ignored by the routines. Otherwise, U must be of dimension $ku \times m$ if all m left singular vectors are requested, and U must be of dimension $ku \times \min\{m, n\}$ if the first $\min\{m, n\}$ left singular vectors are requested.

[1] U^* denotes the adjoint matrix of U.

V is a 2-dimensional array which contains the right singular vectors that are requested, and kv is the number of rows in the dimension statement for V in the calling program. It is assumed that $kv \geq n$. If no right singular vectors are requested then V is ignored by the routines. Otherwise, V must be of dimension $kv \times n$ if the right singular vectors are requested.

E is an array of dimension n or larger, and WORK is an array of dimension m or larger. E and WORK are storage areas for the routines.

Remarks.

(1) If SSVDC is called then it is assumed that the arrays A, D, E, U, V,WORK are real arrays, if DSVDC is called then it is assumed that the arrays are double precision arrays, and if CSVDC is called then it is assumed that the arrays are complex arrays.

(2) The contents of A are destroyed by the routines. If left singular vectors are requested and there is sufficient storage in A to hold the vectors (there will be sufficient storage if $m \leq n$ or JOB ≥ 20), then one may set $U = A$. Similarly, if right singular vectors are requested and $m \geq n$ then one may set $V = A$. However, only one of the two arrays U and V may be identified with A.

Error Return. INFO is an iteger variable. If all the singular values are found then INFO will be set to 0 and the array E will contain zeros. However, if the j^{th} singular value cannot be found then INFO is set to j. In this case, if $j < k$ where $k = \min\{m, n\}$ then the singular values $d_{j+1}, \ldots, d_k$ will have been computed and stored in D. A will have been reduced to an upper bidiagonal matrix B with D as its diagonal and E its super diagonal. If U and V have been requested then $B = U^* AV$ will be satisfied.

Programming. SSVDC, DSVDC, and CSVDC are part of the LINPACK package of matrix subroutines released by Argonne National Laboratory. The routines were coded by G. W. Stewart (University of Maryland). The routines employ the vector subroutines SSWAP, SROT, SDOT, SSCAL, SAXPY, SNRM2, DSWAP, DROT, DDOT, DSCAL, DAXPY, DNRM2, and CSWAP, CSROT, CDOTC, CSCAL, CAXPY, SCNRM2. Also the subroutines SROTG and DROTG are called.

Reference. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

EVALUATION OF THE CHARACTERISTIC POLYNOMIAL OF A MATRIX

The following functions are available for computing the determinant of $A - xI$ where A is an $n \times n$ matrix, x a number, and I the $n \times n$ identity matrix.

DET(A, ka, n, x)
DPDET(A, ka, n, x)
CDET(A, ka, n, x)

DET is a real function that is used when A is a real matrix and x a real number, DPDET is a double precision function that is used when A is a double precision matrix and x a double precision number, and CDET is a complex function that is used when A is a complex matrix and x a complex number.

The value of the appropriate function is the determinant of the matrix $A - xI$. The argument ka has the value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geq n \geq 1$.

Note. A is destroyed during computation.

Algorithm. Gauss partial pivoting is performed to reduce $A - xI$ to upper triangular form. In CDET the pivots a_{kj} are selected so that $|\mathrm{Re}(a_{kj})| + |\mathrm{Im}(a_{kj})| = \mathrm{Max}\{|\mathrm{Re}(a_{ij})| + |\mathrm{Im}(a_{ij})| : i = j, \ldots, n\}$ rather than $|a_{kj}| = \max\{|a_{ij}| : i = j, \ldots, n\}$.

Programmer. A. H. Morris.

SOLUTION OF THE MATRIX EQUATION AX + XB = C

Given an $m \times m$ matrix A, $n \times n$ matrix B, and $m \times n$ matrix C. The subroutines ABSLV and DABSLV are available for obtaining the $m \times n$ matrix X which solves the equation $AX + XB = C$. ABSLV yields single precision results and DABSLV yields double precision results.

CALL ABSLV(MO,$m, n, A, ka, B, kb, C, kc$,WK,IND)
CALL DABSLV(MO,$m, n, A, ka, B, kb, C, kc$,WK,IND)

If ABSLV is called then it is assumed that A, B, C, and WK are real arrays. Otherwise, if DABSLV is called then it is assumed that A, B, C, and WK are double precision arrays.

It is assumed that $m \geq 1$ and $n \geq 1$. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is required that $ka \geq m$, $kb \geq n$, $kc \geq m$.

WK is an array of dimension $m^2 + n^2 + 2k$ or larger where $k = \max\{m, n\}$. WK is a general storage area for the routine.

MO is an input argument which specifies if the routine is being called for the first time. On an initial call MO = 0. In this case, A is reduced to lower real Schur form, B is reduced to upper real Schur form, and then the transformed system of equations is solved.

IND is a variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The solution was obtained and stored in C.
IND = 1 The equations are inconsistent for A and B.
IND = −1 A could not be reduced to lower Schur form.
IND = −2 B could not be reduced to upper Schur form.

If IND $\neq$ 0 then no solution is obtained.

When IND = 0, A contains the lower Schur form of the matrix A, B contains the upper Schur form of the matrix B, and WK contains the orthogonal matrices involved in the decompositions of A and B. This information can be reused to solve a new set of equations. The following options are available:

MO = 1 New matrices A and C are given. The data for B is reused in solving the new set of equations.
MO = 2 New matrices B and C are given. The data for A is reused in solving the new set of equations.
MO $\neq$ 0, 1, 2 A new matrix C is given. The data for A and B is reused in solving the new set of equations.

When the routine is recalled, it is assumed that m, n, and WK have not been modified.

Programming. ABSLV employs the subroutines ABSLV1, ORTHES, ORTRN1, SCHUR, SHRSLV, SLV, and DABSLV employs the routines DABSV1, DORTH, DRTRN1, DSCHUR, DSHSLV, DPSLV. ABSLV and DABSLV are adaptations by A. H. Morris of the subroutine AXPXB written by R. H. Bartels and G. W. Stewart (University of Texas at Austin).

Reference. Bartels, R. H. and Stewart, G. W., "Algorithm 432, Solution of the Matrix Equation $AX + XB = C$," *Comm. ACM* **15** (1972), pp. 820–826.

SOLUTION OF THE MATRIX EQUATION $A^tX + XA = C$ WHEN C IS SYMMETRIC

Given matrices A and C of order n where C is symmetric. Then the subroutines TASLV and DTASLV are available for obtaining the symmetric matrix X which solves the equation $A^tX + XA = C$. TASLV yields single precision results and DTASLV yields double precision results.

CALL TASLV(MO,n, A, ka, C, kc,WK,IND)
CALL DTASLV(MO,n, A, ka, C, kc,WK,IND)

If TASLV is called then it is assumed that A, C and WK are real arrays. Otherwise, if DTASLV is called then it is assumed that A, C, and WK are double precision arrays.

It is assumed that $n \geq 1$. The input arguments ka and kc have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is required that $ka \geq n$ and $kc \geq n$.

WK is an array of dimension $n^2 + 2n$ or larger that is a general storage area for the routine.

MO is an input argument which specifies if the routine is being called for the first time. On an initial call MO = 0. In this case, A is reduced to upper real Schur form and then the transformed system of equations is solved.

IND is a variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The solution was obtained and stored in C.
IND = 1 The equations are inconsistent.
IND = -1 A could not be reduced to upper Schur form.

If IND $\neq$ 0 then no solution is obtained.

When IND = 0, A contains the upper Schur form of the matrix A and WK contains the orthogonal matrix involved in the decomposition of A. This data can be reused to solve a new set of equations $A^tX + XA = \tilde{C}$. In this case, MO can be set to any nonzero value. When MO $\neq$ 0 it is assumed that only C has been modified. When the routine terminates, the solution for the new set of equations is stored in C.

Programming. TASLV employs the subroutines TASLV1, ORTHES, ORTRNI, SCHUR, SYMSLV, SLV and DTASLV employs the routines DTASV1,DORTH,DRTRNI,DSCHUR, DSYMSV, DPSLV. TASLV and DTASLV are adaptations by A. H. Morris of the subroutine ATXPXA written by R. H. Bartels and G. W. Stewart (University of Texas at Austin).

Reference. Bartels, R. H. and Stewart, G. W., "Algorithm 432 - Solution of the Matrix Equation $AX + XB = C$," *Comm. ACM* **15** (9-72), pp. 820-826.

SOLUTION OF THE MATRIX EQUATION $AX^2 + BX + C = 0$

Given complex $n \times n$ matrices A, B, and C. The following subroutine is available for obtaining a complex $n \times n$ matrix X which solves the equation $AX^2 + BX + C = 0$.

CALL SQUINT$(m, n, A, B, C$,IND,X,WK,ℓ, τ,MAX,IERR)

It is assumed that A, B, C, and X are 2-dimensional complex arrays of dimension $m \times n$ where $m \geq n$. When SQUINT is called, the $n \times n$ complex matrix solution obtained for $AX^2 + BX + C = 0$ is stored in X. A, B, and C are modified by the routine.

IND is an integer variable. On input, if IND $\neq 0$ then it is assumed that an initial approximation for the desired solution is provided in X by the user. Otherwise, if IND $= 0$ then the routine provides its own initial approximation. Then Newton iteration is performed. On output, IND = the number of iterations that were performed to compute X.

WK is a complex array of dimension ℓ that is a work space for the routine. It is assumed that $\ell \geq 7n^2 + n$. When SQUINT terminates, WK(1) is a complex number whose real part is the norm $||AX^2 + BX + C||_\infty$.

The argument τ is a real number. If $\tau \leq 0$ then X is computed to machine precision. Otherwise, if $\tau > 0$ then iteration terminates when $||AX^2 + BX + C|| < \tau$.

MAX is a variable. If MAX > 0 then MAX is the maximum number of iterations that may be performed. If MAX ≤ 0 then it is reset by the routine to 30, the default maximum number of iterations.

Error Return. IERR is a variable that is set by the routine. If a solution X is obtained, then IERR is assigned the value 0. Otherwise, IERR has one of the following values:

IERR = 1	MAX iterations were performed. More iterations are needed.
IERR = 2,3	Factorization of the equations could not be completed. X cannot be computed.
IERR = $10 + n$	Newton iteration failed on iteration n. Possibly too much accuracy was requested. X cannot be computed.
IERR = 999	(Input Error) Either $n < 1$, $m < n$, or $\ell < 7n^2 + n$.

When IERR = 1 occurs, X contains the most recent value obtained for the solution and WK(1) is a complex number whose real part is the latest value obtained for the norm $||AX^2 + BX + C||_\infty$.

Programming. SQUINT employs the subroutines SQUIN2, CQZHES, CQZIT, TRISLV, and CTRANS. These routines were designed by George J. Davis (Georgia State University, Atlanta, Georgia). CQZHES and CQZIT are modifications of the EISPACK subroutines QZHES and QZIT, developed at Argonne National Laboratory. The function SPMPAR is also used.

References.

(1) Davis, G. J., "Algorithm 598. An Algorithm to Compute Solvents of the Matrix Equation $AX^2 + BX + C = 0$," ***ACM Trans. Math Software*** **9** (1983), pp. 246–254.

(2) Garbow, B. S., et al., ***Matrix Eigensystems Routines - EISPACK Guide Extension***, Springer-Verlag, 1977.

EXPONENTIAL OF A REAL MATRIX

Let A be a real matrix of order $n \geq 1$. Then the subroutines MEXP and DMEXP are available for computing the exponential matrix $e^A = \sum_{i=0}^{\infty} A^i/i!$. MEXP yields single precision results and DMEXP yields double precision results.

CALL MEXP(A, ka, n, Z, kz,WK,IERR)

A is a real matrix of order $n \geq 1$ and Z a real 2-dimensional array. MEXP computes e^A and stores the results in Z. The arguments ka and kz have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kz = the number of rows in the dimension statement for Z in the calling program

It is assumed that $ka \geq n$, $kz \geq n$, and that A and Z are different storage areas. If $n > 1$ then A is destroyed.

WK is a real array of dimension $n(n+8)$ or larger that is a work space for the routine.

IERR is a variable that reports the status of the results. When MEXP terminates, IERR is assigned one of the following values:

IERR = 0 The exponential was computed.

IERR = 1 The norm $\|A\|_1 = \max_j \sum_i |a_{ij}|$ is too large. e^A cannot be computed.

IERR = 2 The Pade denominator matrix is singular. (This should never occur.)

Algorithm. A is balanced, yielding a matrix $B = D^{-1}P^tAPD$ where D is a diagonal matrix, P a permutation matrix, and $\|B\|_1 < \|A\|_1$. Next m is set to the smallest nonnegative integer such that $\min\{\|B\|_1, \|B\|_\infty\} \leq 2^m$, and the 8^{th} diagonal Pade approximation for e^x is used to compute $\exp(B/2^m)$. Then $e^B = [\exp(B/2^m)]^{2^m}$ is obtained by m squarings, and $e^A = PDe^BD^{-1}P^t$ is applied.

Programming. MEXP calls the subroutines BALANC, BALINV, and SLV. The function IPMPAR is also used. MEXP was written by A. H. Morris

Reference. Ward, Robert, C., "Numerical Computation of the Matrix Exponential with Accuracy Estimate," ***SIAM J. Numerical Analysis*** **14** (1977), pp. 600-610

CALL DMEXP(A, ka, n, Z, kz,WK,IERR)

A is a double precision matrix of order $n \geq 1$ and Z a double precision 2 dimensional array. DMEXP computes e^A and stores the results in Z. The arguments ka and kz have the following values:

ka the number of rows in the dimension statement for A in the calling program

kz the number of rows in the dimension statement for Z in the calling program

It is assumed that $ka \geq n$, $kz \geq n$, and that A and Z are different storage areas. If $n > 1$ then A is destroyed.

WK is a double precision array of dimension $n(n + 12)$ or larger that is a work space for the routine.

IERR is a variable that reports the status of the results. When DMEXP terminates, IERR is assigned one of the following values:

IERR = 0 The exponential was computed.
IERR = 1 The norm $||A||_1$ is too large. e^A cannot be computed.
IERR = 2 The Pade denominator matrix is singular. (This should never occur.)

Programming. DMEXP calls the subroutines DBAL, DBALNV, and DPSLV. The function IPMPAR is also used. DMEXP was written by A. H. Morris.

Reference. Ward, Robert C., "Numerical Computation of the Matrix Exponential with Accuracy Estimate," ***SIAM J. Numerical Analysis*** **14** (1977), pp. 600-610.

SOLVING SYSTEMS OF 200–400 LINEAR EQUATIONS

For $n \geq 1$, let A denote an $n \times n$ matrix and b a column vector of dimension n. Then the subroutines LE, DPLE, and CLE are available for solving the equations $Ax = b$ where A is not stored in-core. For large n, these routines require a work space of dimension $\approx n^2/4$. This permits the solution of systems of equations of double the order permitted by the standard solution procedures.

CALL LE(ROWK,n, b, X,WK,IWK,IERR)
CALL DPLE(ROWK,n, b, X,WK,IWK,IERR)
CALL CLE(ROWK,n, b, X,WK,IWK,IERR)

X is an array of dimension n and IERR an integer variable. When the equations are solved, then IERR is set to 0 and the solution is stored in X.

ROWK is the name of a user defined subroutine that has the format:
CALL ROWK(n, k, R)
R is an array of dimension n and $k = 1, \ldots, n$. When ROWK is called, the k^{th} row of the matrix A is stored in R. ROWK must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $[n^2/4] + n + 3$ or larger,[1] and IWK is an integer array of dimension $\max\{1, n-1\}$ or larger. WK and IWK are work spaces for the routines.

Error Return. IERR $= k$ when the first k rows of A are found to be linearly dependent.

Remarks.

(1) When LE is called then it is assumed that b, X, WK and the array R in ROWK are real arrays. When DPLE is called then it is assumed that these arrays are double precision arrays, and when CLE is called then it is assumed that the arrays are complex.
(2) When the equations are solved, ROWK is called to attach the first row of A, then the second row, etc. Each row of A is attached only once.
(3) The array b is not modified by the routines.

Algorithm. The partial pivot Henderson-Wassyng procedure is used.

Programming. LE, DPLE, and CLE are modified versions by A. H. Morris of the subroutine TE, written by A. Wassyng (University of the Witwatersrand, Johannesburg, South Africa).

Example. Consider a system of $n = 300$ real linear equations $Ax = b$ where the rows of A are stored, one row per logical record, in sequence in an unformatted file (say file 4). Then the following code can be used to solve the equations:

[1] Here $[n^2/4]$ denotes the largest integer $\leq n^2/4$.

```
REAL B(300),X(300),WK(22803)
INTEGER IWK(300)
EXTERNAL GETROW
DATA N/300/
.
.
.
REWIND 4
CALL LE(GETROW,N,B,X,WK,IWK,IERR)
```

Here GETROW may be defined by:

```
SUBROUTINE GETROW(N,I,R)
REAL R(N)
READ(4) (R(J),J=1,N)
RETURN
END
```

Reference. Wassyng, A., "Solving $Ax = b$: A Method with Reduced Storage Requirements," ***SIAM J. Numerical Analysis*** **19** (1982), pp. 197–204.

BAND MATRIX STORAGE

For an $m \times n$ matrix $A = (a_{ij})$, let m_ℓ be the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. Then m_ℓ and m_u are called the ***lower*** and ***upper band widths*** of A, and $m_\ell + m_u + 1$ the ***total band width*** of A. It is clear that $0 \le m_\ell < m$ and $0 \le m_u < n$, and that $a_{ij} \neq 0$ only when $i - m_\ell \le j \le i + m_u$. If the band width $m_\ell + m_u + 1$ is sufficiently small, then it is also clear that a considerable savings in storage can occur by storing only the nonzero diagonals of A. In the adopted band storage scheme, the nonzero diagonals are the columns of an $m \times (m_\ell + m_u + 1)$ matrix $B = (b_{ik})$. For each nonzero a_{ij}, $b_{ik} = a_{ij}$ where $k = j - i + m_\ell + 1$. The remaining b_{ik}'s are zeros.

Example. Consider the matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & a_{46} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} & a_{57} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & 0 & a_{76} & a_{77} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{87} \end{pmatrix}$$

where $m_\ell = 1$ and $m_u = 2$. Then A will be stored in band form as follows:

$$B = \begin{pmatrix} 0 & a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} & a_{35} \\ a_{43} & a_{44} & a_{45} & a_{46} \\ a_{54} & a_{55} & a_{56} & a_{57} \\ a_{65} & a_{66} & a_{67} & 0 \\ a_{76} & a_{77} & 0 & 0 \\ a_{87} & 0 & 0 & 0 \end{pmatrix}$$

Remark. The first m_ℓ columns of B contain the nonzero diagonals of A below the main diagonal, the $(m_\ell + 1)^{st}$ column of B contains the main diagonal, and the last m_u columns of B contain the nonzero diagonals of A above the main diagonal.

CONVERSION OF BANDED MATRICES TO AND FROM THE STANDARD FORMAT

The following subroutines permit one to convert matrices to and from the standard format.

CALL CVBR$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL CVBD$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL CVBC$(A, ka, m, n, m_\ell, m_u, B, kb)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

B is a 2-dimensional array of dimension $kb \times n$ where $kb \geq m$. CVBR is used if A and B are real arrays, CVBD is used if A and B are double precision arrays, and CVBC is used if A and B are complex arrays. When the routine is called, the matrix A is stored in the array B in the standard format.

Remark. B may begin in the same location as A. If B begins in the same location then it is assumed that $kb = ka$. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris.

CALL CVRB$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL CVDB$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL CVCB$(A, ka, m, n, m_\ell, m_u, B, kb)$

A is an $m \times n$ matrix stored in the standard format, and m_ℓ and m_u are integers such that $0 \leq m_\ell < m$ and $0 \leq m_u < n$. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $ka \geq m$.

B is a 2-dimension array of dimension $kb \times \ell$ where $kb \geq m$ and $\ell \geq m_\ell + m_u + 1$. CVRB is used if A and B are real arrays, CVDB is used if A and B are double precision arrays, and CVCB is used if A and B are complex arrays. When the routine is called, the m_ℓ diagonals of A immediately below the main diagonal, the main diagonal, and the m_u diagonals immediately above the main diagonal are stored in band form in B.

Remarks.

(1) Given a matrix $A = (a_{i,j})$, then these routines may be used to convert A to band form when the lower and upper bandwidths m_ℓ and m_u of A are known. If m_ℓ and m_u are

not known, then the subroutines CVRB1, CVDB1, and CVCB1 described below can be used to convert A to band form.

(2) B may begin in the same location as A. If B begins in the same location then it is assumed that $kb = ka$. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris.

CALL CVRB1$(A, ka, m, n, m_\ell, m_u, B, kb, \ell,$IERR$)$
CALL CVDB1$(A, ka, m, n, m_\ell, m_u, B, kb, \ell,$IERR$)$
CALL CVCB1$(A, ka, m, n, m_\ell, m_u, B, kb, \ell,$IERR$)$

A is an $m \times n$ matrix stored in the standard format. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that A is to be stored in band form in B. B is a 2-dimensional array of dimension $kb \times \ell$ where $kb \geq m$. The argument ℓ is an estimate of the maximum number of diagonals of A that will have to be stored.

CVRB1 is used if A and B are real arrays, CVDB1 is used if A and B are double precision arrays, and CVCB1 is used if A and B are complex arrays. IERR, m_ℓ, and m_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for B then A is stored in band form in B. Also IERR is assigned the value 0, m_ℓ = the number of diagonals of A below the main diagonal containing nonzero elements, and m_u = the number of diagonals above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value $m_\ell + m_u + 1$. Reset $\ell \geq$ IERR.

Remark. B may begin in the same location as A. If B begins in the same location then it is assumed that $kb = ka$. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programming. CVRB1 calls the subroutine CVRB, CVDB1 calls the subroutine CVDB, and CVCB1 calls the subroutine CVCB. These routines were written by A. H. Morris.

CONVERSION OF BANDED MATRICES TO AND FROM SPARSE FORM

The following subroutines permit one to convert matrices to and from sparse form.

CALL MCVBS($A, ka, m, n, m_\ell, m_u, B, IB, JB$,NUM,IERR)
CALL DMCVBS($A, ka, m, n, m_\ell, m_u, B, IB, JB$,NUM,IERR)
CALL CMCVBS($A, ka, m, n, m_\ell, m_u, B, IB, JB$,NUM,IERR)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statment for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

It is assumed that A is to be stored in sparse form in the arrays B, IB, JB. NUM is the estimated maximum number of elements that will appear in B and JB. It is assumed that B and JB are of dimension $\max\{1, \text{NUM}\}$ and that IB is of dimension $m + 1$.

MCVES is used if A and B are real arrays, DMCVBS is used if A and B are double precision arrays, and CMCVBS is used if A and B are complex arrays. IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for B and JB, then IERR is assigned the value 0 and A is stored in sparse form in B, IB, JB.

Error Return. If there is not sufficient storage in B and JB for the i^{th} row of A, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A will have been stored in B and JB. Also $IB(1), \ldots, IB(i)$ will contain the appropriate row locations.

Programmer. A. H. Morris.

CALL MCVSB($A, IA, JA, m, n, B, kb, \ell, m_\ell, m_u$,IERR)
CALL DMCVSB($A, IA, JA, m, n, B, kb, \ell, m_\ell, m_u$,IERR)
CALL CMCVSB($A, IA, JA, m, n, B, kb, \ell, m_\ell, m_u$,IERR)

A is an $m \times n$ sparse matrix stored in the arrays A, IA, JA. It is assumed that A is to be stored in band form in B. B is a 2-dimensional array of dimension $kb \times \ell$ where $kb \geq m$. The argument ℓ is an estimate of the maximum number of diagonals of A that will have to be stored.

MCVSB is used if A and B are real arrays, DMCVSB is used if A and B are double precision arrays, and CMCVSB is used if A and B are complex arrays. IERR, m_ℓ, and m_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for B then A is stored in band form in B. Also IERR is assigned the value 0, m_ℓ = the number of diagonals of A below the main diagonal containing nonzero elements, and m_u = the number of diagonals above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value $m_\ell + m_u + 1$. Reset $\ell \geq$ IERR.

Programmer. A. H. Morris.

CONVERSION OF BANDED REAL MATRICES TO AND FROM DOUBLE PRECISION FORM

The following subroutines are available for converting banded real matrices to and from double precision form.

CALL BCVRD$(A, ka, m, n, m_\ell, m_u, B, kb)$

A is an $m \times n$ real matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

B is a 2-dimensional double precision array of dimension $kb \times \ell$ where $kb \geq m$ and $\ell \geq m_\ell + m_u + 1$. When BCVRD is called, A is stored in band form in B.

Programmer. A. H. Morris.

CALL BCVDR$(A, ka, m, n, m_\ell, m_u, B, kb)$

A is an $m \times n$ double precision matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

B is a 2–dimensional real array of dimension $kb \times \ell$ where $kb \geq m$ and $\ell \geq m_\ell + m_u + 1$. When BCVDR is called, A is stored in band form in B.

Programmer. A. H. Morris.

THE REAL AND IMAGINARY PARTS OF A BANDED COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\mathrm{Re}(A) = (\mathrm{Re}(a_{ij}))$ and $\mathrm{Im}(A) = (\mathrm{Im}(a_{ij}))$ denote the real and imaginary parts of A. If the matrix A is stored in band form, then the following subroutines are available for obtaining $\mathrm{Re}(A)$ and $\mathrm{Im}(A)$ in band form.

CALL BREAL$(A, ka, m, n, m_\ell, m_u, B, kb, \ell, n_\ell, n_u, \mathrm{IERR})$

A is an $m \times n$ complex matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

B is a 2-dimensional real array of dimension $kb \times \ell$ where $kb \geq m$. The input argument ℓ is an estimate of the maximum number of diagonals of $\mathrm{Re}(A)$ which will have to be stored ($\ell \leq m_\ell + m_u + 1$). IERR, n_ℓ, and n_u are integer variables. When BREAL is called, if ℓ specifies sufficient storage for B then $\mathrm{Re}(A)$ is stored in band form in B. Also, IERR is assigned the value 0, n_ℓ = the number of diagonals of $\mathrm{Re}(A)$ below the main diagonal containing nonzero elements, and n_u = the number of diagonals of $\mathrm{Re}(A)$ above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value ν where ν is the number of columns needed for B. Reset $\ell \geq \nu$.

Programmer. A. H. Morris.

CALL BIMAG$(A, ka, m, n, m_\ell, m_u, B, kb, \ell, n_\ell, n_u, \mathrm{IERR})$

A is an $m \times n$ complex matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

B is a 2-dimensional real array of dimension $kb \times \ell$ where $kb \geq m$. The input argument ℓ is an estimate of the maximum number of diagonals of $\mathrm{Im}(A)$ which will have to be stored ($\ell \leq m_\ell + m_u + 1$). IERR, n_ℓ, and n_u are integer variables. When BIMAG is called, if ℓ specifies sufficient storage for B then $\mathrm{Im}(A)$ is stored in band form in B. Also, IERR is assigned the value 0, n_ℓ = the number of diagonals of $\mathrm{Im}(A)$ below the main diagonal containing nonzero elements, and n_u = the number of diagonals of $\mathrm{Im}(A)$ above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value ν where ν is the number of columns needed for B. Reset $\ell \geq \nu$.

Programmer. A. H. Morris.

COMPUTING A + Bi FOR BANDED REAL MATRICES A AND B

Given the real $m \times n$ matrices A and B stored in band form. Then the subroutine BCVRC is available for obtaining the complex matrix $A + Bi$ where $i = \sqrt{-1}$.

CALL BCVRC$(m, n, A, \kappa a, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u,$IERR)

A and B are real $m \times n$ matrices stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, m_u the number of diagonals of A above the main diagonal containing nonzero elements, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program.

It is assumed that $A + Bi$ is to be stored in band form in C. C is a 2–dimensional complex array of dimension $kc \times \ell$ where $kc \geq n$ The input argument ℓ is an estimate of the maximum number of diagonals of $A + Bi$ which will have to be stored ($\ell \geq \max\{m_\ell, n_\ell\} + \max\{m_u, n_u\} + 1$). IERR, ν_ℓ, and ν_u are integer variables. When BCVRC is called, if ℓ specifies sufficient storage for C then $A + Bi$ is stored in band form in C. Also IERR is assigned the value 0, ν_ℓ = the number of diagonals of $A + Bi$ below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of $A + Bi$ above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for C, then IERR is assigned the value ν where ν is the number of columns needed for C. Reset $\ell \geq \nu$.

Programmer. A. H. Morris.

TRANSPOSING BANDED MATRICES

The following subroutines are available for transposing banded matrices.

CALL BPOSE$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL DBPOSE$(A, ka, m, n, m_\ell, m_u, B, kb)$
CALL CBPOSE$(A, ka, m, n, m_\ell, m_u, B, kb)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

B is a 2-dimensional array of dimension $kb \times \ell$ where $kb \geq n$ and $\ell \geq m_\ell + m_u + 1$. BPOSE is used if A and B are real arrays, DBPOSE is used if A and B are double precision arrays, and CBPOSE is used if A and B are complex arrays. When the routine is called, the transpose A^t of A is stored in band form in B.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris.

ADDITION OF BANDED MATRICES

Let A and B be $m \times n$ matrices stored in band form. The following subroutines are available for computing the sum $C = A + B$.

CALL BADD$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u,$IERR$)$
CALL DBADD$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u,$IERR$)$
CALL CBADD$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u,$IERR$)$

A and B are $m \times n$ matrices stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, m_u the number of diagonals of A above the main diagonal containing nonzero elements, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program.

It is assumed that $A + B$ is to be stored in band form in C. C is a 2-dimensional array of dimension $kc \times \ell$ where $kc \geq m$. The input argument ℓ is an estimate of the maximum number of diagonals of $A + B$ which will have to be stored ($\ell \leq \max\{m_\ell, n_\ell\} + \max\{m_u, n_u\} + 1$). BADD is used if A and B are real arrays, DBADD is used if A and B are double precision arrays, and CBADD is used if A and B are complex arrays. IERR, ν_ℓ, and ν_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for C then $A + B$ is computed and stored in band form in C. Also IERR is assigned the value 0, ν_ℓ = the number of diagonals of $A + B$ below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of $A + B$ above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for C, then IERR is assigned the value ν where ν is the number of columns needed for C. Reset $\ell = \nu$.

Remarks. If $m_\ell = n_\ell$ then C may begin in the same location as A. If C begins in the same location as A, then it is assumed that $kc = ka$ and that the arrays A and B do not overlap. In this case, the result C will overwrite the input data A. Similarly, if $m_\ell = n_\ell$ then C may begin in the same location as B when $kc = kb$ and A and B do not overlap. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Example. If $B = -A$ then ℓ may be assigned any value ≥ 1. In this case, C will contain only the main diagonal of the zero matrix $A + B$, and $\nu_\ell = \nu_u = 0$.

Programmer. A. H. Morris

SUBTRACTION OF BANDED MATRICES

Let A and B be $m \times n$ matrices stored in band form. The following subroutines are available for computing the difference $C = A - B$.

CALL BSUBT$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u$,IERR)
CALL DBSUBT$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u$,IERR)
CALL CBSUBT$(m, n, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, \ell, \nu_\ell, \nu_u$,IERR)

A and B are $m \times n$ matrices stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, m_u the number of diagonals of A above the main diagonal containing nonzero elements, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program.

It is assumed that $A - B$ is to be stored in band form in C. C is a 2-dimensional array of dimension $kc \times \ell$ where $kc \geq m$. The input argument ℓ is an estimate of the maximum number of diagonals of $A - B$ which will have to be stored ($\ell \leq \max\{m_\ell, n_\ell\} + \max\{m_u, n_u\} + 1$). BSUBT is used if A and B are real arrays, DBSUBT is used if A and B are double precision arrays, and CBSUBT is used if A and B are complex arrays. IERR, ν_ℓ, and ν_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for C then $A - B$ is computed and stored in band form in C. Also IERR is assigned the value 0, ν_ℓ = the number of diagonals of $A - B$ below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of $A - B$ above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for C, then IERR is assigned the value ν where ν is the number of columns needed for C. Reset $\ell \geq \nu$.

Remarks. If $m_\ell \geq n_\ell$ then C may begin in the same location as A. If C begins in the same location as A, then it is assumed that $kc = ka$ and that the arrays A and B do not overlap. In this case, the result C will overwrite the input data A. Similarly, if $m_\ell \leq n_\ell$ then C may begin in the same location as B when $kc = kb$ and A and B do not overlap. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geq m$).

Example. If $B = A$ then ℓ may be assigned any value ≥ 1. In this case, C will contain only the main diagonal of the zero matrix $A - B$, and $\nu_\ell = \nu_u = 0$.

Programmer. A. H. Morris

MULTIPLICATION OF BANDED MATRICES

Let A and B be matrices stored in band form. The following subroutines are available for computing the product $C = AB$.

CALL BPROD$(m, n, \ell, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, nc, \nu_\ell, \nu_u,$IERR)
CALL DBPROD$(m, n, \ell, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, nc, \nu_\ell, \nu_u,$IERR)
CALL CBPROD$(m, n, \ell, A, ka, m_\ell, m_u, B, kb, n_\ell, n_u, C, kc, nc, \nu_\ell, \nu_u,$IERR)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, and m_u the number of diagonals of A above the main diagonal containing nonzero elements. B is an $n \times \ell$ matrix stored in band form, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. It is assumed that $ka \geq m$ and $kb \geq n$.

It is assumed that AB is to be stored in band form in C. C is a 2–dimensional array of dimension $kc \times nc$ where $kc \geq m$. The input argument nc is an estimate of the maximum number of diagonals of AB which will have to be stored ($nc \leq \min\{n-1, m_\ell + n_\ell\} + \min\{\ell - 1, m_u + n_u\} + 1$). BPROD is used if A,B, and C are real arrays, DBPROD is used if A,B, and C are double precision arrays, and CBPROD is used if A,B, and C are complex arrays. IERR, ν_ℓ, and ν_u are integer variables. When the routine is called, if nc specifies sufficient storage for C then AB is computed and stored in band form in C. Also, IERR is assigned the value 0, ν_ℓ = the number of diagonals of AB below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of AB above the main diagonal containing nonzero elements.

Error Return. If nc does not specify sufficient storage for C, then IERR is assigned the value ν where ν is the number of columns needed for C. Reset $nc \geq \nu$.

Note. It is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris.

PRODUCT OF A REAL BANDED MATRIX AND VECTOR

Let A be a real $m \times n$ matrix stored in band form. Then the following subroutines are available for multiplying A with a real vector.

CALL BVPRD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a column vector of dimension n and y an array of dimension m. When BVPRD is called, the product Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL BVPRD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are column vectors of dimension n and m respectively. When BVPRD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL BTPRD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a row vector of dimension m and y an array of dimension n. When BTPRD is called, the product xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL BTPRD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ r stored in band form, m_ℓ the number of diagonals below the main diagonal contai nzero elements, and m_u then number of diagonals above the main diagonal contai nzero elements. The argument ka is the number of rows in the dimension statement fo n the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are row vectors of dimension m and n respectively. When BTPRD1 is called, xA + computed and stored in y.

Remark. It is assumed th the arrays A, x, y do not overlap.

Programmer. A. H. M

PRODUCT OF A DOUBLE PRECISION BANDED MATRIX AND VECTOR

Let A be a double precision $m \times n$ matrix stored in band form. Then the following subroutines are available for multiplying A with a double precision vector.

CALL DBVPD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are double precision arrays. When DBVPD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DBVPD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are column vectors of dimension n and m respectively. A, x, y are double precision arrays. When DBVPD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DBTPD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are double precision arrays. When DBTPD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DBTPD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are row vectors of dimension m and n respectively. A, x, y are double precision arrays. When DBTPD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

PRODUCT OF A COMPLEX BANDED MATRIX AND VECTOR

Let A be a complex $m \times n$ matrix stored in band form. Then the following subroutines are available for multiplying A with a complex vector.

CALL CBVPD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are complex arrays. When CBVPD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CBVPD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are column vectors of dimension n and m respectively. A, x, y are complex arrays. When CBVPD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CBTPD$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are complex arrays. When CBTPD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CBTPD1$(m, n, A, ka, m_\ell, m_u, x, y)$

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$.

The arguments x and y are row vectors of dimension m and n respectively. A, x, y are complex arrays. When CBTPD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

L_1 NORM OF A REAL BANDED MATRIX

If $A = (a_{ij})$ is a real banded matrix then the following functions are available for computing the ℓ_1 norm $||A||_1 = \max_j \Sigma_i |a_{ij}|$ of A.

B1NRM$(A, ka, m, n, m_\ell, m_u)$
DB1NRM$(A, ka, m, n, m_\ell, m_u)$

B1NRM is used if A is a real array and DB1NRM is used if A is a double precision array. B1NRM is a real function and DB1NRM a double precision function.

The function has the value $||A||_1$ for any $m \times n$ matrix A stored in band form. The argument m_ℓ is the number of diagonals below the main diagonal containing nonzero elements, m_u the number of diagonals above the main diagonal containing nonzero elements, and ka the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

Remark. DB1NRM must be declared to be of type DOUBLE PRECISION in the calling program.

Programming. B1NRM calls the function SASUM and DB1NRM calls the function DASUM. B1NRM and DB1NRM were written by A. H. Morris.

L_∞ NORM OF A REAL BANDED MATRIX

If $A = (a_{ij})$ is a real banded matrix then the following functions are available for computing the ℓ_∞ norm $||A||_\infty = \max_i \Sigma_j |a_{ij}|$ of A.

BNRM$(A, ka, m, n, m_\ell, m_u)$
DBNRM$(A, ka, m, n, m_\ell, m_u)$

BNRM is used if A is a real array and DBNRM is used if A is a double precision array. BNRM is a real function and DBNRM a double precision function.

The function has the value $||A||_\infty$ for any $m \times n$ matrix A stored in band form. The argument m_ℓ is the number of diagonals below the main diagonal containing nonzero elements, m_u the number of diagonals above the main diagonal containing nonzero elements, and ka the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m$, $0 \leq m_u < n$, and $ka \geq m$.

Remark. DBNRM must be declared to be of type DOUBLE PRECISION in the calling program.

Programmer. A. H. Morris.

SOLUTION OF BANDED SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ real matrix stored in band form and b a real column vector of dimension n. The subroutine BSLV is available for solving the system of equations $Ax = b$, and the subroutine BSLV1 is available for solving the transposed system of equations $A^t x = b$. On an initial call to either routine, partial pivot Gauss elimination is first employed to obtain an LU decomposition of A, and then the equations are solved. BSLV and BSLV1 always generate the same LU decomposition of A. After the decomposition is obtained on an initial call to BSLV or BSLV1, either routine may be called to solve a new system of equations $Ax = r$ or $A^t x = r$ without having to redecompose the matrix A.

CALL BSLV(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)
CALL BSLV1(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)

BSLV is called for solving $Ax = b$ and BSLV1 is called for solving $A^t x = b$. The argument m_ℓ is the number of diagonals below the main diagonal of A containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. It is assumed that $n \geq 1$, $0 \leq m_\ell < n$, and $0 \leq m_u < n$. MO is an input argument which specifies if BSLV or BSLV1 is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional array of dimension $ka \times m$ where $ka \geq n$ and $m \geq 2m_\ell + m_u + 1$. On input, the first $m_\ell + m_u + 1$ columns of the array contain the matrix A in band form. When the routine terminates, the array A will contain the upper triangular matrix U of the LU decomposition and the multipliers which were used to obtain it.

X is an array of dimension n or larger. On input, X contains the vector b. On output, X will contain the solution of the system of equations.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the LU decomposition are stored in IWK.

On an initial call to BSLV or BSLV1, IND is an integer variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The system of equations was solved.
IND = −1 Either $n < 0$ or $ka < n$.
IND = −2 Either $m_\ell < 0$ or $m_\ell > n$.
IND = −3 Either $m_u < 0$ or $m_u > n$.
IND = k Column k of A has been reduced to a column containing only zeros.

After an initial call to BSLV or BSLV1, if IND = 0 on output then either routine may be called with MO $\neq$ 0. When MO $\neq$ 0 it is assumed that only b may have been modified. BSLV is called for solving the new set of equations $Ax = b$, and BSLV1 is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to BSLV or BSLV1 to solve the new set of equations. On input, X contains

the new vector b. On output, X will contain the solution to the new set of equations. In this case, IND is not referenced by the routine.

Programming. BSLV and BSLV1 employ the subroutines SNBFA,SNBSL,SAXPY,SSCAL, SSWAP and the functions ISAMAX and SDOT. SNBFA and SNBSL were written by E. A. Voorhees (Los Alamos Scientific Laboratory) and modified by A. H. Morris. The original versions of SNBFA and SNBSL are distributed by the SLATEC library.

COMPUTATION OF THE CONDITION NUMBER OF A REAL BANDED MATRIX

If A is a real $n \times n$ banded matrix then the following subroutine is available for estimating the ℓ_1 condition number $\text{cond}_1(A)$ of A.

CALL B1CND(A, ka, n, m_ℓ, m_u,COND,IWK,WK,IND)

A is a real $n \times n$ matrix stored in band form. A, ka, m_ℓ, m_u, and IWK are the same arguments used for the subroutines BSLV and BSLV1, the only exception being that IWK now is an array of dimension $2n$ or larger.

COND is a real variable. When B1CND is called, then the subroutine BSLV is first invoked to obtain an LU decomposition of A which is stored in A and IWK. Then COND is assigned the value 0 if A is singular and an approximation of the condition number $\text{cond}_1(A)$ if A is nonsingular.

WK is an array of dimension $2n$ or larger that is a work space for the routine.

IND is an integer variable that reports the status of the results. When B1CND terminates, IND has one of the following values:

- IND = 1 A is singular and COND = 0.
- IND = 0 A is nonsingular and COND is an approximation of the condition number of A.
- IND = −1 Either $n < 0$ or $ka < n$.
- IND = 2 Either $m_\ell < 0$ or $m_\ell > n$.
- IND = 3 Either $m_u < 0$ or $m_u > n$.

When an error is detected, the routine immediately terminates.

Usage. After B1CND terminates, if IND = 0 then BSLV or BSLV1 may be called with MO ≠ 0 to solve a system of equations. In this case A, ka, n, m_ℓ, m_u and IWK are used by BSLV and BSLV1, and must not be modified by the user.

Algorithm. A modification of the Hager procedure by Nicholas J. Higham (University of Manchester, England) is used.

Programming. B1CND employs the subroutines SONEST, BSLV, BSLV1, SNBFA, SNBSL, SCOPY, SAXPY, SSCAL, and SSWAP and the functions B1NRM, SDOT, SASUM, and ISAMAX. B1CND was written by A. H. Morris. SONEST was written by N. J. Higham and modified by A. H. Morris.

References.

(1) Higham, N.J.,"FORTRAN codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," *ACM Trans. Math Software* 14 (1988), pp. 381-396.

(2) _____,"Algorithm 674: "FORTRAN codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," ***ACM Trans. Math Software*** **15** (1989), p. 168.

DOUBLE PRECISION SOLUTION OF BANDED SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ double precision matrix stored in band form and b a double precision column vector of dimension n. The subroutine DBSLV is available for solving the system of equations $Ax = b$, and the subroutine DBSLV1 is available for solving the transposed system of equations $A^t x = b$. On an initial call to either routine, partial pivot Gauss elimination is first employed to obtain an LU decomposition of A, and then the equations are solved. DBSLV and DBSLV1 always generate the same LU decomposition of A. After the decomposition is obtained on an initial call to DBSLV or DBSLV1, either routine may be called to solve a new system of equations $Ax = r$ or $A^t x = r$ without having to redecompose the matrix A.

CALL DBSLV(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)
CALL DBSLV1(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)

DBSLV is called for solving $Ax = b$ and DBSLV1 is called for solving $A^t x = b$. The argument m_ℓ is the number of diagonals below the main diagonal of A containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. It is assumed that $n \geq 1$, $0 \leq m_\ell < n$, and $0 \leq m_u < n$. MO is an input argument which specifies if DBSLV or DBSLV1 is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a double precision 2-dimensional array of dimension $ka \times m$ where $ka \geq n$ and $m \geq 2m_\ell + m_u + 1$. On input, the first $m_\ell + m_u + 1$ columns of the array contain the matrix A in band form. When the routine terminates, the array A will contain the upper triangular matrix U of the LU decomposition and the multipliers which were used to obtain it.

X is a double precision array of dimension n or larger. On input, X contains the vector b. On output, X will contain the solution of the system of equations.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the LU decomposition are stored in IWK.

On an initial call to DBSLV or DBSLV1, IND is an integer variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The system of equations was solved.
IND = -1 Either $n \leq 0$ or $ka < n$.
IND = -2 Either $m_\ell < 0$ or $m_\ell \geq n$.
IND = -3 Either $m_u < 0$ or $m_u \geq n$.
IND = k Column k of A has been reduced to a column containing only zeros.

After an initial call to DBSLV or DBSLV1, if IND = 0 on output then either routine may be called with MO $\neq$ 0. When MO $\neq$ 0 it is assumed that only b may have been modified. DBSLV is called for solving the new set of equations $Ax = b$, and DBSLV1 is called

for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to DBSLV or DBSLV1 to solve the new set of equations. On input, X contains the new vector b. On output, X will contain the solution to the new set of equations. In this case, IND is not referenced by the routine.

Programming DBSLV and DBSLV1 employ the subroutines DBFA, DBSL, DAXPY, DSCAL, DSWAP and the functions IDAMAX and DDOT. DBFA and DBSL are adaptations by A. H. Morris of the subroutines SNBFA and SNBSL, written by E. A. Voorhees (Los Alamos Scientific Laboratory). SNBFA and SNBSL are distributed by the SLATEC library.

COMPUTATION OF THE CONDITION NUMBER OF A DOUBLE PRECISION BANDED MATRIX

If A is a double precision $n \times n$ banded matrix then the following subroutine is available for estimating the ℓ_1 condition number $\text{cond}_1(A)$ of A.

CALL DB1CND(A, ka, n, m_ℓ, m_u,COND,IWK,WK,IND)

A is a double precision $n \times n$ matrix stored in band form. A, ka, m_ℓ, m_u, and IWK are the same arguments used for the subroutines DBSLV and DBSLV1, the only exception being that IWK now is an array of dimension $2n$ or larger.

COND is a double precision variable. When DB1CND is called, then the subroutine DBSLV is first invoked to obtain an LU decomposition of A which is stored in A and IWK. Then COND is assigned the value 0 if A is singular and an approximation of the condition number $\text{cond}_1(A)$ if A is nonsingular.

WK is a double precision array of dimension $2n$ or larger that is a work space for the routine.

IND is an integer variable that reports the status of the results. When DB1CND terminates, IND has one of the following values:

IND = 1 A is singular and COND = 0.
IND = 0 A is nonsingular and COND is an approximation of the condition number of A.
IND = -1 Either $n \leq 0$ or $ka < n$.
IND = -2 Either $m_\ell < 0$ or $m_\ell \geq n$.
IND = -3 Either $m_u < 0$ or $m_u \geq n$.

When an error is detected, the routine immediately terminates.

Usage. After DB1CND terminates, if IND = 0 then DBSLV or DBSLV1 may be called with MO $\neq$ 0 to solve a system of equations. In this case A, ka, n, m_ℓ, m_u and IWK are used by DBSLV and DBSLV1, and must not be modified by the user.

Algorithm. A modification of the Hager procedure by Nicholas J. Higham (University of Manchester, England) is used.

Programming. DB1CND employs the subroutines DONEST, DBSLV, DBSLV1, DBFA, DBSL, DCOPY, DAXPY, DSCAL, and DSWAP, and the functions DB1NRM, DDOT, DASUM, and IDAMAX. DB1CND was written by A. H. Morris. DONEST is the double precision form of the subroutine SONEST, written by N. J. Higham and modified by A. H. Morris.

References.

(1) Higham, N.J., "FORTRAN codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," *ACM Trans. Math Software* 14 (1988), pp. 381-396.

(2) _____,"Algorithm 674: "FORTRAN codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," ***ACM Trans. Math Software*** **15** (1989), p. 168.

SOLUTION OF BANDED SYSTEMS OF COMPLEX LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ complex matrix stored in band form and b a complex column vector of dimension n. The subroutine CBSLV is available for solving the system of equations $Ax = b$, and the subroutine CBSLV1 is available for solving the transposed system of equations $A^t x = b$. On an initial call to either routine, partial pivot Gauss elimination is first employed to obtain an LU decomposition of A, and then the equations are solved. CBSLV and CBSLV1 always generate the same LU decomposition of A. After the decomposition is obtained on an initial call to CBSLV or CBSLV1, either routine may be called to solve a new system of equations $Ax = r$ or $A^t x = r$ without having to redecompose the matrix A.

CALL CBSLV(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)
CALL CBSLV1(MO,A, ka, n, m_ℓ, m_u, X,IWK,IND)

CBSLV is called for solving $Ax = b$ and CBSLV1 is called for solving $A^t x = b$. The argument m_ℓ is the number of diagonals below the main diagonal of A containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. It is assumed that $n \geq 1$, $0 \leq m_\ell < n$, and $0 \leq m_u < n$. MO is an input argument which specifies if CBSLV or CBSLV1 is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a complex array of dimension $ka \times m$ where $ka \geq n$ and $m \geq 2m_\ell + m_u + 1$. On input, the first $m_\ell + m_u + 1$ columns of the array contain the matrix A in band form. When the routine terminates, the array A will contain the upper triangular matrix U of the LU decomposition and the multipliers which were used to obtain it.

X is a complex array of dimension n or larger. On input, X contains the vector b. On output, X will contain the solution of the system of equations.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the LU decomposition are stored in IWK.

On an initial call to CBSLV or CBSLV1, IND is an integer variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The system of equations was solved.
IND = −1 Either $n \leq 0$ or $ka < n$.
IND = −2 Either $m_\ell < 0$ or $m_\ell \geq n$.
IND = −3 Either $m_u < 0$ or $m_u \geq n$.
IND = k Column k of A has been reduced to a column containing only zeros.

After an initial call to CBSLV or CBSLV1, if IND = 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. CBSLV is called for solving the new set of equations $Ax = b$, and CBSLV1 is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition

obtained on the initial call to CBSLV or CBSLV1 to solve the new set of equations. On input, X contains the new vector b. On output, X will contain the solution to the new set of equations. In this case, IND is not referenced by the routine.

Programming. CBSLV and CBSLV1 employ the subroutines CBFA, CBSL, CAXPY, CSCAL, CSWAP and the functions ICAMAX and CDOTU. CBFA and CBSL are adaptations by A. H. Morris of the subroutines SNBFA and SNBSL, written by E. A. Voorhees (Los Alamos Scientific Laboratory). SNBFA and SNBSL are distributed by the SLATEC library.

STORAGE OF SPARSE MATRICES

A matrix is said to be ***sparse*** if it contains sufficiently many zero elements for it to be worthwhile to use special techniques that avoid storing and operating with the zeros. The scheme adopted for storing a sparse $m \times n$ matrix (a_{ij}) requires three 1-dimensional arrays A, IA, JA. The array A contains the nonzero elements of the matrix, stored row by row. The array JA contains the column numbers of the corresponding elements of the A array; i.e., if $A(k)$ contains (a_{ij}) then $JA(k) = j$. The elements of a row of the matrix may be given in any order in A.

IA is an array containing $m+1$ integers which specify where the rows of the matrix are stored in A. For $i \leq m$, $IA(i)$ is the index of the location in A where the i^{th} row information begins. It is assumed that the rows are stored sequentially; i.e., that $IA(1) \leq \cdots \leq IA(m)$. $IA(m+1)$ is set so that $IA(m+1) - IA(1) =$ the number of elements stored in A. For $i \leq m$, if $IA(i) < IA(i+1)$ then $A(\ell)$ is the first entry of the i^{th} row of the matrix in A where $\ell = IA(i)$. Otherwise, if $IA(i) = IA(i+1)$ then no entries for the i^{th} row of the matrix are stored in A. This can occur only if the i^{th} row of the matrix consists entirely of zeros. If this occurs then the i^{th} row is called a ***null row*** of A. For any $i \leq m$, $IA(i+1) - IA(i)$ is the number of entries for the i^{th} row of the matrix that are stored in A. For convenience, $IA(i+1) - IA(i)$ is called the ***length*** of the i^{th} row.

Example. The matrix

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 & a_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{37} & a_{38} \\ 0 & 0 & a_{43} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

can be stored as follows:

A:	a_{11}	a_{18}	a_{12}	a_{37}	a_{38}	a_{43}
JA:	1	8	2	7	8	3
IA:	1	4	4	6	7	

The storage of the elements a_{11} a_{12} a_{18} in the order a_{11} a_{18} a_{12} is permissable. The elements of a row of the matrix may be given in any order desired.

Remark. It is not required that each a_{ij} in A be nonzero.

CONVERSION OF SPARSE MATRICES TO AND FROM THE STANDARD FORMAT

The following subroutines permit one to convert sparse matrices to and from the standard format.

CALL CVRS$(A, ka, m, n, B, IB, JB$,NUM,IERR)
CALL CVDS$(A, ka, m, n, B, IB, JB$,NUM,IERR)
CALL CVCS$(A, ka, m, n, B, IB, JB$,NUM,IERR)

A is an $m \times n$ matrix stored in the standard format. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that A is to be stored in sparse form in the arrays B, IB, JB. CVRS is used if A is a real matrix and B a real array, CVDS is used if A is a double precision matrix and B a double precision array, and CVCS is used if A is a complex matrix and B a complex array.

The input argument NUM is the estimated maximum number of elements that will appear in B and JB. It is assumed that B and JB are of dimension $\max\{1, \text{NUM}\}$ and that IB is of dimension $m + 1$. IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for B and JB, then A is stored in B, IB, JB and IERR is assigned the value 0.

Error Return. If it is found that there is not sufficient storage in B and JB for the i^{th} row of A, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A will have been stored in B and JB, and $IB(1), \ldots, IB(i)$ will contain the appropriate row locations.

Remark. No zero elements of A are stored in B, and the elements of each row of B are ordered so that the column indices of the elements of the row are in ascending order.

Example. If A is the $m \times n$ zero matrix then NUM can be set to 0. In this case the result will be $IB(1) = \cdots = IB(m + 1) = 1$.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris.

CALL CVSR(A, IA, JA, B, kb, m, n)
CALL CVSD(A, IA, JA, B, kb, m, n)
CALL CVSC(A, IA, JA, B, kb, m, n)

A is an $m \times n$ sparse matrix stored in the arrays A, IA, JA, and B is a 2-dimensional array of dimension $kb \times n$ where $kb \geq m$. CVSR is used if A and B are real arrays, CVSD is used if A and B are double precision arrays, and CVSC is used if A and B are complex arrays. When the routine is called, the matrix A is stored in the array B in the standard format.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris.

CONVERSION OF SPARSE REAL MATRICES TO AND FROM DOUBLE PRECISION FORM

The following subroutines are available for converting sparse real matrices to and from double precision form.

CALL SCVRD$(A, IA, JA, B, IB, JB, m)$

A is a sparse real matrix stored in the arrays A, IA, JA. A is a real array and B a double precision array. If A and JA contain k elements then it is assumed that B and JB are arrays of dimension k. It is also assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m+1$. SCVRD stores the matrix A in double precision form in B, IB, JB.

Remarks.

(1) No zero elements of A are stored in B.
(2) It is assumed that the arrays IA, JA and IB, JB reference different storage areas.

Programmer. A. H. Morris.

CALL SCVDR$(A, IA, JA, B, IB, JB, m)$

A is a sparse double precision matrix stored in the arrays A, IA, JA. A is a double precision array and B a real array. If A and JA contain k elements then it is assumed that B and JB are arrays of dimension k. It is also assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m+1$. SCVDR stores the matrix A in single precision form in B, IB, JB.

Remarks.

(1) No zero elements of A are stored in B.
(2) It is assumed that the arrays IA, JA and IB, JB reference different storage areas.

Programmer. A. H. Morris.

THE REAL AND IMAGINARY PARTS OF A SPARSE COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\mathrm{Re}(A) = (\mathrm{Re}(a_{ij}))$ and $\mathrm{Im}(A) = (\mathrm{Im}(a_{ij}))$ denote the real and imaginary parts of A. If the matrix A is stored in sparse form, then the following subroutines are available for obtaining $\mathrm{Re}(A)$ and $\mathrm{Im}(A)$ in sparse form.

CALL CSREAL$(A, IA, JA, B, IB, JB, m)$

A is a sparse complex matrix stored in the arrays A, IA, JA. A is a complex array and B a real array. If A and JA contain k elements then it is assumed that B and JB are arrays of dimension k. It is also assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m + 1$. CSREAL stores $\mathrm{Re}(A)$ in sparse form in B, IB, JB.

Remarks.

(1) No zero elements of $\mathrm{Re}(A)$ are stored in B.
(2) It is assumed that the arrays IA, JA and IB, JB reference different storage areas.

Programmer. A. H. Morris.

CALL CSIMAG$(A, IA, JA, B, IB, JB, m)$

A is a sparse complex matrix stored in the arrays A, IA, JA. A is a complex array and B a real array. If A and JA contain k elements then it is assumed that B and JB are arrays of dimension k. It is also assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m + 1$. CSIMAG stores $\mathrm{Im}(A)$ in sparse form in B, IB, JB.

Remarks.

(1) No zero elements of $\mathrm{Im}(A)$ are stored in B.
(2) It is assumed that the arrays IA, JA and IB, JB reference different storage areas.

Programmer. A. H. Morris.

COMPUTING A + Bi FOR SPARSE REAL MATRICES A AND B

Given the real $m \times n$ matrices A and B stored in sparse form. Then the subroutine SCVRC is available for obtaining the complex matrix $A + Bi$ where $i = \sqrt{-1}$.

CALL SCVRC(A,IA,JA,B,IB,JB,C,IC,JC,m,n,NUM,WK,IERR)

A and B are real $m \times n$ matrices stored in the arrays A,IA,JA and B,IB,JB. It is assumed that $A + Bi$ is to be stored in sparse form in the arrays C,IC,JC. A and B are real arrays and C a complex array. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are arrays of dimension $\max\{1, \text{NUM}\}$ and that IC is an array of dimension $m + 1$.

WK is a real array of dimension n or larger that is a work space for the routine.

IERR is an integer variable. When SCVRC is called, if NUM specifies sufficient storage for C and JC then $A + Bi$ is stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the k^{th} row of $A + Bi$, then IERR is set to k and the routine terminates. In this case, if $k > 1$ then the first $k - 1$ rows of $A + Bi$ will have been stored in C and JC. Also $IC(1), \ldots, IC(k)$ will contain the appropriate row locations.

Remark. No zeros are stored in C.

Programmer. A. H. Morris.

COPYING SPARSE MATRICES

The following subroutines are available for copying sparse matrices.

CALL RSCOPY(A,IA,JA,B,IB,JB,m)
CALL DSCOPY(A,IA,JA,B,IB,JB,m)
CALL CSCOPY(A,IA,JA,B,IB,JB,m)

RSCOPY is used if A and B are real arrays, DSCOPY is used if A and B are double precision arrays, and CSCOPY is used if A and B are complex arrays.

A is a sparse matrix stored in the arrays A, IA,JA. If A and JA contain k elements then it is assumed that B and JB are arrays of dimension k. It is also assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m + 1$. The routine copies the matrix A and stores the copy in B,IB,JB.

Remarks.

(1) No zero elements of A are stored in B.
(2) It is assumed that the arrays A, IA, JA and B, IB,JB reference different storage areas.

Programmer. A. H. Morris.

COMPUTING CONJUGATES OF SPARSE COMPLEX MATRICES

If $A = (a_{ij})$ is a complex matrix stored in sparse form, then the following subroutine is available for computing the conjugate matrix $\bar{A} = (\bar{a}_{ij})$.

CALL SCONJ$(A, IA, JA, B, IB, JB, m)$

It is assumed that the sparse complex matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k. A and B are complex arrays. It is assumed that the matrix A has $m \geq 1$ rows and that IB is an array of dimension $m + 1$. When the routine is called, the conjugate matrix $\bar{A}$ is stored in B, IB, JB.

Remark. The user may let $B = A$, $IB = IA$, and $JB = JA$ when $IA(1) = 1$.

Programmer. A. H. Morris.

TRANSPOSING SPARSE REAL MATRICES

The subroutines RPOSE and RPOSE1 are available for transposing a sparse $m \times n$ real matrix A. RPOSE1 is more general than RPOSE. For any permutation $\pi = \{i_1, \ldots, i_m\}$ of $\{1, \ldots, m\}$ let P denote the corresponding $m \times m$ permutation matrix. Then RPOSE1 computes the matrix $(PA)^t$.

CALL RPOSE$(A, IA, JA, B, IB, JB, m, n)$

It is assumed that the sparse matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When RPOSE is called, the transpose A^t is stored in B, IB, JB.

Remarks. RPOSE orders the elements of each row of A^t so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

CALL RPOSE1$(\pi, A, IA, JA, B, IB, JB, m, n)$

It is assumed that π is an integer array of dimension m containing the data $\{i_1, \ldots, i_m\}$, and that the sparse matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When RPOSE1 is called, $(PA)^t$ is computed and the results are stored in B, IB, JB.

Remarks. RPOSE1 orders the elements of each row of $(PA)^t$ so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

TRANSPOSING SPARSE DOUBLE PRECISION MATRICES

The subroutines DPOSE and DPOSE1 are available for transposing a sparse $m \times n$ double precision matrix A. DPOSE1 is more general than DPOSE. For any permutation $\pi = \{i_1, \ldots, i_m\}$ of $\{1, \ldots, m\}$ let P denote the corresponding $m \times m$ permutation matrix. Then DPOSE1 computes the matrix $(PA)^t$.

CALL DPOSE$(A, IA, JA, B, IB, JB, m, n)$

It is assumed that the sparse matrix A is stored in the arrays A, IA, JA. A and B are double precision arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When DPOSE is called, the transpose A^t is stored in B, IB, JB.

Remarks DPOSE orders the elements of each row of A^t so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

CALL DPOSE1$(\pi, A, IA, JA, B, IB, JB, m, n)$

It is assumed that π is an integer array of dimension m containing the data $\{i_1, \ldots, i_m\}$, and that the sparse matrix A is stored in the arrays A, IA, JA. A and B are double precision arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When DPOSE1 is called, $(PA)^t$ is computed and the results are stored in B, IB, JB.

Remarks. DPOSE1 orders the elements of each row of $(PA)^t$ so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

TRANSPOSING SPARSE COMPLEX MATRICES

The subroutines CPOSE and CPOSE1 are available for transposing a sparse $m \times n$ complex matrix A. CPOSE1 is more general than CPOSE. For any permutation $\pi = \{i_1, \ldots, i_m\}$ of $\{1, \ldots, m\}$ let P denote the corresponding $m \times m$ permutation matrix. Then CPOSE1 computes the matrix $(PA)^t$.

CALL CPOSE$(A, IA, JA, B, IB, JB, m, n)$

It is assumed that the sparse matrix A is stored in the arrays A, IA, JA. A and B are complex arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When CPOSE is called, the transpose A^t is stored in B, IB, JB.

Remarks. CPOSE orders the elements of each row of A^t so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

CALL CPOSE1$(\pi, A, IA, JA, B, IB, JB, m, n)$

It is assumed that π is an integer array of dimension m containing the data $\{i_1, \ldots, i_m\}$, and that the sparse matrix A is stored in the arrays A, IA, JA. A and B are complex arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n+1$. When CPOSE1 is called, $(PA)^t$ is computed and the results are stored in B, IB, JB.

Remarks. CPOSE1 orders the elements of each row of $(PA)^t$ so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris.

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software*** **4** (1978), pp. 250–269.

ADDITION OF SPARSE MATRICES

The following subroutines are available for adding sparse matrices.

CALL SADD$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)
CALL DSADD$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)
CALL CSADD$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)

A and B are sparse $m \times n$ matrices stored in the arrays A,IA,JA and B,IB,JB. It is assumed that $A + B$ is to be stored in sparse form in the arrays C,IC,JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are arrays of dimension $\max\{1, \text{NUM}\}$ and that IC is an array of dimension $m + 1$.

SADD is used if A,B,C, and WK are real arrays, DSADD is used if A,B,C, and WK are double precision arrays, and CSADD is used if A,B,C, and WK are complex arrays.

WK is an array of dimension n or larger that is a work space for the routine.

IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for C and JC then $A + B$ is computed and stored in C,IC,JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of $A + B$, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of $A + B$ will have been computed and stored in C and JC. Also $IC(1), \ldots, IC(i)$ will contain the appropriate row locations.

Remarks.

(1) No zeros are stored in C.
(2) It is assumed that C, IC, JC reference different storage areas than A, IA, JA and B, IB, JB.

Programmer. A. H. Morris

SUBTRACTION OF SPARSE MATRICES

The following subroutines are available for subtracting sparse matrices.

CALL SSUBT$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)
CALL DSSUBT$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)
CALL CSSUBT$(A, IA, JA, B, IB, JB, C, IC, JC, m, n$,NUM,WK,IERR)

A and B are sparse $m \times n$ matrices stored in the arrays A,IA,JA and B,IB,JB. It is assumed that $A - B$ is to be stored in sparse form in the arrays C,IC,JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are arrays of dimension $\max\{1, \text{NUM}\}$ and that IC is an array of dimension $m + 1$.

SSUBT is used if A,B,C, and WK are real arrays, DSSUBT is used if A,B,C, and WK are double precision arrays, and CSSUBT is used if A,B,C, and WK are complex arrays.

WK is an array of dimension n or larger that is a work space for the routine.

IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for C and JC then $A - B$ is computed and stored in C,IC,JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of $A - B$, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of $A - B$ will have been computed and stored in C and JC. Also $IC(1), \ldots, C(i)$ will contain the appropriate row locations.

Remarks.

(1) No zeros are stored in C.
(2) It is assumed that C, IC, JC reference different storage areas than A, IA, JA and B, IB, JB.

Programmer. A. H. Morris.

MULTIPLICATION OF SPARSE MATRICES

The following subroutines are available for multiplying sparse matrices.

CALL SPROD$(A, IA, JA, B, IB, JB, C, IC, JC, \ell, m, n$,NUM,WK,IERR)
CALL DSPROD$(A, IA, JA, B, IB, JB, C, IC, JC, \ell, m, n$,NUM,WK,IERR)
CALL CSPROD$(A, IA, JA, B, IB, JB, C, IC, JC, \ell, m, n$,NUM,WK,IERR)

A is a sparse $\ell \times m$ matrix stored in the arrays A,IA,JA, and B a sparse $m \times n$ matrix stored in the arrays B, IB, JB. It is assumed that AB is to be stored in sparse form in the arrays C,IC,JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are arrays of dimension $\max\{1, \text{NUM}\}$ and that IC is an array of dimension $\ell + 1$.

SPROD is used if A,B,C, and WK are real arrays, DSPROD is used if A,B,C, and WK are double precision arrays, and CSPROD is used if A,B,C, and WK are complex arrays.

WK is an array of dimension n or larger that is a work space for the routine.

IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for C and JC then AB is computed and stored in C,IC,JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of AB, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of AB will have been computed and stored in C and JC. Also $IC(1), \ldots, IC(i)$ will contain the appropriate row locations.

Remarks.

(1) No zeros are stored in C.
(2) It is assumed that C, IC, JC reference different storage areas than A, IA, JA and B, IB, JB.

Programmer. A. H. Morris.

PRODUCT OF A REAL SPARSE MATRIX AND VECTOR

Let A be a real $m \times n$ sparse matrix stored in the arrays A, IA, JA. Then the following subroutines are available for multiplying A with a real vector.

CALL MVPRD(m, n, A, IA, JA, x, y)

The argument x is a column vector of dimension n and y an array of dimension m. When MVPRD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL MVPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are column vectors of dimension n and m respectively. When MVPRD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL MTPRD(m, n, A, IA, JA, x, y)

The argument x is a row vector of dimension m and y an array of dimension n. When MTPRD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL MTPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are row vectors of dimension m and n respectively. When MTPRD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

PRODUCT OF A DOUBLE PRECISION SPARSE MATRIX AND VECTOR

Let A be a double precision $m \times n$ sparse matrix stored in the arrays A, IA, JA. Then the following subroutines are available for multiplying A with a double precision vector.

CALL DVPRD(m, n, A, IA, JA, x, y)

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are double precision arrays. When DVPRD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DVPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are column vectors of dimension n and m respectively. A, x, y are double precision arrays. When DVPRD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DTPRD(m, n, A, IA, JA, x, y)

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are double precision arrays. When DTPRD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL DTPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are row vectors of dimension m and n respectively. A, x, y are double precision arrays. When DTPRD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

PRODUCT OF A COMPLEX SPARSE MATRIX AND VECTOR

Let A be a complex $m \times n$ sparse matrix stored in the arrays A, IA, JA. Then the following subroutines are available for multiplying A with a complex vector.

CALL CVPRD(m, n, A, IA, JA, x, y)

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are complex arrays. When CVPRD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CVPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are column vectors of dimension n and m respectively. A, x, y are complex arrays. When CVPRD1 is called, $Ax + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CTPRD(m, n, A, IA, JA, x, y)

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are complex arrays. When CTPRD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

CALL CTPRD1(m, n, A, IA, JA, x, y)

The arguments x and y are row vectors of dimension m and n respectively. A, x, y are complex arrays. When CTPRD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris.

L_1 NORM OF A SPARSE REAL MATRIX

If $A = (a_{ij})$ is a sparse real matrix then the following subroutines are available for computing the ℓ_1 norm $||A||_1 = \max_j \Sigma_i |a_{ij}|$ of A.

CALL S1NRM(A,IA,JA,m,n,ANORM,WK)
CALL DS1NRM(A,IA,JA,m,n,ANORM,WK)

S1NRM is used if A and WK are real arrays and ANORM a real variable, and DS1NRM is used if A and WK are double precision arrays and ANORM a double precision variable.

A is a sparse $m \times n$ matrix stored in the arrays A, IA, JA. When the routine is called, the variable ANORM is assigned the value $||A||_1$.

WK is an array of dimension n or larger that is a work space for the routine.

Programmer. A. H. Morris.

L_∞ NORM OF A SPARSE REAL MATRIX

If $A = (a_{ij})$ is a sparse real $m \times n$ matrix then the following functions are available for computing the ℓ_∞ norm $||A||_\infty = \max_i \Sigma_j |a_{ij}|$ of A.

SNRM(A,IA,JA,m,n)
DSNRM(A,IA,JA,m,n)

SNRM is used if A is a real array and DSNRM is used if A is a double precision array. SNRM is a real function and DSNRM a double precision function.

A is a sparse $m \times n$ matrix stored in the arrays A, IA, JA. When either of these functions is called then the function is assigned the value $||A||_\infty$.

Remark. DSNRM must be declared to be of type DOUBLE PRECISION in the calling program.

Programmer. A. H. Morris.

ORDERING THE ROWS OF A SPARSE MATRIX BY INCREASING LENGTH

Let A be a sparse $m \times n$ matrix stored in the arrays A, IA, JA. The following subroutine is available for ordering the rows of the matrix by increasing length.

CALL SPORD(m, n, IA, R,IWK)

R is an integer array of dimension m. When SPORD is called, the rows of the matrix are ordered by increasing length. The row ordering is given in R.

IWK is an integer array of dimension $m+n+1$ or larger that is used for a work space.

Remark. If rows $i_1, \ldots, i_k$ are the rows of length ℓ, then the indices $i_1, \ldots, i_k$ are listed in R in increasing sequence.

Programmer. A. H. Morris.

REORDERING SPARSE MATRICES INTO BLOCK TRIANGULAR FORM

Let A be a sparse $n \times n$ matrix stored in the arrays A, IA, JA. Then the subroutine BLKORD is available for reordering the rows and columns of A so that one has a lower block triangular matrix

$$(*) \qquad \begin{pmatrix} A_{11} & & & 0 \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{pmatrix}$$

where the blocks A_{ii} are square and cannot themselves be reordered into lower block triangular form.

CALL BLKORD(n, IA, JA, R, C, IB, k,IWK,IERR)

R and C are integer arrays of dimension n, and IERR is an integer variable. When BLKORD is called, the rows of the matrix are first ordered so that the main diagonal contains a maximum number of nonzeros. After this is done then IERR = the number of zeros that appear on the diagonal. If IERR = 0 then the rows and columns of the matrix are ordered into block triangular form $(*)$. The row ordering is given in R and the column ordering is given in C.

IB is an integer array of dimension n and k is an integer variable. When the matrix has been ordered into block triangular form $(*)$ then k = the number of blocks A_{ii}. Also $IB(i)$ = the row number in the block triangular matrix of the beginning of block A_{ii} $(i = 1, \ldots, k)$.

IWK is an integer array of dimension $5n$ or larger that is used for a work space by the routine.

Error Return. If IERR $\neq 0$ then the routine terminates. In this case, R contains the row ordering that gives the main diagonal with the maximum number of nonzeros.

Remark. IA, JA, and n are not modified by the routine.

Programming. BLKORD employs the subroutines MC21A, MC21B, MC13D, MC13E designed by I. S. Duff and J. K. Reid (AERE Harwell,England).

References.

(1) Duff, I. S., "On Algorithms for Obtaining a Maximum Transversal," ***ACM Trans. Math Software*** **7** (1981), pp.315–330.

(2) Duff, I. S. and Reid, J. K., "An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix," ***ACM Trans. Math Software*** **4** (1978), pp. 137–147.

SOLUTION OF SPARSE SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ sparse real matrix stored in the arrays A, IA, JA and let b be a real column vector of dimension n. The subroutines SPSLV and RSLV are available for solving the system of equations $Ax = b$, and the subroutine TSLV is available for solving the transposed system of equations $A^t x = b$. These routines employ partial pivot gauss elimination with column interchanges to first obtain an LU decomposition of A. If SPSLV is called then only the off-diagonal nonzero elements of U are stored, and then the equations are solved. However, if RSLV or TSLV is called then the off-diagonal nonzero elements of both L and U are stored. Thus RSLV and TSLV will frequently require at least double the amount of storage needed by SPSLV, but they can be recalled to solve other systems of equations $Ax = r$ and $A^t x = r$ without having to redecompose the matrix A. Moreover, since RSLV and TSLV will always generate the same LU decomposition of A, RSLV can be called to decompose A and solve a system of equations $Ax = b$, and then TSLV can be called to solve a transposed system of equations $A^t x = r$ using the decomposition obtained by RSLV.

CALL SPSLV(n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

It is assumed that $n \geq 1$ and that X is an array of dimension n. The solution of the system of equations $Ax = b$ is computed and stored in X. A,IA, JA and b are not modified by the routine.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of a subroutine such as SPSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routine, and MAX is an input argument. When SPSLV is called, an LU decomposition of A is first obtained where U is a unit upper triangular matrix. The off-diagonal portion of U is stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of U that might be nonzero and have to be stored (MAX $= n(n-1)/2$). IWK is an integer array of dimension $3n +$ MAX $+ 2$ or larger, and WK is a real array of dimension $n +$ MAX or larger.

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	$Ax = b$ was solved. IERR $= \max\{1, m\}$ where m is the number of off-diagonal nonzero elements of U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n - k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n - k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n - k$	Row k of the upper triangular matrix exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

Remarks. The amount of storage MAX depends critically on the row ordering given in R. If it is suspected that the rows and columns of A can be reordered so that one has a lower block triangular matrix

$$\begin{pmatrix} A_{11} & & & 0 \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{k1} & A_{k2} & \dots & A_{kk} \end{pmatrix}$$

then the subroutine BLKORD should first be tried. This subroutine will specify an ordering for lower block triangular form if one exists. However, if such an ordering does not exist and one is uncertain how to order the rows, then the row ordering given by the subroutine SPORD frequently yields good results. In any case, the selection of an initial column ordering C is never bothersome since partial pivoting with column interchanges is performed. The initial ordering $C(i) = i$ $(i = 1, \dots, n)$ always suffices.

Programming. SPSLV is a modification by A. H. Morris of the subroutine NSPIV. SPSLV employs the subroutine NSPIV1. NSPIV and NSPIV1 were written by Andrew H. Sherman (University of Texas at Austin).

Reference. Sherman, Andrew H.,"Algorithms for Sparse Gaussian Elimination with Partial Pivoting," *ACM Trans. Math Software* 4 (1978), pp.330–338.

CALL RSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)
CALL TSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

RSLV is called for solving $Ax = b$, and TSLV is called for solving $A^t x = b$. MO is an input argument which specifies if RSLV or TSLV is being called for the first time. On an initial call, MO $= 0$ and we have the following setup:

It is assumed that $n > 1$ and that X is an array of dimension n. The solution of the system of equations is stored in X. A, IA, JA are not modified by the routines. X and b may share the same storage area. If X is a separate storage area then b is not modified by the routines.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of subroutines such as RSLV and TSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routines, and MAX is an input argument. On an initial call to RSLV or TSLV, an LU decomposition of A is first obtained where L is a lower triangular matrix and U a unit upper triangular matrix. The off-diagonal portions of L and U are stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of L and U that might be nonzero and have to be stored (MAX $\leq n(n-1)$). IWK is an integer array of dimension $4n + \text{MAX} + 2$ or larger, and WK is a real array of dimension $2n + \text{MAX}$ or larger.

On an initial call to RSLV or TSLV, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	The system of equations was solved. IERR=max{1, m} where m is the total number of off-diagonal nonzero elements of L and U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n-k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n-k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n-k$	Row k of L or U exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

After an initial call to RSLV or TSLV, if IERR > 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. RSLV is called for solving the new set of equations $Ax = b$, and TSLV is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to RSLV or TSLV to solve the new system of equations. The solution is stored in X. As before, X and b may share the same storage area. If MO $\neq 0$ then only n, R, C, IWK, and WK are used. A, IA, JA, MAX, and IERR are not referenced by the routine.

Note. The remarks concerning the ordering of the rows and columns of A when SPSLV is

used hold also for RSLV and TSLV.

Programming. RSLV calls the subroutines RSLV1 and SPLU, and TSLV calls the subroutines TSLV1 and SPLU. These routines were written by A. H. Morris.

COMPUTATION OF THE CONDITION NUMBER OF A REAL SPARSE MATRIX

If A is a real $n \times n$ sparse matrix then the following subroutine is available for estimating the ℓ_1 condition number $\text{cond}_1(A)$ of A.

CALL S1CND(n, A, IA, JA, R, C,MAX,COND,IWK,WK,IERR)

A is a sparse matrix stored in the arrays A, IA, JA. The arguments R, C, MAX, IWK, and WK are the same arguments used for the subroutines RSLV and TSLV, the only exceptions being that IWK now is an array of dimension $5n$ + MAX + 2 or larger, and WK is an array of dimension $4n$ + MAX or larger.

COND is a real variable. When S1CND is called, then the subroutine RSLV is first invoked to obtain an LU decomposition of A, which is stored in the arrays IWK and WK. Then COND is assigned the value 0 if A is singular and an approximation of the condition number $\text{cond}_1(A)$ if A is nonsingular.

IERR is an integer variable that reports the status of the results. When S1CND terminates, IERR has one of the following values:

IERR > 0	A is nonsingular. COND = an approximation of the condition number of A, and IERR = $\max\{1, m\}$ where m is the number of off-diagonal nonzero elements of L and U.
IERR = 0	A is singular and COND = 0.
IERR = $-k$	Row R(k) of A has a duplicate entry.
IERR < $-n$	Row R(k) of L or U exceeds storage where IERR = $-(n + k)$. MAX must be increased.

When an error is detected, the routine immediately terminates.

Usage. After S1CND terminates, if IERR > 0 then RSLV or TSLV may be called with MO $\neq 0$ to solve a system of equations. In this case n, R, C,WK, and IWK are used by RSLV and TSLV, and must not be modified by the user.

Algorithm. A modification of the Hager procedure by Nicholas J. Higham (University of Manchester, England) is used.

Programming. S1CND employs the subroutines S1NRM, SONEST, SCOPY, RSLV, TSLV, RSLV1, TSLV1, and SPLU, and the functions SASUM and ISAMAX. S1CND was written by A. H. Morris. SONEST was written by N. J. Higham and modified by A. H. Morris.

References.

(1) Higham, N.J.,"FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," *ACM Trans. Math Software* **14** (1988), pp. 381–396.

(2) ___ ___,"Algorithm 674: FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," ***ACM Trans. Math Software*** **15** (1989), p. 168.

DOUBLE PRECISION SOLUTION OF SPARSE SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ sparse double precision matrix stored in the arrays A, IA, JA and let b be a double precision column vector of dimension n. The subroutines DSPSLV and DSLV are available for solving the system of equations $Ax = b$, and the subroutine DTSLV is available for solving the transposed system of equations $A^t x = b$. These routines employ partial pivot Gauss elimination with column interchanges to first obtain an LU decomposition of A. If DSPSLV is called then only the off-diagonal nonzero elements of U are stored, and then the equations are solved. However, if DSLV or DTSLV is called then the off-diagonal nonzero elements of both L and U are stored. Thus DSLV and DTSLV will frequently require at least double the amount of storage needed by DSPSLV, but they can be recalled to solve other systems of equations $Ax = r$ and $A^t x = r$ without having to redecompose the matrix A. Moreover, since DSLV and DTSLV will always generate the same LU decomposition of A, DSLV can be called to decompose A and solve a system of equations $Ax = b$, and then DTSLV can be called to solve a transposed system of equations $A^t x = r$ using the decomposition obtained by DSLV.

CALL DSPSLV(n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

A , b, and X are double precision arrays. It is assumed that $n \geq 1$ and that X is an array of dimension n. The solution of the system of equations $Ax = b$ is computed and stored in X. A,IA, JA and b are not modified by the routine.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of a subroutine such as DSPSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routine, and MAX is an input argument. When DSPSLV is called, an LU decomposition of A is first obtained where U is a unit upper triangular matrix. The off-diagonal portion of U is stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of U that might be nonzero and have to be stored (MAX $\leq n(n-1)/2$). IWK is an integer array of dimension $3n$ + MAX + 2 or larger, and WK is a double precision array of dimension n + MAX or larger.

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	$Ax = b$ was solved. IERR $= \max\{1, m\}$ where m is the number of off-diagonal nonzero elements of U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n - k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n - k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n - k$	Row k of the upper triangular matrix exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

Remarks. The amount of storage MAX depends critically on the row ordering given in R. If it is suspected that the rows and columns of A can be reordered so that one has a lower block triangular matrix

$$\begin{pmatrix} A_{11} & & & 0 \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{pmatrix}$$

then the subroutine BLKORD should first be tried. This subroutine will specify an ordering for lower block triangular form if one exists. However, if such an ordering does not exist and one is uncertain how to order the rows, then the row ordering given by the subroutine SPORD frequently yields good results. In any case, the selection of an initial column ordering C is never bothersome since partial pivoting with column interchanges is performed. The initial ordering $C(i) = i$ $(i = 1, \ldots, n)$ always suffices.

Programming. DSPSLV an adaptation by A. H. Morris of the subroutine NSPIV written by Andrew H. Sherman (University of Texas at Austin). DSPSLV employs the subroutine DNSPIV.

Reference. Sherman, Andrew H.,"Algorithms for Sparse Gaussian Elimination with Partial Pivoting," ***ACM Trans. Math Software*** **4** (1978), pp.330–338.

CALL DSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)
CALL DTSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

DSLV is called for solving $Ax = b$, and DTSLV is called for solving $A^t x = b$. MO is an input argument which specifies if DSLV or DTSLV is being called for the first time. On an initial call, MO $= 0$ and we have the following setup:

A, b, and X are double precision arrays. It is assumed that $n \geq 1$ and that X is an array of dimension n. The solution of the system of equations is stored in X. A, IA, JA are not modified by the routines. X and b may share the same storage area. If X is a separate storage area then b is not modified by the routines.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of subroutines such as DSLV and DTSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routines, and MAX is an input argument. On an initial call to DSLV or DTSLV, an LU decomposition of A is first obtained where L is a lower triangular matrix and U a unit upper triangular matrix. The off-diagonal portions of L and U are stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of L and U that might be nonzero and have to be stored (MAX $\leq n(n-1)$). IWK is an integer array of dimension $4n + \text{MAX} + 2$ or larger, and WK is a double precision array of dimension $2n + \text{MAX}$ or larger.

On an initial call to DSLV or DTSLV, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	The system of equations was solved. IERR=max$\{1, m\}$ where m is the total number of off-diagonal nonzero elements of L and U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n - k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n - k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n - k$	Row k of L or U exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

After an initial call to DSLV or DTSLV, if IERR > 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. DSLV is called for solving the new set of equations $Ax = b$, and DTSLV is called for solving the new set of equations $A^t x = b$. The routine emplcys the LU decomposition obtained on the initial call to DSLV or DTSLV to solve the new system of equations. The solution is stored in X. As before, X and b may share the same storage area. If MO $\neq 0$ then only n, R, C, IWK, and WK are used. A, IA, JA, MAX, and IERR are not referenced by the routine.

Note. The remarks concerning the ordering of the rows and columns of A when DSPSLV

is used hold also for DSLV and DTSLV.

Programming. DSLV calls the subroutines DSLV1 and DSPLU, and DTSLV calls the subroutines DTSLV1 and DSPLU. These routines were written by A. H. Morris.

COMPUTATION OF THE CONDITION NUMBER OF A DOUBLE PRECISION SPARSE MATRIX

If A is a double precision $n \times n$ sparse matrix then the following subroutine is available for estimating the ℓ_1 condition number $\text{cond}_1(A)$ of A.

CALL DS1CND(n, A, IA, JA, R, C,MAX,COND,IWK,WK,IERR)

A is a sparse matrix stored in the arrays A, IA, JA where A is a double precision array. The arguments R, C, MAX, IWK, and WK are the same arguments used for the subroutines DSLV and DTSLV, the only exceptions being that IWK now is an array of dimension $5n$ + MAX + 2 or larger, and WK is an array of dimension $4n$ + MAX or larger.

COND is a double precision variable. When DS1CND is called, then the subroutine DSLV is first invoked to obtain an LU decomposition of A, which is stored in the arrays IWK and WK. Then COND is assigned the value 0 if A is singular and an approximation of the condition number $\text{cond}_1(A)$ if A is nonsingular.

IERR is an integer variable that reports the status of the results. When DS1CND terminates, IERR has one of the following values:

IERR $>$ 0 A is nonsingular. COND = an approximation of the condition number of A, and IERR = $\max\{1, m\}$ where m is the number of off-diagonal nonzero elements of L and U.
IERR = 0 A is singular and COND = 0.
IERR = $-k$ Row R(k) of A has a duplicate entry.
IERR $< -n$ Row R(k) of L or U exceeds storage where IERR = $-(n + k)$. MAX must be increased.

When an error is detected, the routine immediately terminates.

Usage. After DS1CND terminates, if IERR > 0 then DSLV or DTSLV may be called with MO $\neq 0$ to solve a system of equations. In this case n, R, C,WK, and IWK are used by DSLV and DTSLV, and must not be modified by the user.

Algorithm. A modification of the Hager procedure by Nicholas J. Higham (University of Manchester, England) is used.

Programming. DS1CND employs the subroutines DS1NRM, DONEST, DCOPY, DSLV, DSLV1, DTSLV, DTSLV1, and DSPLU, and the functions DASUM and IDAMAX. DS1CND was written by A. H. Morris. DONEST is the double precision form of the subroutine SONEST, written by N. J. Higham and modified by A. H. Morris.

References.

(1) Higham, N.J.,"FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," ***ACM Trans. Math Software*** 14 (1988), pp. 381–396.

(2) _____,"Algorithm 674: FORTRAN Codes for Estimating the One-Norm of a Real or Complex Matrix, with Applications to Condition Estimation," ***ACM Trans. Math Software*** **15** (1989), p. 168.

SOLUTION OF SPARSE SYSTEMS OF COMPLEX LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ sparse complex matrix stored in the arrays A, IA, JA and let b be a complex column vector of dimension n. The subroutines CSPSLV and CSLV are available for solving the system of equations $Ax = b$, and the subroutine CTSLV is available for solving the transposed system of equations $A^t x = b$. These routines employ partial pivot Gauss elimination with column interchanges to first obtain an LU decomposition of A. If CSPSLV is called then only the off-diagonal nonzero elements of U are stored, and then the equations are solved. However, if CSLV or CTSLV is called then the off-diagonal nonzero elements of both L and U are stored. Thus CSLV and CTSLV will frequently require at least double the amount of storage needed by CSPSLV, but they can be recalled to solve other systems of equations $Ax = r$ and $A^t x = r$ without having to redecompose the matrix A. Moreover, since CSLV and CTSLV will always generate the same LU decomposition of A, CSLV can be called to decompose A and solve a system of equations $Ax = b$, and then CTSLV can be called to solve a transposed system of equations $A^t x = r$ using the decomposition obtained by CSLV.

CALL CSPSLV(n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

A , b, and X are complex arrays. It is assumed that $n \geq 1$ and that X is an array of dimension n. The solution of the system of equations $Ax = b$ is computed and stored in X. A,IA, JA and b are not modified by the routine.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of a subroutine such as CSPSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routine, and MAX is an input argument. When CSPSLV is called, an LU decomposition of A is first obtained where U is a unit upper triangular matrix. The off-diagonal portion of U is stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of U that might be nonzero and have to be stored (MAX $\leq n(n-1)/2$). IWK is an integer array of dimension $3n$ + MAX + 2 or larger, and WK is a complex array of dimension n + MAX or larger.

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	$Ax = b$ was solved. IERR $= \max\{1, m\}$ where m is the number of off-diagonal nonzero elements of U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n - k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n - k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n - k$	Row k of the upper triangular matrix exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

Remarks. The amount of storage MAX depends critically on the row ordering given in R. If it is suspected that the rows and columns of A can be reordered so that one has a lower block triangular matrix

$$\begin{pmatrix} A_{11} & & & 0 \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{pmatrix}$$

then the subroutine BLKORD should first be tried. This subroutine will specify an ordering for lower block triangular form if one exists. However, if such an ordering does not exist and one is uncertain how to order the rows, then the row ordering given by the subroutine SPORD frequently yields good results. In any case, the selection of an initial column ordering C is never bothersome since partial pivoting with column interchanges is performed. The initial ordering $C(i) = i$ $(i = 1, \ldots, n)$ always suffices.

Programming. CSPSLV is an adaptation by A. H. Morris of the subroutine NSPIV written by Andrew H. Sherman (University of Texas at Austin). CSPSLV employs the subroutine CNSPIV.

Reference. Sherman, Andrew H.,"Algorithms for Sparse Gaussian Elimination with Partial Pivoting," ***ACM Trans. Math Software*** **4** (1978), pp.330-338.

CALL CSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)
CALL CTSLV(MO,n, A, IA, JA, b, R, C,MAX,X,IWK,WK,IERR)

CSLV is called for solving $Ax = b$, and CTSLV is called for solving $A^t x = b$. MO is an input argument which specifies if CSLV or CTSLV is being called for the first time. On an initial call, MO $= 0$ and we have the following setup:

A, b, and X are complex arrays. It is assumed that $n \geq 1$ and that X is an array of dimension n. The solution of the system of equations is stored in X. A, IA, JA are not modified by the routines. X and b may share the same storage area. If X is a separate storage area then b is not modified by the routines.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1, \ldots, i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of subroutines such as CSLV and CTSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1, \ldots, j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routines, and MAX is an input argument. On an initial call to CSLV or CTSLV, an LU decomposition of A is first obtained where L is a lower triangular matrix and U a unit upper triangular matrix. The off-diagonal portions of L and U are stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of L and U that might be nonzero and have to be stored (MAX $\leq n(n-1)$). IWK is an integer array of dimension $4n + \text{MAX} + 2$ or larger, and WK is a complex array of dimension $2n + \text{MAX}$ or larger.

On an initial call to CSLV or CTSLV, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR > 0	The system of equations was solved. IERR$=\max\{1, m\}$ where m is the total number of off-diagonal nonzero elements of L and U.
IERR $= 0$	The argument n is nonpositive.
IERR $= -k$	Row $R(k)$ of A is null.
IERR $= -n - k$	Row $R(k)$ of A has a duplicate entry.
IERR $= -2n - k$	Row $R(k)$ of A has been reduced to a row containing only zeros.
IERR $= -3n - k$	Row k of L or U exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

After an initial call to CSLV or CTSLV, if IERR > 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. CSLV is called for solving the new set of equations $Ax = b$, and CTSLV is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to CSLV or CTSLV to solve the new system of equations. The solution is stored in X. As before, X and b may share the same storage area. If MO $\neq 0$ then only n, R, C, IWK, and WK are used. A, IA, JA, MAX, and IERR are not referenced by the routine.

Note. The remarks concerning the ordering of the rows and columns of A when CSPSLV is used hold also for CSLV and CTSLV.

Programming. CSLV calls the subroutines CSLV1 and CSPLU, and CTSLV calls the subroutines CTSLV1 and CSPLU. These routines were written by A. H. Morris.

COMPUTATION OF EIGENVALUES OF GENERAL REAL MATRICES

The subroutines EIG and EIG1 are available for computing the eigenvalues of real matrices. These routines frequently yield results accurate to 13–14 significant digits. Indeed, for symmetric matrices they may give 2 or more digits better accuracy than the routines designed specifically for symmetric matrices. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur when the matrix is not symmetric. In this case one should not expect more than 7–8 digit accuracy.

CALL EIG(IBAL,A, ka, n, WR, WI,IERR)
CALL EIG1(IBAL,A, ka, n, WR, WI,IERR)

A is a matrix of order $n \geq 1$ and WR,WI are real arrays of dimension n or larger. When EIG or EIG1 is called then the eigenvalues $\lambda_1, \ldots, \lambda_n$ of A are computed. The real parts of the eigenvalues are stored in $WR(1), \ldots, WR(n)$ and the imaginary parts are stored in $WI(1), \ldots, WI(n)$. The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

IBAL and ka are input arguments. The argument ka is the number of rows in the dimension statement for A in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the routines balance A before they compute the eigenvalues. Otherwise, if IBAL $= 0$ then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks.

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy by as much as 5–6 significant digits. Thus it is recommended that balancing be done.

(2) A is destroyed during computation. EIG and EIG1 reduce A to upper Hessenberg form and then apply the QR algorithm to obtain the eigenvalues. They differ only in the choice of transformations used to reduce A to upper Hessenberg form. EIG employs elementary similarity transformations and EIG1 employs orthogonal similarity transformations. In theory the use of orthogonal transformations assures one of a tighter bound on the errors. However, since in practice matrices infrequently arise for which the orthogonal transformations actually generate more accurate results, and since the orthogonal transformations normally require more time than the elementary transformations, therefore EIG is the recommended routine.

Programming. EIG and EIG1 are driver routines for the EISPACK subroutines BALANC, ELMHS0, ORTHES, and HQR. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines - EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF GENERAL REAL MATRICES

The subroutines EIGV and EIGV1 are available for computing the eigenvalues and eigenvectors of real matrices. These routines are extensions of the respective eigenvalue routines EIG and EIG1. Thus all comments made concerning the accuracy of the eigenvalues produced by EIG and EIG1 apply also to EIGV and EIGV1. In particular, EIGV and EIGV1 can frequently yield high precision results for the eigenvalues if they are distinct. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2–3 units of the 14^{th} significant digit, but the components of the corresponding eigenvector are only accurate to 9–10 significant digits. In the case of repeated eigenvalues the situation regarding the eigenvectors is totally unpredictable. The components of such an eigenvector may be correct to 6–7 significant digits, or the eigenvector may not even be an eigenvector! In this case the results should be checked.

CALL EIGV(IBAL, A, ka, n, WR,WI,ZR,ZI,IERR)
CALL EIGV1(IBAL, A, ka, n, WR,WI,ZR,ZI,IERR)

A is a matrix of order $n \geq 1$ and WR, WI are real arrays of dimension n or larger. When EIGV or EIGV1 is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ...,WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

The input argument ka is the number of rows in the dimension statement for A in the calling program. ZR and ZI are real arrays of dimension $ka \times n$. For $j = 1, \ldots, n$ the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1,j), ...,ZR(n, j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1, \ldots, z_n$ are now normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the routines balance A before they compute the eigenvalues and eigenvectors. Otherwise, if IBAL $= 0$ then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks.

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy of the eigenvalues by as much as 5–6 significant digits. When this occurs balancing will normally be needed to obtain the eigenvectors. In general, it is recommended that balancing be done.

(2) A is destroyed during computation. EIGV and EIGV1 both reduce A to upper Hessenberg form, apply the QR algorithm to the Hessenberg matrix to obtain the eigenvalues, and then backsubstitute to generate the eigenvectors. They differ only in the choice of transformations used to reduce A to upper Hessenberg form. EIGV employs elementary similarity transformations and EIGV1 employs orthogonal similarity transformations. In theory the use of orthogonal transformations assures one of a tighter bound on the errors. However, since in practice matrices infrequently arise for which the orthogonal transformations actually generate more accurate results, and since the orthogonal transformations normally require more time than the elementary transformations, therefore EIGV is the recommended routine.

Programming. EIGV and EIGV1 are driver routines for the EISPACK subroutines BALANC, ELMHS0, ORTHES, ELTRN0, ORTRAN, HQR2, and BALBAK. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES OF REAL MATRICES

The subroutine DEIG is available for the double precision computation of the eigenvalues of real matrices. This routine frequently yields results accurate to 26–28 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 13–14 digit accuracy.

CALL DEIG(IBAL, A, ka, n, WR,WI,IERR)

A is a double precision matrix of order $n \geq 1$ and WR, WI are double precision arrays of dimension n or larger. When DEIG is called then the eigenvalues $\lambda_1, \ldots, \lambda_n$ of A are computed. The real parts of the eigenvalues are stored in WR(1), ...,WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

IBAL and ka are input arguments. The argument ka is the number of rows in the dimension statement for A in the calling program. IBAL may be any integer. If IBAL $\neq$ 0 then the routine balances A before it computes the eigenvalues. Otherwise, if IBAL $= 0$ then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks.

(1) A is destroyed during computation.
(2) DEIG is a double precision version of the eigenvalue routine EIG1.

Programming. DEIG is a driver routine for the subroutines DBAL, DORTH, and DHQR. These subroutines are double precision versions of the EISPACK subroutines BALANC, ORTHES, and HQR, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF REAL MATRICES

The subroutine DEIGV is available for the double precision computation of the eigenvalues and eigenvectors of real matrices. This routine frequently yields values for the eigenvalues that are accurate to 26–28 significant digits. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. If the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 13–14 digit accuracy.

CALL DEIGV(IBAL, A, ka, n, WR,WI,ZR,ZI,IERR)

A is a double precision matrix of order $n \geq 1$ and WR, WI are double precision arrays of dimension n or larger. When DEIGV is called then the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ...,WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

The input argument ka is the number of rows in the dimension statement for A in the calling program. ZR and ZI are double precision arrays of dimension $ka \times n$. For $j = 1, \ldots, n$ the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1,j), ...,ZR(n, j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1, \ldots, z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the routine balances A before it computes the eigenvalues and eigenvectors. Otherwise, if IBAL $= 0$ then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks.

(1) A is destroyed during computation.

(2) DEIGV is a double precision version of the eigenvalue/eigenvector routine EIGV1.

Programming. DEIGV is a driver routine for the subroutines DBAL, DORTH, DORTRN, DHQR2, and DBABK. These subroutines are double precision versions of the EISPACK routines BALANC, ORTHES, ORTRAN, HQR2, and BALBAK, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES OF SYMMETRIC REAL MATRICES

The subroutines SEIG and SEIG1 are available for computing the eigenvalues of symmetric real matrices. These routines frequently yield high precision results. SEIG is faster than SEIG1, but at times SEIG1 will produce better results when the symmetric matrix is tridiagonal. For arbitrary symmetric matrices it is not clear if there is any difference in the reliability of the routines.

CALL SEIG$(A, ka, n, W, T, \text{IERR})$
CALL SEIG1$(A, ka, n, W, T, \text{IERR})$

A is a symmetric matrix of order $n \geq 1$ and W an array of dimension n or larger. When SEIG or SEIG1 is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ are computed and stored in $W(1), \ldots, W(n)$. The eigenvalues are ordered so that $\lambda_1 \leq \cdots \leq \lambda_n$.

A may be packed or in standard form.[1] The input argument ka is a nonnegative integer. If $ka = 0$ then A is assumed to be packed. Otherwise, if $ka \neq 0$ then A is assumed to be in the standard format. In this case ka has the value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geq n$. However, it is not required that $A(i, j)$ be defined for $i < j$. Only the lower triangular elements of A are used.

T is an array used for temporary storage. If SEIG is called then T must be of dimension $2n$. However, if SEIG1 is called then T need only be of dimension n.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations of the QL algorithm are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1, \ldots, \lambda_{j-1}$ will have been computed and stored in W. The eigenvalues are ordered so that $\lambda_1 \leq \cdots \leq \lambda_{j-1}$. However, they need not be the smallest eigenvalues of A.

Remark. A is destroyed during computation.

Programming. SEIG and SEIG1 are driver routines for the EISPACK subroutines TRED1, TRED3, TQLRAT, and IMTQL1. These subroutines were developed at Argonne National Laboratory. The function SPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

[1] For details on the packed format see the section on packing and unpacking symmetric matrices.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF SYMMETRIC REAL MATRICES

The subroutines SEIGV and SEIGV1 are available for computing the eigenvalues and eigenvectors of symmetric real matrices. These routines frequently yield high precision results for the eigenvalues. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2–3 units of the 14^{th} significant digit, but the components of the corresponding eigenvector are only accurate to 9–10 significant digits. SEIGV is faster than SEIGV1, but at times SEIGV1 will produce better results when the symmetric matrix is tridiagonal. For arbitrary symmetric matrices it is not clear if there is any difference in the reliability of the routines.

CALL SEIGV$(A, ka, n, W, Z, T, \text{IERR})$
CALL SEIGV1$(A, ka, n, W, Z, T, \text{IERR})$

A is a symmetric matrix of order $n \geq 1$ and W an array of dimension n or larger. When SEIGV or SEIGV1 is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding orthonormal eigenvectors $z_1, \ldots, z_n$ are computed. The eigenvalues are stored in $W(1), \ldots, W(n)$ and are ordered so that $\lambda_1 \leq \cdots \leq \lambda_n$.

A must be in the standard format, having the dimension $ka \times n$. It is assumed that $ka \geq n$. However, it is not required that $A(i,j)$ be defined for $i < j$. Only the lower triangular elements of A are used.

Z is an array of dimension $ka \times n$ or larger. For $j = 1, \ldots, n$ the components of the eigenvector z_j are stored in the j^{th} column of Z (in locations $Z(1,j), \ldots, Z(n,j)$). To conserve memory one may let A and Z denote the same array.

T is an array of dimension n used for temporary storage.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations of the QL algorithm are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1, \ldots, \lambda_{j-1}$ and eigenvectors $z_1, \ldots, z_{j-1}$ will have been computed and stored in the W and Z arrays. However, the eigenvalues will be unordered.

Remark. A is not modified except when A and Z denote the same array.

Programming. SEIGV and SEIGV1 are driver routines for the EISPACK subroutines TRED2, TQL2, and IMTQL2. These subroutines were developed at Argonne National Laboratory. The function SPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines — EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES OF SYMMETRIC REAL MATRICES

The subroutine DSEIG is available for the double precision computation of the eigenvalues of symmetric real matrices. The subroutine frequently yields results accurate to 27–28 significant digits.

CALL DSEIG$(A, ka, n, W, T,$ IERR$)$

A is a double precision symmetric matrix of order $n \geq 1$ and W a double precision array of dimension n or larger. When DSEIG is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ are computed and stored in $W(1), \ldots, W(n)$. The eigenvalues are ordered so that $\lambda_1 \leq \cdots \leq \lambda_n$.

A may be packed or in standard form.[1] The input argument ka is a nonnegative integer. If $ka = 0$ then A is assumed to be packed. Otherwise, if $ka \neq 0$ then A is assumed to be in the standard format. In this case ka has the value:

$ka =$ the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geq n$. However, it is not required that $A(i, j)$ be defined for $i < j$. Only the lower triangular elements of A are used.

T is a double precision array of dimension $2n$ or larger that is a work space for the routine.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 50 iterations of the QL algorithm are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1, \ldots, \lambda_{j-1}$ will have been computed and stored in W. The eigenvalues are ordered so that $\lambda_1 \leq \cdots \leq \lambda_{j-1}$. However, they need not be the smallest eigenvalues of A.

Remarks.

(1) A is destroyed during computation.

(2) DSEIG is a double precision version of the eigenvalue routine SEIG.

Programming. DSEIG is a driver routine for the subroutines DTRED1, DTRED3, and DTQC. These subroutines are double precision versions of the EISPACK routines TRED1, TRED3, and TQLRAT, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The function DPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines — EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

[1] For details on the packed format see the section on packing and unpacking symmetric matrices.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF SYMMETRIC REAL MATRICES

The subroutine DSEIGV is available for the double precision computation of the eigenvalues and eigenvectors of symmetric real matrices. This subroutine frequently yields values for the eigenvalues that are accurate to 26–28 significant digits. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2–3 units of the 28^{th} significant digit, but the components of the eigenvector are only accurate to 22–24 significant digits.

CALL DSEIGV$(A, ka, n, W, Z, T, \text{IERR})$

A is a double precision symmetric matrix of order $n \geq 1$, and W a double precision array of dimension n or larger. When DSEIGV is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The eigenvalues are stored in $W(1), \ldots, W(n)$ and are ordered so that $\lambda_1 \leq \cdots \leq \lambda_n$.

A must be in the standard format, having the dimension $ka \times n$. It is assumed that $ka \geq n$. However, it is not required that $A(i,j)$ be defined for $i < j$. Only the lower triangular elements of A are used.

Z is a double precision array of dimension $ka \times n$ or larger. For $j = 1, \ldots, n$ the components of the eigenvector z_j are stored in the j^{th} column of Z (in locations $Z(1,j), \ldots, Z(n,j)$). To conserve memory one may let A and Z denote the same array.

T is a double precision array of dimension n or larger that is a work space for the routine.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 50 iterations of the QL algorithm are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1, \ldots, \lambda_{j-1}$ and eigenvectors $z_1, \ldots, z_{j-1}$ will have been computed and stored in the W and Z arrays. However, the eigenvalues will be unordered.

Remarks.

(1) A is not modified except when A and Z denote the same array.

(2) DSEIGV is a double precision version of the eigenvalue/eigenvector subroutine SEIGV.

Programming. DSEIGV is a driver routine for the subroutines DTRED2 and DTQL2. These subroutines are double precision versions of the EISPACK subroutines TRED2 and TQL2 developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The function DPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., *Matrix Eigensystem Routines – EISPACK Guide* (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES OF COMPLEX MATRICES

The subroutine CEIG is available for computing the eigenvalues of complex matrices. This routine frequently yields results accurate to 13–14 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 7–8 digit accuracy.

CALL CEIG(IBAL,AR,AI,*ka*,*n*,WR,WI,IERR)

AR and AI are real matrices of order $n \geq 1$, and WR and WI are real arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues are to be computed. When CEIG is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ...,WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered.

IBAL and *ka* are input arguments. It is assumed that *ka* is the number of rows in the dimension statements for AR and AI in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues are computed. Otherwise, if IBAL $= 0$ then the complex matrix is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks.

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy by as much as 5–6 significant digits. Thus it is recommended that balancing be done.

(2) AR and AI are destroyed during computation. CEIG reduces the complex matrix (represented by AR and AI) to upper Hessenberg form with unitary similarity transformations. Then the QR algorithm is used to obtain the eigenvalues.

Usage. If one has a complex matrix A, then AR and AI can be obtained using the matrix subroutines CMREAL and CMIMAG.

Programming. CEIG is a driver routine for the EISPACK subroutines CBAL, CORTH and COMQR. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF COMPLEX MATRICES

The subroutine CEIGV is available for computing the eigenvalues and eigenvectors of complex matrices. This routine frequently yields values for the eigenvalues that are accurate to 13–14 significant digits. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2–3 units of the 14^{th} significant digit, but the components of the corresponding eigenvector are only accurate to 9–10 significant digits. If the eigenvalues of a matrix are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 7–8 digit accuracy, and the situation regarding the eigenvectors is totally unpredictable. The components of such an eigenvector may be correct to 6–7 significant digits, or the eigenvector may not even be an eigenvector! In this case the results should be checked.

CALL CEIGV(IBAL,AR,AI,ka, n, WR,WI,ZR,ZI,IERR,TEMP)

AR and AI are real matrices of order $n \geq 1$ and WR and WI are real arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues and eigenvectors are to be computed. When CEIGV is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ... WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered.

It is assumed that the input argument ka is the number of rows in the dimension statements for AR and AI in the calling program. ZR and ZI are real arrays of dimension $ka \times n$. For $j = 1, \ldots, n$ the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1,j), ...,ZR(n, j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1, \ldots, z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues and eigenvectors are computed. Otherwise, if IBAL $= 0$ then the complex matrix is not balanced.

TEMP is a real array used for temporary storage by the routine. If no balancing is to be done (i.e., if IBAL $= 0$) then TEMP must be of dimension $2n$ or larger. Otherwise, if balancing is to be performed then TEMP must be of dimension $3n$ or larger.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks.

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy of the eigenvalues by as much as 5–6 significant digits. When this occurs balancing will normally be needed to obtain the eigenvectors. In general, it is recommended that balancing be done.

(2) AR and AI are destroyed during computation. CEIGV reduces the complex matrix (represented by AR and AI) to upper Hessenberg form with unitary similarity transformations. Then the QR algorithm is employed to obtain the eigenvalues, and back-substitution is performed to generate the eigenvectors.

Usage. If one has a complex matrix A, then AR and AI can be obtained using the matrix subroutines CMREAL and CMIMAG.

Programming. CEIGV is a driver routine for the EISPACK subroutines CBAL, CORTH, COMQR2, and CBABK2. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and IPMPAR are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES OF COMPLEX MATRICES

The subroutine DCEIG is available for the double precision computation of the eigenvalues of complex matrices. This routine frequently yields results accurate to 26–28 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 13–14 digit accuracy.

CALL DCEIG(IBAL,AR,AI, ka, n, WR,WI,IERR)

AR and AI are double precision matrices of order $n \geq 1$, and WR and WI are double precision arrays of dimension n or larger. AR and AI are the real and imaginary parts of the matrix whose eigenvalues are to be computed. When DCEIG is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ...,WR(n) and the imaginary parts are stored in WI(1), ..., WI(n). The eigenvalues are unordered.

IBAL and ka are input arguments. It is assumed that ka is the number of rows in the dimension statements for AR and AI in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues are computed. Otherwise, if IBAL $= 0$ then the complex matrix is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks.

(1) AR and AI are destroyed during computation.
(2) DCEIG is a double precision version of the eigenvalue routine CEIG.

Programming. DCEIG is a driver routine for the subroutines DCBAL, DCORTH, and DCOMQR. These subroutines are double precision versions of the EISPACK subroutines CBAL, CORTH, and COMQR, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DCPABS, DPMPAR, IPMPAR and subroutine DCSQRT are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines — EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF COMPLEX MATRICES

The subroutine DCEIGV is available for the double precision computation of the eigenvalues and eigenvectors of complex matrices. The routine frequently yields values for the eigenvalues that are accurate to 26–28 significant digits. However, be aware that the errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. If the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 13–14 digit accuracy.

CALL DCEIGV(IBAL,AR,AI,ka,n,WR,WI,ZR,ZI,IERR,TEMP)

AR and AI are double precision matrices of order $n \geq 1$ and WR and WI are double precision arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues and eigenvectors are to be computed. When DCEIGV is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The real parts of the eigenvalues are stored in WR(1), ..., WR(n) and the imaginary parts are stored in WI(1), ...,WI(n). The eigenvalues are unordered.

It is assumed that the input argument ka is the number of rows in the dimension statements for AR and AI in the calling program. ZR and ZI are double precision arrays of dimension $ka \times n$. For $j = 1, \ldots, n$ the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1,j), ...,ZR(n,j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1, \ldots, z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues and eigenvectors are computed. Otherwise, if IBAL $= 0$ then the complex matrix is not balanced.

TEMP is a double precision array used for temporary storage by the routine. If no balancing is to be done (i.e., if IBAL $= 0$) then TEMP must be of dimension $2n$ or larger. Otherwise, if balancing is to be performed then TEMP must be of dimension $3n$ or larger.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks

(1) AR and AI are destroyed during computation.

(2) DCEIGV is a double precision version of the eigenvalue/eigenvector routine CEIGV.

gramming. DCEIGV is a driver for the subroutines DCBAL, DCORTH, DCMQR2, DCBABK. These subroutines are double precision versions of the EISPACK routines CBAL, CORTH, COMQR2, and CBABK2, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DCPABS, DPMPAR, IPMPAR and subroutine DCSQRT are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

ℓ_1 SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND INEQUALITY CONSTRAINTS

Let A be a $k \times n$ matrix, C an $\ell \times n$ matrix, and E an $m \times n$ matrix. Also let b, d, and f be column vectors of dimensions k, ℓ, and m respectively. The following subroutine is available for obtaining a column vector x of dimension n which minimizes $\|Ax - b\|_1 = \sum_{i=1}^{k} |A_i x - b_i|$ subject to the constraints

$$Cx = d$$
$$Ex \leq f.$$

Here A_i denotes the i^{th} row of A, and $Ex \leq f$ means that every component of Ex is less than or equal to the corresponding component of f.

CALL CL1$(k, \ell, m, n, Q, kq,$ KODE,TOL,ITER, X, RES,RNORM,WK,IWK)

It is assumed that $k \geq 1, \ell \geq 0, m \geq 0$, and $n \geq 1$. Q is a 2-dimensional array with kq rows and at least $n + 2$ columns where $kq \geq k + \ell + m + 2$. The matrices A, C, E and vectors b, d, f are stored in the first $k + \ell + m$ rows and $n + 1$ columns of Q as follows:

$$Q = \begin{pmatrix} A & b \\ C & d \\ E & f \end{pmatrix}$$

Q is modified by the routine.

KODE is a variable used for input/output purposes, X an array of dimension $n + 2$ or larger, and RES an array of dimension $k + \ell + m$ or larger. On input KODE is normally set by the user to 0. This indicates that $\|Ax - b\|_1$ is to be minimized subject only to the constraints $Cx = d$ and $Ex \leq f$. However, if it is also desired that one or more variables x_j satisfy $x_j \leq 0$ or $x_j \geq 0$, or that one or more residuals $b_i - A_i x$ satisfy $b_i - A_i x \leq 0$ or $b_i - A_i x \geq 0$, then the user may set KODE to a nonzero value. If KODE $\neq 0$ on input, then the user must also set $X(j)$ and RES(i) to the values

$$X(j) = \begin{cases} -1.0 & x_j \leq 0 \\ 0.0 & x_j \text{ is unrestricted} \\ 1.0 & x_j \geq 0 \end{cases}$$

$$\text{RES}(i) = \begin{cases} -1.0 & b_i - A_i x \leq 0 \\ 0.0 & b_i - A_i x \text{ is unrestricted} \\ 1.0 & b_i - A_i x \geq 0 \end{cases}$$

for $j = 1, \ldots, n$ and $i = 1, \ldots, k$ to indicate the additional constraints which are desired.

RNORM is a variable. When CL1 is called, if a vector x is found that minimizes $\|Ax - b\|_1$ subject to the desired constraints, then KODE $= 0$ on output and the solution x

is stored in X. Also RNORM is assigned the value $\|Ax - b\|_1$, $b - Ax$ is stored in the first k locations of RES, $d - Cx$ is stored in the next ℓ locations, and $f - Ex$ is stored in the last m locations.

When CL1 is called, a modified form of the simplex algorithm is used to minimize $\|Ax - b\|_1$. The arguments TOL and ITER control the use of this algorithm. The input argument TOL is a positive tolerance. CL1 will not pivot on any quantity whose magnitude is less than TOL. Normally the setting TOL $= 10^{-2\nu/3}$ suffices where ν is the number of decimal digits of accuracy available.

Frequently the routine requires less than $5(k+\ell+m)$ iterations of the simplex alogrithm to solve the problem. ITER is a variable used for input/output purposes. On input the user must set ITER to the maximum number of iterations that will be permitted. When the routine terminates, ITER has for its value the number of iterations that were performed.

On output KODE reports the status of the results. The routine assigns KODE one of the following values:

KODE = 0 The problem was solved.
KODE = 1 The problem has no solution.
KODE = 2 Sufficient accuracy cannot be maintained to solve the problem using the current value of TOL.
KODE = 3 The maximum number of iterations were performed. More iterations are needed.

When KODE ≥ 1 on output, X contains the last vector $\bar{x}$ which was obtained, RNORM $= \|A\bar{x} - b\|_1$, and RES contains the vectors $b - A\bar{x}$, $d - C\bar{x}$, and $f - E\bar{x}$.

WK is an array of dimension $2(k + \ell + m + n)$ or larger, and IWK is an array of dimension $3(k + \ell + m) + 2n$ or larger. WK and IWK are work spaces for the routine.

Programming. CL1 calls the subroutine KL1. CL1 was written by I. Barrodale and F. D. K. Roberts (University of Victoria, British Columbia, Canada).

References.

(1) Barrodale, I. and Roberts, F. D. K., "An Improved Algorithm for Discrete ℓ_1 Linear Approximation," ***SIAM J. Numer. Analysis*** **10** (1973), pp. 839–848.

(2) _____, "An Efficient Algorithm for Discrete ℓ_1 Linear Approximation with Linear Constraints," ***SIAM J. Numer. Analysis*** **15** (1978), pp. 603–611.

(3) _____, "Algorithm 552, Solution of the Constrained ℓ_1 Linear Approximation Problem," ***ACM Trans. Math Software*** **6** (1980), pp. 231–235.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

Given an $m \times n$ matrix A and an $m \times \ell$ matrix B. The column vectors $b_1, \ldots, b_\ell$ of B specify ℓ distinct linear least squares problems

$$Ax_j = b_j \quad (j = 1, \ldots, \ell).$$

This set of problems can be written in the form $AX = B$ where X is the $n \times \ell$ matrix having the column vectors $x_1, \ldots, x_\ell$. There always exists a unique minimum length least squares solution x_j for each $Ax_j = b_j$. If B is the $m \times m$ identity matrix, then the matrix whose columns are the minimum length solution vectors x_j is called the ***pseudoinverse*** of A.

For any B, the subroutines LLSQ, LSQR, HFTI, and HFTI2 are available for obtaining the matrix X whose columns are the minimum length solution vectors. LSQR, HFTI, and HFTI2 are more general than LLSQ, begin able to solve arbitrary systems $AX = B$. LLSQ assumes that $m \geq n > 1$ and that the rank of A is n. LSQR employs a more involved procedure than HFTI and HFTI2. LSQR computes the rank of A, whereas HFTI and HFTI2 leave the determination of the rank to the user.

CALL LLSQ$(m, n, A, ka, B, kb, \ell,$ WK,IWK,IERR)

It is assumed that $m \geq n > 1$ and that the rank of A is n. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is required that $ka \geq m$ and $kb \geq m$.

IERR is an integer variable. When LLSQ is called, if no input errors are detected then IERR is set to 0 and the solution matrix X stored in B. Also, if $m \neq n$ then the residual norm $\|Ax_j - b_j\|$ is computed and stored in $B(n+1, j)$ for $j = 1, \ldots, \ell$.[1]

WK and IWK are arrays of dimension n or larger that are work spaces for the routine.

Error Return. IERR $\neq 0$ when $m \geq n > 1$ is not satisfied (IERR $= 1$) or the rank of A is less than n (IERR $= 2$).

Note. A is destroyed during computation.

Programming. LLSQ is a driver for the subroutines ORTHO and ORSOL, written by Nai-Kuan Tsao and Paul J. Nikolai (Aerospace Research Laboratories, Wright-Patterson Air Force Base).

Reference. Tsao, N. K. and Nikolai, P. J., ***Procedures using Orthogonal Transformations for Linear Least Squares Problems***. Report ARL TR 74-0124, Aerospace Research Laboratories, Wright-Patterson Air Force Base, 1974.

[1] Throughout this section $\|c\| = \sqrt{\sum_{i=1}^{m} c_i^2}$ for any vector $c = (c_1, \ldots, c_m)$.

CALL LSQR(MO,A, ka, m, n, B, kb, ℓ,RERR,AERR,k,k_0,RNORM,
WK,ℓw,IWK,ℓiw,IERR)

MO is an argument which specifies if LSQR is being used for the first time. On an initial call, MO = 0 and we have the following setup:

A is an $m \times n$ matrix where $m, n \geq 1$, and ka is the number of rows in the dimension statement for A in the calling program. It is required that $ka \geq m$. A is destroyed by the routine.

The argument ℓ is an integer. If $\ell \geq 1$ then B is an $m \times \ell$ matrix and kb is the number of rows in the dimension statement for B in the calling program. Also, RNORM is an array of dimension ℓ or larger (for output). It is required that $kb \geq \max\{m, n\}$. If $\ell \leq 0$ then there are no equations to be solved, and B, kb, and RNORM are not used.

RERR is an argument which specifies the relative accuracy of the data in A. If it is estimated that the elements in A are accurate to μ significant decimal digits then one may set RERR $= 10^{-\mu}$. It is required that RERR ≥ 0. If RERR $= 0$ then it is assumed that the elements in A are accurate to machine precision.

AERR is an argument which specifies the maximum absolute uncertainty of the data in A. For example, if it is estimated that the elements a_{ij} of A have the relative accuracy RERR except when $|a_{ij}| < 10^{-30}$, then one may set AERR $= 10^{-30}$. It is required that AERR ≥ 0.

The arguments k, k_0, and IERR are variables. When LSQR is called, if no input errors are detected then IERR is set to 0 and the rank of A is bounded from above and below using the tolerances RERR and AERR. The variable k is set to the upper bound and k_0 to the lower bound. If $\ell \geq 1$ then the $n \times \ell$ matrix X whose columns are the minimum length solution vectors is computed (using k as the rank) and stored in B. Also, the residual norm $\| Ax_j - b \|$ is computed and stored in RNORM(j) for $j = 1, \ldots, \ell$.

WK is an array of dimension ℓw where $\ell w \geq 5 \cdot \min\{m, n\}$, and IWK is an array of dimension ℓiw where $\ell iw \geq m + n$. WK and IWK are work spaces for the routine.

Error Return. IERR is assigned one of the following values where an error is detected:

IERR = 1 $m < 1$ or $n < 1$.
IERR = 2 $ka < m$.
IERR = 3 $kb < \max\{m, n\}$
IERR = 4 $\ell iw < m + n$.
IERR = 5 ℓw is too small.
IERR = 6 RERR or AERR is negative.
IERR = 7 MO $\neq 0$ and $\ell = 0$.

No computation is performed when an error is detected.

After an initial call to LSQR, if IERR $= 0$ then the routine may be called with MO $\neq 0$. In this case, it is assumed that only B has been modified and that the new right

equations $AX = B$ are to be solved. The routine employs the Householder factorization of A stored in A, WK, and IWK on the initial call to LSQR.

When MO $\neq 0$, it is assumed that $A, ka, m, n, \ldots$ VK, and ℓiw have not been modified by the user. If $k = 0$ or $k = \min\{m, n\}$ then only the first $\min\{m, n\}$ elements of WK are needed and one may set $\ell w \geq \min\{m, n\}$. Otherwise, it is required that $\ell w \geq 2 \cdot \min\{m, n\}$.

RERR, AERR, and k_0 are not used when MO $\neq 0$. The only assumptions concerning the new B, kb, and ℓ are that $kb \geq \max\{m, n\}$, $\ell \geq 1$, and the dimension of RNORM is now the new value of ℓ. When LSQR is called, if no input errors are detected then IERR is set to 0 and the new solution matrix X obtained. As before, X is stored in B and the residual norms are computed and stored in RNORM.

Remarks. Normally $k_0 = k$ in which case k is the rank of A. However, if $k_0 \neq k$ then it is recommended that HFTI2 be used for obtaining the most appropriate value for the rank and solving $AX = B$. If HFTI2 is used, the sizes of the elements of the solution matrix X should be considered (in addition to the values of the residual norms). Frequently, the lower bound k_0 will be found to be the most appropriate value for the rank.

Programming. LSQR employs the subroutines U11LS, U12LS, U11US, U12US, ISWAP, SSWAP, SAXPY, SSCAL and functions SDOT, SNRM2, ISAMAX, SPMPAR, IPMPAR. LSQR was written by A. H. Morris and U11LS, U12LS, U11US, U12US were written by T. Manteuffel (Los Alamos).

References. Manteuffel, T., ***An Interval Analysis Approach to Rank Determination in Linear Least Squares Problems***, Report SAND 80-0655, Sandia Laboratories, Albuquerque, New Mexico, 1980.

CALL **HFTI**($A, ka, m, n, B, kb, \ell, \tau, k$,RNORM,$H, G$,IP)
CALL **HFTI2**($A, ka, m, n, B, kb, \ell, D, \tau, k$,RNORM,$H, G$,IP,IERR)

A is an $m \times n$ matrix where $m, n \geq 1$, and ka is the number of rows in the dimension statement for A in the calling program. It is required that $ka \geq m$.

The argument ℓ is an integer. If $\ell \geq 1$ then B is an $m \times \ell$ matrix and kb is the number of rows in the dimension statement for B in the calling program. It is required that $kb \geq \max\{m, n\}$. If $\ell < 0$ then there are no equations to be solved, and B and kb are not used.

If $\ell \geq 1$ then RNORM is an array of dimension ℓ or larger. Otherwise, if $\ell < 0$ then RNORM is ignored. When HFTI or HFTI2 is called, if $\ell \geq 1$ then the $n \times \ell$ matrix X whose columns are the minimum length solution vectors is computed and stored in B. Also, the residual norm $\|Ax_j - b_j\|$ is computed and stored in RNORM(j) for $j = 1, \ldots, \ell$.

H, G, and IP are arrays of dimension n or larger that are work spaces for the routines.

The parameters τ, k, and D

τ is a real tolerance that is set by the user, k is a variable, and D an array of

dimension $\min\{m,n\}$ or larger. It is assumed that $\tau \geq 0$. Normally $\tau = 0$ is the setting that is used. D and k are set by the routines.

In order to understand the use of τ, k, and D one must be briefly acquainted with the processing of A. The routines first reduce A to a triangular matrix C where $A = QCP$. Q is an orthogonal matrix and P a permutation matrix. P is defined so that the diagonal elements c_{ii} of C satisfy $|c_{ii}| \geq |c_{i+1,i+1}|$ for each i. The variable k is set to the largest integer such that $|c_{kk}| > \tau$, and if HFTI2 is used then the diagonal elements c_{ii} are stored in D. C is now regarded as the partitioned matrix

$$C = \begin{pmatrix} C_1 & C_2 \\ 0 & C_3 \end{pmatrix}$$

where C_1 is a $k \times k$ matrix. Minimum length least squares solutions x_j are then computed for the problems $Ax_j = b_j$ using only the first k rows of C. This is equivalent to replacing A with

$$\tilde{A} = Q \begin{pmatrix} C_1 & C_2 \\ 0 & 0 \end{pmatrix} P$$

and solving $\tilde{A}x_j = b_j$ for $j = 1, \ldots, \ell$.

Since $|c_{11}| \geq \cdots \geq |c_{kk}| > \tau$ clearly k is the rank of $\tilde{A}$. It is also true that the ratio $|c_{11}|/|c_{kk}|$ is a lower bound on the condition number of C_1 (relative to the spectral norm). Thus, if the ratio is extremely large (say $\geq 10^8$) then a severe loss of accuracy can be expected. A large ratio may be due all or in part to rank deficiency (or near rank deficiency) of the matrix A. Fortunately, rank deficiency is frequently not too difficult to detect and cure. When A is rank deficient then machine roundoff may assign c_{kk} a small value, say 10^{-14}, when it should be 0. The cure is to examine the diagonal elements c_{ii} which are stored in D, to reset τ so as to eliminate the unwanted c_{ii}'s, and then to rerun the problem. This will reduce the order of C_1, thereby lowering the rank of the replacement matrix $\tilde{A}$. C_1 will now be better conditioned, but the value of the residual norms $\|Ax_j - b_j\|$ may be larger. If the norms do increase, then the solution obtained will be satisfactory only if the size of the increased norms fall within acceptable bounds.

Remarks.

(1) The variable k is set to 0 if all $|c_{ii}| \leq \tau$. If $k = 0$ then the zero matrix is the solution for $AX = B$.

(2) If $\ell < 0$ then the decomposition $A = QCP$ is performed, the diagonal elements of C are stored in D, and k is computed.

(3) The contents of A are destroyed by the routines.

(4) HFTI and HFTI2 yield the same results.

Error Return. IERR is a variable that is set by the routine. If no input errors are detected then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1 if $m = ka$

IERR = 2 if $\ell = 1$ and $kb = \max\{m,n\}$

When an error is detected, the routine immediately terminates.

Programming. HFTI and HFTI2 call the subroutine H12. These routines were written by Charles L. Lawson and Richard J. Hanson (Jet Propulsion Laboratory), and modified by A. H. Morris.

Reference. Lawson, C. L., and Hanson, R. J., ***Solving Least Squares Problems***, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

LEAST SQUARES SOLUTION OF OVERDETERMINED SYSTEMS OF LINEAR EQUATIONS WITH ITERATIVE IMPROVEMENT

Given an $m \times n$ matrix A and an $m \times \ell$ matrix B. The column vectors $b_1, \ldots, b_\ell$ of B specify ℓ distinct linear least squares problems

$$Ax_j = b_j \quad (j = 1, \ldots, \ell).$$

This set of problems can be written in the form $AX = B$ where X is the $n \times \ell$ matrix having the column vectors $x_1, \ldots, x_\ell$. Assume that $m \geq n > 1$ and that the rank of A is n. Then there exists a unique least squares solution x_j for each $Ax_j = b_j$. The subroutine LLSQMP is available for obtaining the solution matrix X. Iterative improvement is performed to compute X to machine accuracy.

CALL LLSQMP$(m, n, A, ka, B, kb, \ell,$ WK,IWK,IERR)

It is assumed that $m \geq n > 1$ and that the rank of A is n. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is required that $ka \geq m$ and $kb \geq m$.

When LLSQMP is called, the solution X is computed and stored in B. Also, if $m \neq n$ then the residual norm $\|Ax_j - b_j\|$ is computed and stored in $B(n+1, j)$ for $j = 1, \ldots, \ell$.[1] A is not modified by the routine.

WK is an array of dimension $mn + 2m + n$ or larger, and IWK an array of dimension n or larger. WK and IWK are work spaces for the routine.

IERR is a variable that is set by the routine. When LLSQMP terminates, IERR has one of the following values:

IERR = 0 The solution X was computed to machine accuracy.
IERR = 1 X was obtained, but not to machine accuracy.
IERR = 2 The restriction $m \geq n > 1$ is not satisfied
IERR = 3 The rank of A is less than n.

Programming. LLSQMP is a driver for the subroutines ORTHO, ORSOL, and ORIMP. These subroutines were written by Nai-Kuan Tsao and Paul J. Nikolai (Aerospace Research Laboratories, Wright-Patterson Air Force Base). ORIMP was modified by A. H. Morris. The function SPMPAR and subroutine MCOPY are also used.

Reference. Tsao, N. K. and Nikolai, P. J., ***Procedures using Orthogonal Transformations for Linear Least Squares Problems***, Report ARL TR 74-0124, Aerospace Research Laboratories, Wright-Patterson Air Force Base, 1974

[1] Here $\|v\| = \sqrt{\Sigma_i v_i^2}$ for any vector $v = (v_1, \ldots, v_m)$.

DOUBLE PRECISION LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

Given an $m \times n$ matrix A and an $m \times \ell$ matrix B. The column vectors $b_1, \ldots, b_\ell$ of B specify ℓ distinct linear least squares problems

$$Ax_j = b_j \quad (j = 1, \ldots, \ell).$$

This set of problems can be written in the form $AX = B$ where X is the $n \times \ell$ matrix having the column vectors $x_1, \ldots, x_\ell$. There always exists a unique minimum length least squares solution x_j for each $Ax_j = b_j$. If B is the $m \times m$ identity matrix, then the matrix whose columns are the minimum length solution vectors x_j is called the ***pseudoinverse*** of A.

For any B, the subroutines DLLSQ, DLSQR, DHFTI, and DHFTI2 are available for obtaining the matrix X whose columns are the minimum length solution vectors. DLSQR, DHFTI, and DHFTI2 are more general than DLLSQ, begin able to solve arbitrary systems $AX = B$. DLLSQ assumes that $m \geq n > 1$ and that the rank of A is n. DLSQR employs a more involved procedure than DHFTI and DHFTI2. DLSQR computes the rank of A, whereas DHFTI and DHFTI2 leave the determination of the rank to the user. The subroutines perform all calculations in double precision.

CALL DLLSQ(m, n, A, ka, B, kb, ℓ, WK,IWK,IERR)

A and B are double precision arrays. It is assumed that $m \geq n > 1$ and that the rank of A is n. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is required that $ka \geq m$ and $kb \geq m$.

IERR is an integer variable. When DLLSQ is called, if no input errors are detected then IERR is set to 0 and the solution matrix X stored in B. Also, if $m \neq n$ then the residual norm $\|Ax_j - b_j\|$ is computed and stored in $B(n+1, j)$ for $j = 1, \ldots, \ell$.[1]

WK and IWK are arrays of dimension n or larger that are work spaces for the routine. WK is a double precision array.

Error Return IERR $\neq 0$ when $m \geq n \geq 1$ is not satisfied (IERR = 1) or the rank of A is less than n (IERR = 2).

Note A is destroyed during computation.

Programming DLLSQ is a driver for the subroutines DORTHO and DORSOL. These subroutines are double precision versions of ORTHO and ORSOL, written by Nai Kuan Tsao and Paul J. Nikolai (Aerospace Research Laboratories, Wright Patterson Air Force Base).

[1] Throughout this section $\|c\| = \sqrt{\sum c_i^2}$ for any vector $c = (c_1, \ldots, c_m)$.

Reference. Tsao, N. K. and Nikolai, P. J., ***Procedures using Orthogonal Transformations for Linear Least Squares Problems***. Report ARL TR 74-0124, Aerospace Research Laboratories, Wright-Patterson Air Force Base, 1974.

CALL DLSQR(MO,A, ka, m, n, B, kb, ℓ,RERR,AERR,k,k_0,RNORM, WK,ℓw,IWK,ℓiw,IERR)

MO is an argument which specifies if DLSQR is being used for the first time. On an initial call, MO = 0 and we have the following setup:

A is a double precision $m \times n$ matrix where $m, n \geq 1$, and ka is the number of rows in the dimension statement for A in the calling program. It is required that $ka \geq m$. A is destroyed by the routine.

The argument ℓ is an integer. If $\ell \geq 1$ then B is a double precision $m \times \ell$ matrix and kb is the number of rows in the dimension statement for B in the calling program. Also, RNORM is a double precision array of dimension ℓ or larger (for output). It is required that $kb \geq \max\{m, n\}$. If $\ell \leq 0$ then there are no equations to be solved, and B, kb, and RNORM are not used.

RERR is a double precision value which specifies the relative accuracy of the data in A. If it is estimated that the elements in A are accurate to μ significant decimal digits then one may set RERR $= 10^{-\mu}$. It is required that RERR ≥ 0. If RERR $= 0$ then it is assumed that the elements in A are accurate to machine precision.

AERR is a double precision value which specifies the maximum absolute uncertainty of the data in A. For example, if it is estimated that the elements a_{ij} of A have the relative accuracy RERR except when $|a_{ij}| \leq 10^{-30}$, then one may set AERR $= 10^{-30}$. It is required that AERR > 0.

The arguments k, k_0, and IERR are variables. When DLSQR is called, if no input errors are detected then IERR is set to 0 and the rank of A is bounded from above and below using the tolerances RERR and AERR. The variable k is set to the upper bound and k_0 to the lower bound. If $\ell \geq 1$ then the $n \times \ell$ matrix X whose columns are the minimum length solution vectors is computed (using k as the rank) and stored in B. Also, the residual norm $\|Ax_j - b_j\|$ is computed and stored in RNORM(j) for $j = 1, \ldots, \ell$.

WK is a double precision array of dimension ℓw where $\ell w \geq 5 \cdot \min\{m, n\}$, and IWK is an integer array of dimension ℓiw where $\ell iw \geq m + n$. WK and IWK are work spaces for the routine.

Error Return. IERR is assigned one of the following values where an error is detected:

IERR = 1 $m < 1$ or $n < 1$
IERR = 2 $ka < m$
IERR = 3 $kb < \max\{m, n\}$
IERR = 4 $\ell iw < m + n$
IERR = 5 ℓw is too small

IERR = 6 RERR or AERR is negative.
IERR = 7 MO $\neq 0$ and $\ell \leq 0$.

No computation is performed when an error is detected.

After an initial call to DLSQR, if IERR = 0 then the routine may be called with MO $\neq 0$. In this case, it is assumed that only B has been modified and that the new set of equations $AX = B$ are to be solved. The routine employs the Householder factorization of A stored in A, WK, and IWK on the initial call to DLSQR.

When MO $\neq 0$, it is assumed that A, ka, m, n, k,IWK, and ℓiw have not been modified by the user. If $k = 0$ or $k = \min\{m, n\}$ then only the first $\min\{m, n\}$ elements of WK are needed and one may set $\ell w \geq \min\{m, n\}$. Otherwise, it is required that $\ell w \geq 2 \cdot \min\{m, n\}$.

RERR, AERR, and k_0 are not used when MO $\neq 0$. The only assumptions concerning the new B, kb, and ℓ are that $kb \geq \max\{m, n\}$, $\ell \geq 1$, and the dimension of RNORM is now the new value of ℓ. When DLSQR is called, if no input errors are detected then IERR is set to 0 and the new solution matrix X obtained. As before, X is stored in B and the residual norms are computed and stored in RNORM.

Remarks.Normally $k_0 = k$, in which case k is the rank of A. However, if $k_0 \neq k$ then it is recommended that DHFTI2 be used for obtaining the most appropriate value for the rank and solving $AX = B$. If DHFTI2 is used, the sizes of the elements of the solution matrix X should be considered (in addition to the values of the residual norms). Frequently, the lower bound k_0 will be found to be the most appropriate value for the rank.

Programming. DLSQR employs the subroutines DU11LS, DU12LS, DU11US, DU12US, ISWAP, DSWAP, DAXPY, DSCAL and functions DDOT, DNRM2, IDAMAX, DPMPAR, IPMPAR. DLSQR was written by A. H. Morris and DU11LS, DU12LS, DU11US, DU12US were written by T. Manteuffel (Los Alamos).

References. Manteuffel, T., ***An Interval Analysis Approach to Rank Determination in Linear Least Squares Problems***, Report SAND 80-0655, Sandia Laboratories, Albuquerque, New Mexico, 1980.

CALL DHFTI($A, ka, m, n, B, kb, \ell, \tau, k$,RNORM,$H$,$G$,IP)
CALL DHFTI2($A, ka, m, n, B, kb, \ell, D, \tau, k$,RNORM,$H$,$G$,IP,IERR)

A is a double precision $m \times n$ matrix where $m, n \geq 1$, and ka is the number of rows in the dimension statement for A in the calling program. It is required that $ka \geq m$.

The argument ℓ is an integer. If $\ell \geq 1$ then B is a double precision $m \times \ell$ matrix and kb is the number of rows in the dimension statement for B in the calling program. It is required that $kb \geq \max\{m, n\}$. If $\ell \leq 0$ then there are no equations to be solved, and B and kb are not used.

If $\ell \geq 1$ then RNORM is a double precision array of dimension ℓ or larger. Otherwise, if $\ell \leq 0$ then RNORM is ignored. When DHFTI or DHFTI2 is called, if $\ell \geq 1$ then the $n \times \ell$ matrix X whose columns are the minimum length solution vectors is computed and

stored in B. Also, the residual norm $||Ax_j - b_j||$ is computed and stored in RNORM(j) for $j = 1, \ldots, \ell$.

H, G, and IP are arrays of dimension n or larger that are work spaces for the routines. H and G are double precision arrays.

The parameters τ, k, and D.

τ is a double precision tolerance that is set by the user, k an integer variable, and D a double precision array of dimension $\min\{m, n\}$ or larger. It is assumed that $\tau \geq 0$. Normally $\tau = 0$ is the setting that is used. D and k are set by the routines.

In order to understand the use of τ, k, and D one must be briefly acquainted with the processing of A. The routines first reduce A to a triangular matrix C where $A = QCP$. Q is an orthogonal matrix and P a permutation matrix. P is defined so that the diagonal elements c_{ii} of C satisfy $|c_{ii}| \geq |c_{i+1,i+1}|$ for each i. The variable k is set to the largest integer such that $|c_{kk}| > \tau$, and if DHFTI2 is used then the diagonal elements c_{ii} are stored in D. C is now regarded as the partitioned matrix

$$C = \begin{pmatrix} C_1 & C_2 \\ 0 & C_3 \end{pmatrix}$$

where C_1 is a $k \times k$ matrix. Minimum length least squares solutions x_j are then computed for the problems $Ax_j = b_j$ using only the first k rows of C. This is equivalent to replacing A with

$$\tilde{A} = Q \begin{pmatrix} C_1 & C_2 \\ 0 & 0 \end{pmatrix} P$$

and solving $\tilde{A}x_j = b_j$ for $j = 1, \ldots, \ell$.

Since $|c_{11}| \geq \cdots \geq |c_{kk}| > \tau$ clearly k is the rank of $\tilde{A}$. It is also true that the ratio $|c_{11}| / |c_{kk}|$ is a lower bound on the condition number of C_1 (relative to the spectral norm). Thus, if the ratio is extremely large (say $\geq 10^8$) then a severe loss of accuracy can be expected. A large ratio may be due all or in part to rank deficiency (or near rank deficiency) of the matrix A. Fortunately, rank deficiency is frequently not too difficult to detect and cure. When A is rank deficient then machine roundoff may assign c_{kk} a small value, say 10^{-28}, when it should be 0. The cure is to examine the diagonal elements c_{ii} which are stored in D, to reset τ so as to eliminate the unwanted c_{ii}'s, and then to rerun the problem. This will reduce the order of C_1, thereby lowering the rank of the replacement matrix $\tilde{A}$. C_1 will now be better conditioned, but the value of the residual norms $||Ax_j - b_j||$ may be larger. If the norms do increase, then the solution obtained will be satisfactory only if the size of the increased norms fall within acceptable bounds.

Remarks.

(1) The variable k is set to 0 if all $|c_{ii}| \leq \tau$. If $k = 0$ then the zero matrix is the solution for $AX = B$.

(2) If $\ell \leq 0$ then the decomposition $A = QCP$ is performed, the diagonal elements of C are stored in D, and k is computed.

(3) The contents of A are destroyed by the routines.

(4) DHFTI and DHFTI2 yield the same results. These routines are double precision versions of the subroutines HFTI and HFTI2.

Error Return. IERR is a variable that is set by the routine. If no input errors are detected then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1 if $m > ka$
IERR = 2 if $\ell > 1$ and $kb < \max\{m, n\}$

When an error is detected, the routine immediately terminates.

Programming. DHFTI and DHFTI2 call the subroutine DH12. These routines are modifications by A. H. Morris of the subroutines HFTI and H12, written by Charles L. Lawson and Richard J. Hanson (Jet Propulsion Laboratory).

Reference. Lawson, C. L., and Hanson, R. J., ***Solving Least Squares Problems***, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND INEQUALITY CONSTRAINTS

Let A be an $m_a \times n$ matrix, E an $m_e \times n$ matrix, G an $m_g \times n$ matrix, b a column vector of dimension m_a, f a column vector of dimension m_e, and h a column vector of dimension m_g. The subroutine LSEI is available for finding a column vector x of dimension n that minimizes $\|Ax - b\|$ subject to the constraints[1]

$$Ex \simeq f$$
$$Gx \geq h.$$

$Ex \simeq f$ states that x is a least squares solution of the equation $Ex = f$, and $Gx \geq h$ means that every component of the vector Gx must be equal to or greater than the corresponding component of h. It is assumed that $m_a > 0$, $m_e \geq 0$, and $m_g \geq 0$.

CALL LSEI(W, kw, m_e, m_a, m_g, n, OPT, x, RNORME, RNORMA, IERR, WK, IWK)

If $m = m_e + m_a + m_g$ then W is the $m \times (n+1)$ matrix:

$$W = \begin{pmatrix} E & f \\ A & b \\ G & h \end{pmatrix}$$

The input argument kw is assumed to have the value:

kw = the number of rows in the dimension statement for W in the calling program

Thus it is required that $kw \geq m$.

RNORME and RNORMA are real variables. When LSEI is called, if the constraints $Ex \simeq f$ and $Gx \geq h$ are consistent then x is computed, RNORME is assigned the value $\|Ex - f\|$,[2] and RNORMA is assigned the value $\|Ax - b\|$.

OPT is an array, called the ***option vector***, which permits the user to take advantage of certain options that are supplied by the routine. If no options are desired then OPT may be declared to have dimension 1 and OPT(1) must be assigned the value 1. The details concerning the available options and how to specify them in OPT are given below.

IWK is an array of dimension $m_g + 2n + 2$ or larger, and WK is an array of dimension $2(m_e + n) + \max\{m_a + m_g, n\} + (m_g + 2)(n + 7)$ or larger. IWK and WK are work spaces. When LSEI is called, using a solution for $Ex \simeq f$, a reduced least squares problem with inequality constraints is obtained and solved. When the routine terminates IWK(1), IWK(2), IWK(3) contain the following information:

[1] Throughout this section $\|e\|$ denotes the norm $\sqrt{\Sigma_i e_i^2}$ for any vector $e = (e_1, e_2, \ldots)$.
[2] If $m_e = 0$ then RNORME = 0.

IWK(1) = the estimated rank of the matrix E
IWK(2) = the estimated rank of the reduced problem
IWK(3) = the amount of storage in the array WK that was actually needed

IERR is a variable that is set by the routine. When LSEI terminates, IERR has one of the following values:

IERR = 0 The solution x was obtained. The equality $Ex = f$ is satisfied when $m_e \neq 0$.
IERR = 1 The solution x was obtained. In this case $\|Ex - f\| > 0$.
IERR = 2 The problem cannot be solved. The constraints are inconsistent.
IERR = 4 (Input error) Either $kw < m$, the covariance matrix is requested and $kw < n$, or the option vector OPT is not defined properly.

If IERR ≥ 2 then x, RNORME, and RNORMA are not defined.

Remarks.

(1) W is modified by the routine.
(2) If $m_e + m_a \leq 0$ or $n \leq 0$ then IERR is set to 0 and the routine terminates.

The option vector OPT. If no options are desired then OPT may be declared to be of dimension 1 and OPT(1) must have the value 1. Otherwise, OPT is a linked list consisting of groups of data link_i, key_i, data_i $(i = 1, \ldots, s)$. Each link_i and key_i is an integer. The amount of storage required by data_i depends on the value of key_i. The general layout of OPT is as follows:

OPT(1) = link_1 (index of the first entry of the next group)
OPT(2) = key_1 (key to the option)
OPT(3) = the first word of the data (data_1) for this option
.
.
.
OPT(link_1) = link_2 (index of the first entry of the next group)
OPT($\text{link}_1 + 1$) = key_2 (key to the option)
OPT($\text{link}_1 + 2$) = the first word of the data (data_2) for this option
.
.
.
OPT(link_s) = 1.0 (There are no more options to be considered.)

The following options are available:

key = 1 It is assumed that $kw \geq n$. Compute the $n \times n$ covariance matrix and store it in the first n rows and columns of W. The data for this option is a single value. It must be nonzero for the covariance matrix to be computed.

key = 2 Scale the nonzero columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$ so that they have

length 1. The data for this option is a single value. It must be nonzero for the scaling to be performed.

key = 3 Scale the columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$. The data for this option consists of n scaling factors, one for each matrix column.

key = 4 Change the internal tolerance τ which is used for determining the rank of E. The data for this option is the new tolerance. τ may be set to any value $\geq \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1 (\epsilon = 2^{-47}$ for the CDC 6700). If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.

key = 5 Change the internal tolerance τ which is used for rank determination in the reduced least squares problem. The data for this option is the new tolerance. τ may be set to any value $\geq \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1$. If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.

Also the key 8 and 9 options for the least squares subroutine WNNLS are permitted. (WNNLS is employed by LSEI.) The order of the options in the array OPT is arbitrary. If an option has an unrecognized key then the option is ignored. It is assumed that the dimension of OPT is no greater than 100000 and that the number of options is ≤ 1000. If either of these assumptions is violated then IERR is set to 4 and the routine terminates. It is also required that $\text{link}_i \neq \text{link}_j$ for $i \neq j$. If this restriction is not satisfied then the linked list OPT is circular and we again have an IERR = 4 error.

Example. Assume that we have an array D containing n scaling factors for the columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$, and that the tolerance TOL is always to be used for rank determination. Then OPT will have to be of dimension $\geq n + 9$ and OPT can be defined as follows:

```
      OPT(1) = N + 3                (Scaling option)
      OPT(2) = 3.0
      DO 10 I = 1, N
10        OPT(I + 2) = D(I)
      OPT(N + 3) = N + 6            (Matrix E tolerance option)
      OPT(N + 4) = 4.0
      OPT(N + 5) = TOL
      OPT(N + 6) = N + 9            (Reduced problem tolerance option)
      OPT(N + 7) = 5.0
      OPT(N + 8) = TOL
      OPT(N + 9) = 1.0              (There are no more options.)
```

Remarks.

(1) LSEI may perform poorly if the norms of the rows of A and E differ by many orders of magnitude, or if the norms of the rows of E are exceedingly small.

(2) The covariance matrix obtained by the key = 1 option may not be meaningful when

there are inequality constraints $Gx \geq h$. This matrix assumes that any inequalities which are selected by the algorithm to be equalities remain equalities when the solution is perturbed. This, of course, may not be the case.

Programming. LSEI employs the subroutine LSI, LPDP, WNNLS, WNLSM, and WNLIT. These routines were written by Karen H. Haskell and Richard J. Hanson (Sandia Laboratories) and modified by A. H. Morris. The subroutines HFTI, H12, SROTM, SROTMG, SCOPY, SSWAP, SSCAL, SAXPY and functions SPMPAR, SDOT, SASUM, SNRM2, ISAMAX are also used.

References.

(1) Hanson, R. J. and Haskell, K. H., "Algorithm 587: Two Algorithms for the Linearly Constrained Least Squares Problem," ***ACM Trans. Math Software*** **8** (1982), pp. 323–333.

(2) Haskell, K. H. and Hanson, R. J., "An Algorithm for Linear Least Squares Problems with Equality and Nonnegativity Constraints," ***Math. Programming*** **21** (1981), pp. 98–118.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND NONNEGATIVITY CONSTRAINTS

Let A be an $m_a \times n$ matrix, E an $m_e \times n$ matrix, b a column vector of dimension m_a, and f a column vector of dimension m_e The subroutine WNNLS is available for finding a column vector $x = (x_1, \ldots, x_n)^t$ that minimizes $\|Ax - b\|$ subject to the constraints[1]

$$Ex \simeq f$$
$$x_i \geq 0 \text{ for } i > \ell.$$

$Ex \simeq f$ states that x is a least squares solution of the equation $Ex = f$. It is assumed that $m_a \geq 0, m_e \geq 0$, and $0 \leq \ell \leq n$. If $m_a = 0$ then WNNLS solves $Ex \simeq f$ subject to the constraints $x_i \geq 0$ $(i > \ell)$.

CALL WNNLS$(W, kw, m_e, m_a, n, \ell, \text{OPT}, x, \text{RNORM,MODE,IWK,WK})$

If $m = m_e + m_a$ then W is the $m \times (n+1)$ matrix:

$$W = \begin{pmatrix} E & f \\ A & b \end{pmatrix}$$

The input argument kw is assumed to have the value:
kw = the number of rows in the dimension statement for W in the calling progam
Thus it is required that $kw \geq m$.

RNORM is a variable. When WNNLS is called, x is computed and RNORM is assigned the value $\sqrt{\|Ax - b\|^2 + \|Ex - f\|^2}$.[2]

OPT is an array, called the ***option vector***, which permits the user to take advantage of certain options that are supplied by the routine. If no options are desired then OPT may be declared to have dimension 1 and OPT(1) must be assigned the value 1. The details concerning the available options and how to specify them in OPT are given below.

IWK is an array of dimension $m + n$ or larger, and WK is an array of dimension $m + 5n$ or larger. IWK and WK are work spaces for the routine.

Error Return. MODE is an integer variable that is set by the routine. If the problem is solved then MODE is assigned the value 0. Otherwise, MODE is assigned one of the following values:

MODE = 1 The maximum number of iterations ($3(n - \ell)$ iterations) was exceeded. An approximate solution and its residual norm are stored in x and RNORM.

MODE = 2 (Input error) Either $kw > m$ or $0 \leq \ell \leq n$ is violated, or the option vector OPT is not properly defined.

[1] Throughout this section $\|c\|$ denotes the norm $\sqrt{\Sigma c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.
[2] If $m_a = 0$ then RNORM $= \|Ex - f\|$, and if $m_e = 0$ then RNORM $= \|Ax - b\|$.

When an input error is detected, the routine immediately terminates. In this case x and RNORM are not defined.

Remarks.

(1) W is modified by the routine.
(2) If $m \leq 0$ or $n \leq 0$ then MODE is set to 0 and the routine terminates.

The option vector OPT. If no options are desired then OPT may be declared to be of dimension 1 and OPT(1) must have the value 1. Otherwise, OPT is a linked list consisting of groups of data link_i, key_i, data_i $(i = 1, \ldots, s)$. Each link_i and key_i is an integer. The amount of storage required by data_i depends on the value of key_i. The general layout of OPT is as follows:

$\text{OPT}(1) = \text{link}_1$ (index of the first entry of the next group)
$\text{OPT}(2) = \text{key}_1$ (key to the option)
$\text{OPT}(3) =$ the first word of the data (data_1) for this option
.
.
.
$\text{OPT}(\text{link}_1) = \text{link}_2$ (index of the first entry of the next group
$\text{OPT}(\text{link}_1 + 1) = \text{key}_2$ (key to the option)
$\text{OPT}(\text{link}_1 + 2) =$ the first word of the data (data_2) for this option
.
.
.
$\text{OPT}(\text{link}_s) = 1.0$ (There are no more options to be considered.)

The following options are permitted:

key = 6 Scale the nonzero columns of the matrix $\begin{pmatrix} E \\ A \end{pmatrix}$ so that they have length 1. The data for this option is a single value. It must be nonzero for the scaling to be performed.

key = 7 Scale the columns of the matrix $\begin{pmatrix} E \\ A \end{pmatrix}$. The data for this option consists of n scaling factors, one for each matrix column.

key = 8 Change the internal tolerance τ which is used for rank determination. The data for this option is the new tolerance. τ may be set to any value $\geq \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1$. ($\epsilon = 2^{-47}$ for the CDC 6700.) If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.

key = 9 Change the parameter BLOWUP. The reciprocal of this parameter is used in determining when solution components are too large. The data for this option is the new value for BLOWUP. It is assumed that BLOWUP < 1. BLOWUP may be set to any value $\geq \epsilon$ where ϵ is the smallest number for which $1 + \epsilon > 1$. If the new value is less than ϵ then it is ignored and BLOWUP is set to ϵ. The default value used for BLOWUP is $\sqrt{\epsilon}$.

The order of the options in the array OPT is arbitrary. If an option has an unrecognized key then the option is ignored. It is assumed that the dimension of OPT is no greater than 100000 and that the number of options $\leq$ 1000. If either of these assumptions is violated then MODE is set to 2 and the routine terminates. It is also required that $\text{link}_i \neq \text{link}_j$ for $i \neq j$. If this restriction is not satisfied then the linked list OPT is circular and we again have a MODE = 2 error.

Example. Assume that we have an array D containing n scaling factors for the columns of the matrix $\binom{E}{A}$, and that TOL is the tolerance to be used for rank determination. Then OPT will have to be of dimension $\geq n + 6$ and OPT can be defined as follows:

```
      OPT(1) = N + 3          (Scaling option)
      OPT(2) = 7.0
      DO 10 I = 1, N
10    OPT(I + 2) = D(I)
      OPT(N + 3) = N + 6      (Tolerance option)
      OPT(N + 4) = 8.0
      OPT(N + 5) = TOL
      OPT(N + 6) = 1.0        ( here are no more options.)
```

Remark. WNNLS may perform poorly if the norms of the rows of A and E differ by many orders of magnitude, or if the norms of the rows of E are exceedingly small.

Programming. WNNLS employs the subroutines WNLSM and WNLIT. These routines were written by Karen H. Haskell and Richard J. Hanson (Sandia Laboratories), and modified by A. H. Morris and Virgis Dadurkevicius (Astronomical Observatory, Vilnius University, Lithuania). The subroutines H12, SROTM, SROTMG, SCOPY, SSWAP, SSCAL, SAXPY and functions SPMPAR, SASUM, SNRM2, ISAMAX are also used.

References.

(1) Dadurkevicius,V.,"Remark on Algorithm 587,"***ACM Trans. Math Software*** 15 (1989), pp. 257-261

(2) Hanson, R. J. and Haskell, K. H., "Algorithm 587: Two Algorithms for the Linearly Constrained Least Squares Problem," ***ACM Trans. Math Software*** 8 (1982), pp. 323-333.

(3) Haskell, K. H. and Hanson, R. J., "An Algorithm for Linear Least Squares Problems with Equality and Nonnegativity Constraints," ***Math. Programming*** 2 (1981), pp. 98-118.

LEAST SQUARES ITERATIVE IMPROVEMENT SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY CONSTRAINTS

Let A be an $m_a \times n$ matrix, E an $m_e \times n$ matrix, B an $m_a \times \ell$ matrix, and F an $m_e \times \ell$ matrix. It is assumed that $0 \leq m_e \leq n$ and that the rank of E is m_e. Let $b_1, \ldots, b_\ell$ denote the column vectors of B and $f_1, \ldots, f_\ell$ the column vectors of F. The subroutine L2SLV is available for finding the unique minimum length column vector x_j of dimension n that minimizes $\|Ax_j - b_j\|$ subject to the equality constraints $Ex_j = f_j$ (if there are any) for $j = 1, \ldots, \ell$.[1] Iterative improvement is performed to compute the vectors $x_1, \ldots, x_\ell$ to machine accuracy. It is assumed that $m_a \geq 0$. If $m_a = 0$ then L2SLV finds the unique minimum length solution x_j to $Ex_j = f_j$ for $j = 1, \ldots, \ell$.

CALL L2SLV($m, n, m_e, \ell, \tilde{A}, ka, \tilde{B}, kb$, WGTS,TOL,N1,IPIVOT, X, kx, R, kr, T, kt, WK,IERR)

If $m = m_e + m_a$ then $\tilde{A}$ is the $m \times n$ matrix $\binom{E}{A}$ and $\tilde{B}$ is the $m \times \ell$ matrix $\binom{F}{B}$. The input arguments ka and kb have the values:

ka = the number of rows in the dimension statement for $\tilde{A}$ in the calling program

kb = the number of rows in the dimension statement for $\tilde{B}$ in the calling program

It is assumed that $m \geq 1, n \geq 1, \ell \geq 1, ka \geq m$, and $kb \geq m$. $\tilde{A}$ and $\tilde{B}$ are not modified by the routine.

WGTS is an array containing m nonnegative weights. The first m_e weights are set to 1.0 by the routine. Let $w_1, \ldots, w_{m_a}$ denote the remaining weights (i.e., let $w_i = \text{WGTS}(m_e + i)$ for $i = 1, \ldots, m_a$). The remaining weights are supplied by the user. In effect, w_i is the weighting that is given to the i^{th} equation in the least squares problem $Ax_j = b_j$. If W denotes the $m_a \times m_a$ diagonal matrix $\text{diag}(w_1, \ldots, w_{m_a})$ then L2SLV finds the unique minimum length vector that minimizes $\|WAx_j - Wb_j\|$ subject to $Ex_j = f_j$ for $j = 1, \ldots, \ell$. For convenience, $\widetilde{W}$ will denote the $m \times m$ diagonal matrix $\text{diag}\,(1, \ldots, 1, w_1, \ldots, w_{m_a})$.

X is an $n \times \ell$ matrix that contains the solution vectors $x_1, \ldots, x_\ell$ when the routine terminates. The input argument kx is the number of rows in the dimension statement for X in the calling program. It is assumed that $kx \geq n$.

R is an $m \times \ell$ matrix. Let $\tilde{b}_j$ denote the j^{th} column vector $\binom{f_j}{b_j}$ of $\tilde{B}$ for $j = 1, \ldots, \ell$. Then L2SLV stores the residual vector $r_j = \widetilde{W}\tilde{b}_j - \widetilde{W}\tilde{A}x_j$ in the j^{th} column of R. The input argument kr is the number of rows in the dimension statement for R in the calling progam. It is assumed that $kr \geq m$.

WK is an array of dimension $6(m + n) + 2\ell$ or larger that is used for a work space. When L2SLV terminates, for $j = 1, \ldots, \ell$

$$WK(j) = \begin{cases} n_j & \text{if iterative improvement of the solution } x_j \text{ converged} \\ -n_j & \text{if iterative improvement of } x_j \text{ failed to converge} \end{cases}$$

[1] Throughout this section $\|c\|$ denotes the norm $\sqrt{\sum_i c_i^2}$ for any vector $c = (c_1, \ldots, c_{m_a})$

where n_j is the number of iterations in the iterative improvement process that were performed in computing x_j. Also

$\mathrm{WK}(\ell+j)$ = the estimated number of correct digits in x_j before iterative improvement was performed

for $j = 1, \ldots, \ell$.

TOL and N1 correspond to the parameters τ and k in the least squares subroutines HFTI and HFTI2. TOL is a nonnegative number that is specified by the user, and N1 is a variable that is set by the routine. When L2SLV is called, modified Gram-Schmidt orthogonalization with column pivoting is used to reduce $\widetilde{WA}$ to the form $(A_1 A_2)$ where A_1 is an $m \times N_1$ matrix having rank N_1. A_1 is of the form QU where $Q^tQ = \mathrm{diag}(d_1, \ldots, d_{N_1})$ and U is an upper unit triangular matrix. The values $d_1, d_2, \ldots$ correspond to the diagonal elements $c_{11}, c_{22}, \ldots$ generated by HFTI and HFTI2 ($d_i = c_{ii}^2$ for $i = 1, 2, \ldots$). The values are ordered so that $d_i \geq d_{i+1}$, and $d_1, d_2, \ldots$ are stored in $\mathrm{WK}(2\ell+1)$, $\mathrm{WK}(2\ell+2)$, ... If $m = m_e$ then $N1$ is assigned the value m_e. Otherwise, if $m > m_e$ then $N1$ is the largest integer k for which $d_k > \tau$. Here $\tau = \mathrm{TOL}$ if $\mathrm{TOL} > 0$, and $\tau = (n\epsilon)^2 d_{m_e+1}$ where ϵ is the smallest value for which $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ on the CDC 6700) if $\mathrm{TOL} = 0$. Thus, if $\mathrm{TOL} = 0$ then a tolerance based on the computer precision is used to determine the rank of N_1 of $\widetilde{WA}$. Otherwise, if $\mathrm{TOL} > 0$ then TOL is the tolerance that is used to specify the rank of the problem to be solved. If the user inadvertently sets TOL to be negative then L2SLV resets TOL to be 0.

IPIVOT is an array of dimension n or larger that is used by L2SLV to record the order in which the columns of $\widetilde{WA}$ are selected by the pivoting procedure when $\widetilde{WA}$ is reduced to $(A_1 A_2)$. If $N1 < n$ then the first $N1$ elements of IPIVOT are the indices of the columns of $\widetilde{WA}$ from which the matrix A_1 is generated.

T is a 2-dimensional array of dimension $kt \times n$ that is used for temporary storage. It is assumed that $kt \cdot m + n$. When L2SLV terminates, if $N1 = n$ then the unscaled $n \times n$ covariance matrix is stored in the first n rows and columns of T. Iterative improvement is not performed on the covariance matrix.

Error Return IERR is an integer variable that is set by the routine. If no input errors are detected and the results appear to be satisfactory, then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

- IERR = 1 Either m, n, or ℓ is not positive.
- IERR = 2 The restriction $0 < m_e < \min\{m, n\}$ is not satisfied.
- IERR = 3 Either $ka < m$, $kb < m$, $kx < n$, $kr < m$, or $kt < m + n$ is violated.
- IERR = 4 $\mathrm{WGTS}(i)$ is negative for some $i < m_e$.
- IERR = 5 Either $\widetilde{WA} = 0$ or $E = 0$.
- IERR = 6 The rank of E is less than m_e.
- IERR = 7 Iterative improvement of all the solutions $x_1, \ldots, x_\ell$ failed to converge.
- IERR = 8 Iterative improvement of one or more solutions failed to converge.
- IERR = 9 More than μ iterations of the iterative improvement procedure were performed in computing some x_j. (Here it is assumed that a

μ decimal digit floating-point arithmetic is being used. $\mu = 14$ for the CDC 6700.)

IERR = 10 The accuracy of some x_j before iterative improvement was estimated to be less than half a decimal digit.

IERR = 11 One or more of the computed diagonal elements of the covariance matrix is negative. This is due to roundoff error. Theoretically, all the diagonal elements should be nonnegative. No evidence of severe ill-conditioning was detected.

IERR = 12 One or more of the computed diagonal elements of the covariance matrix is negativ This is due to roundoff error. Theoretically, all the diagonal elements should be nonnegative. The problem appears to be extremely ill-conditioned.

When an input error is detected (IERR = $1, 2, \ldots, 6$) then L2SLV immediately terminates. If evidence of severe ill-conditioning is detected, then IERR is set to 8,9, or 10 and computation of the solutions continues. If iterative improvement appears to converge for one or more of the solutions, then the covariance matrix is also computed (when $N1 = n$). However, if iterative improvement fails for all the solutions $x_1, \ldots, x_\ell$ then IERR is set to 7 and the covariance matrix is not computed.

Note. WK(1), ...,WK(2ℓ) should be examined when severe ill-conditioning is detected.

Programming. L2SLV employs the subroutines DECOM2, SOLVE2, SOLVE3, and COVAR. These routines were written by Roy Wampler (National Bureau of Standards). L2SLV is a slightly modified version by A. H. Morris of the subroutine L2B discussed in reference (4). The algorithm employed for finding and iteratively improving the least squares solutions is described in references (1)–(3). The function SPMPAR is also used.

References.

(1) Bjorck, Ake, "Solving Linear Least Squares Problems by Gram-Schmidt Orthogonalization," ***BIT* 7** (1967), pp. 1–21.

(2) _____, "Iterative Refinement of Linear Least Squares Solutions I," ***BIT* 7** (1967), pp. 257–278.

(3) _____, "Iterative Refinement of Linear Least Squares Solutions II," ***BIT* 8** (1968), pp. 8–30.

(4) Wampler, Roy, "Solutions to Weighted Least Squares Problems by Modified Gram-Schmidt with Iterative Refinement," ***ACM Trans. Math Software* 5** (1979), pp. 457–465.

ITERATIVE LEAST SQUARES SOLUTION OF BANDED LINEAR EQUATIONS

Given an $m \times n$ matrix A, a column vector b of dimension m, and a real number λ. Let $\bar{A} = \begin{pmatrix} A \\ \lambda I \end{pmatrix}$ where I is the $n \times n$ identity matrix, and let $\bar{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}$. The problem is to find a column vector x of dimension n which is a least squares solution of $\bar{A}x = \bar{b}$. If A is stored in band form then the following subroutine is available for solving this problem.

CALL BLSQ($m, n, A, ka, m_\ell, m_u, \lambda, b, x$, ATOL,BTOL,CONLIM,MXITER, IND,ITER,COND,RNORM,XNORM,WK)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leq m_\ell < m, 0 \leq m_u < n$, and $ka \geq m$. When BLSQ is called, an iterative procedure is used to obtain a least squares solution x of $\bar{A}x = \bar{b}$. The vector b is modified by the routine.

ATOL and BTOL are input arguments which specify the relative accuracy of A and b respectively. For example, if it is estimated that b is accurate to k decimal digits then one may set BTOL $= 10^{-k}$. It is required that ATOL ≥ 0 and BTOL ≥ 0. If ATOL $= 0$ or BTOL $= 0$, then it is assumed that A or b is accurate to machine precision.

Let cond($\bar{A}$) denote the condition number of $\bar{A}$ relative to the Frobenius norm.[1] In each iteration of the algorithm being used, an estimate is made of the condition number cond($\bar{A}$). The estimates form a monotonically nondecreasing sequence. The input argument CONLIM is an upper limit on cond($\bar{A}$). If CONLIM > 0 then BLSQ terminates when an estimate of cond($\bar{A}$) exceeds CONLIM. This termination may be needed to prevent small or zero singular values of A from coming into effect and causing damage to the solution x. CONLIM may be ignored by being set to 0. It is assumed that CONLIM ≥ 0.

The input argument MXITER is the maximum number of iterations that are permitted. Normally BLSQ requires less than $4n$ iterations. The related argument ITER is a variable. When the routine terminates ITER = the number of iterations that were performed.

COND,RNORM, and XNORM are variables. When BLSQ terminates COND = the last estimate made for cond($\bar{A}$), RNORM = $\|\bar{A}x - \bar{b}\|$, and XNORM = $\|x\|$.[2]

WK is an array of dimension $2n$ or larger that is a work space for the routine.

The equations $\bar{A}x = \bar{b}$ are considered to be ***compatible*** if for any least squares solution x, $\|\bar{A}x - \bar{b}\| = 0$. IND is a variable that reports the status of the results. When BLSQ terminates, IND has one of the following values:

[1] cond $(\bar{A}) = \|\bar{A}\|_F \|\bar{A}^+\|_F$ where A^+ is the pseudoinverse of A. Here $\|C\|_F = \sqrt{\Sigma c_{ij}^2}$ for any matrix $C = (c_{ij})$.

[2] $\|c\| = \sqrt{\Sigma c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.

IND = 0 The solution is $x = 0$. No iterations were performed.

IND = 1 The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is sufficiently accurate, given the values ATOL and BTOL.

IND = 2 The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is sufficiently accurate, given the value ATOL.

IND = 3 An estimate COND of cond($\bar{A}$) exceeds CONLIM. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

IND = 4 The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is as accurate as seems reasonable on this machine.

IND = 5 The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is as accurate as seems reasonable on this machine.

IND = 6 cond($\bar{A}$) appears to be so large that there is not much point in doing further iterations. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

IND = 7 MXITER iterations were performed. More iterations are needed. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

Remarks.

(1) A large estimate of the condition number cond($\bar{A}$) may be due to rank deficiency or near rank deficiency of the matrix $\bar{A}$. If it is suspected that a large estimate of cond($\bar{A}$) has occurred for this reason, then it is recommended that CONLIM be set to a moderate value such as $\sqrt{\epsilon}$ where ϵ is the smallest value such that $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6000-7000 series computers). Setting CONLIM to 0 is equivalent to setting CONLIM to ϵ^{-1}.

(2) The vector b is the only input argument modified by the routine.

Algorithm. BLSQ employs an iterative algorithm developed by Golub and Kahan.

Programming. BLSQ calls the subroutines NORMLZ, BVPRD1, BTPRD1, SCOPY, and SSCAL. The function SNRM2 is also used. BSLQ is an adaptation by A. H. Morris of the subroutine LSQR, written by Christopher C. Paige (McGill University, Montreal, Canada) and Michael A. Saunders (Stanford University).

References.

(1) Paige, C. C. and Saunders, M. A., "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," ***ACM Trans. Math Software*** **8** (1982), pp. 43-71.

(2) _____, "Algorithm 583. LSQR: Sparse Linear Equations and Least Squares Problems," ***ACM Trans. Math Software*** **8** (1982), pp. 195-209.

ITERATIVE LEAST SQUARES SOLUTION OF SPARSE LINEAR EQUATIONS

Given a [illegible] n matrix A, a column vector b of dimension m, and a real number λ. Let $\bar{A} = \begin{pmatrix} A \\ \lambda I \end{pmatrix}$ [illegible] I is the $n \times n$ identity matrix, and let $\bar{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}$. The problem is to find a column vector [illegible] of dimension n which is a least squares solution of $\bar{A}x = \bar{b}$. If A is sparse then the following subroutines are available for solving this problem.

CALL SPLSQ(m, n, A, IA, JA, λ, b, x, ATOL, BTOL, CONLIM, MXITER, IND, ITER, COND, RNORM, XNORM, WK)

CALL STLSQ(m, n, TA, ITA, JTA, λ, b, x, ATOL, BTOL, CONLIM, MXITER, IND, ITER, COND, RNORM, XNORM, WK)

If SPLSQ is called then A, IA, JA are arrays containing the matrix A in sparse form. Otherwise, if STLSQ is called then TA, ITA, JTA are arrays containing the transpose matrix A^t in sparse form. An iterative procedure is used to obtain a least squares solution x of $\bar{A}x = \bar{b}$. The vector b is modified by the routines.

ATOL and BTOL are input arguments which specify the relative accuracy of A and b respectively. For example, if it is estimated that b is accurate to k decimal digits then one may set BTOL $= 10^{-k}$. It is required that ATOL ≥ 0 and BTOL ≥ 0. If ATOL $= 0$ or BTOL $= 0$, then it is assumed that A or b is accurate to machine precision.

Let cond$(\bar{A})$ denote the condition number of $\bar{A}$ relative to the Frobenius norm.[1] In each iteration of the algorithm being used, an estimate is made of the condition number cond$(\bar{A})$. The estimates form a monotonically nondecreasing sequence. The input argument CONLIM is an upper limit on cond $(\bar{A})$. If CONLIM > 0 then the routines terminate when an estimate of cond$(\bar{A})$ exceeds CONLIM. This termination may be needed to prevent small or zero singular values of $\bar{A}$ from coming into effect and causing damage to the solution x. CONLIM may be ignored by being set to 0. It is assumed that CONLIM ≥ 0.

The input argument MXITER is the maximum number of iterations that are permitted. Normally the routines require less than $4n$ iterations. The related argument ITER is a variable. When the routines terminate ITER = the number of iterations that were performed.

COND, RNORM, and XNORM are variables. When the routines terminate COND = the last estimate made for cond$(\bar{A})$, RNORM $= \|\bar{A}x - \bar{b}\|$, and XNORM $= \|x\|$.[2]

WK is an array of dimension $2n$ or larger that is a work space for the routines.

---

[1] cond $(\bar{A}) = \|\bar{A}\|_F \|A^+\|_F$ where $\bar{A}^+$ is the pseudoinverse of $\bar{A}$. Here $\|C\|_F = \sqrt{\Sigma c_{ij}^2}$ for any matrix $C = (c_{ij})$.

[2] $\|c\| = \sqrt{\Sigma c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.

The equations $\bar{A}x = \bar{b}$ are considered to be ***compatible*** if for any least squares solution x, $\|\bar{A}x - \bar{b}\| = 0$. IND is a variable that reports the status of the results. When the routines terminate, IND has one of the following values:

IND = 0 The solution is $x = 0$. No iterations were performed.

IND = 1 The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is sufficiently accurate, given the values ATOL and BTOL.

IND = 2 The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is sufficiently accurate, given the value ATOL.

IND = 3 An estimate COND of cond($\bar{A}$) exceeds CONLIM. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

IND = 4 The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is as accurate as seems reasonable on this machine.

IND = 5 The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is accurate as seems reasonable on this machine.

IND = 6 cond($\bar{A}$) appears to be so large that there is not much point in doing further iterations. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

IND = 7 MXITER iterations were performed. More iterations are needed. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

Remarks.

(1) A large estimate of the condition number cond($\bar{A}$) may be due to rank deficiency or near rank deficiency of the matrix $\bar{A}$. If it is suspected that a large estimate of cond($\bar{A}$) has occurred for this reason, then it is recommended that CONLIM be set to a moderate value such as $\sqrt{\epsilon}$ where ϵ is the smallest value such that $1+\epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6000–7000 series computers). Setting CONLIM to 0 is equivalent to setting CONLIM to ϵ^{-1}.

(2) The vector b is the only input argument modified by the routine.

Algorithm. SPLSQ and STLSQ employ an iterative algorithm developed by Golub and Kahan.

Programming. SPLSQ and STLSQ call the subroutines NORMLZ, MVPRD1, MTPRD1, SCOPY, and SSCAL. The function SNRM2 is also used. SPLSQ and STLSQ are adaptations by A. H. Morris of the subroutine LSQR, written by Christopher C. Paige (McGill University, Montreal, Canada) and Michael A. Saunders (Stanford University).

References.

(1) Paige, C. C. and Saunders, M. A., "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," ***ACM Trans. Math Software*** **8** (1982), pp. 43–71.

(2) _____, "Algorithm 583. LSQR: Sparse Linear Equations and Least Squares Problems," ***ACM Trans. Math Software*** **8** (1982), pp. 195–209.

MINIMIZATION OF FUNCTIONS OF A SINGLE VARIABLE

Let $F(x)$ be a continuous real-valued function defined for $a \leq x \leq b$. Then the following subroutine is available for finding a local minimum of $F(x)$.

CALL FMIN(F, a, b, x, w, AERR,RERR,ERROR,IND)

It is assumed that $a \leq b$. FMIN finds a value x in the interval $[a, b]$ which is a local minimum of F. ERROR and w are variables. When FMIN terminates, $w = F(x)$ and ERROR is the estimated maximum absolute error of x.

The input arguments AERR and RERR are the absolute and relative error tolerances to be satisfied. For example, if k significant digit accuracy is desired then one may set RERR $= 10^{-k}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. The setting AERR $= 0$ is equivalent to the setting AERR $= 10^{-20}$, and the setting RERR $= 0$ is a request for machine precision.

IND is a variable that reports the status of the results. IND $= 0$ if x is found to the desired accuracy. Otherwise, IND $= 1$ when x cannot be obtained to the desired accuracy. In this case, w satisfies the tolerances AERR and RERR.

Note. F must be declared in the calling program to be of type EXTERNAL.

Algorithm. The golden section search procedure is used.

Programming. The function SPMPAR is called. FMIN was written by A. H. Morris.

MINIMIZATION OF FUNCTIONS OF N VARIABLES

Let $f(x)$ be a real-valued function of n variables $x = (x_1,\ldots,x_n)$ where $n \geq 2$. If $f(x)$ is twice continuously differentiable then the following subroutine is available for finding a local minimum of $f(x)$.

CALL OPTF(F, n,RERR,ITER,X,FVAL,IND,WK)

X is an array of dimension n and FVAL a variable. On input, X contains an initial guess $a = (a_1,\ldots,a_n)$ to a minimum of f. When OPTF terminates, X contains the final estimate $x = (x_1,\ldots,x_n)$ of a local minimum of f and FVAL $= f(x)$.

The argument F is the name of a user defined subroutine that has the format:

CALL $F(n, X$,FVAL)

Here X is an array of dimension n containing a point $x = (x_1,\ldots,x_n)$, and FVAL is a variable. F sets FVAL to the value of the function f at the point x. F must be declared in the calling program to be of type EXTERNAL.

RERR is an input argument that specifies the relative accuracy of F. If it is estimated that the subroutine F produces results accurate to k significant digits then one may set RERR $= 10^{-k}$. It is required that RERR ≥ 0. If RERR $= 0$ then it is assumed that F produces results accurate to machine precision.

When OPTF is called, line search iteration is performed to find the local minimum of f. ITER is a variable. On input, ITER is the maximum number of iterations that are permitted. When the routine terminates, ITER = the number of iterations that were actually performed.

WK is an array of dimension $n(n+8)$ or larger that is a work space for the routine.

IND is a variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND $= -1$ (Input error) $n \leq 0$.
IND $= -2$ (Input error) $n = 1$.
IND $= -4$ (Input error) ITER ≤ 0.
IND $= -5$ (Input error) Either RERR < 0 or RERR $> 10^{-4}$.
IND $= 1$ A local minimum x was found. The gradient of f at x was considered to be sufficiently small.
IND $= 2$ The steps taken became so small that OPTF had to terminate. X is probably a local minimum, but it need not be a local minimum. The algorithm frequently requires exceedingly small steps to be taken, no matter whether X is close to or far from a local minimum.
IND $= 3$ A local minimum has possibly been found. OPTF could not find a point for which f would take a smaller value.
IND $= 4$ ITER iterations were performed.
IND $= 5$ The algorithm appears to be diverging. This generally occurs when f is unbounded from below.

When an input error is detected, the routine immediately terminates.

Accuracy and Efficiency. OPTF is frequently extremely efficient in finding a value FVAL which roughly approximates a local minimum value $f(x_0)$, but at times it can be quite slow in obtaining FVAL to greater precision. A rough approximation is often obtained in 20–30 iterations. If $f(x_0) \neq 0$ then FVAL may be accurate to 4–6 significant digits after 20–30 additional iterations, or FVAL may not be accurate to 1 significant digit after several hundred iterations. 4–6 digit accuracy is the greatest precision that can be expected. In general, it is recommended that ITER $\leq$ 200. Each iteration can take considerable time, even if the subroutine F is cheap to evaluate.

Remark. OPTF can be quite sensitive to the scaling of the variables $(x_1, \ldots, x_n)$. The routine tends to operate more efficiently when the components of a local minimum $x = (x_1, \ldots, x_n)$ are all roughly of the same magnitude. If the components are of considerably different magnitudes (say $|x_1| \approx 10^{-6}$ and $|x_2| \approx 10^3$) then convergence may be extremely slow. In such a case, OPTF attempts to rescale the variables, but the rescaling is not always helpful.

Algorithm. The line search algorithm given in pp. 325–327 of the reference is employed. Also, BFGS secant updates for the hessian are used.

Programming. OPTF employs the subroutines OPTDRV, OPCHK1, OPSTP, FXDEC, SCALEX, LLTSLV, FSTOFD, FSTOCD, LNSRCH, SECFAC, QRUPDT, and JROT. OPTF, FXDEC, and SCALEX were written by A.H. Morris. The remaining subroutines were written by Robert B. Schnabel (University of Colorado at Boulder) and modified by A.H. Morris. The functions SDOT, SNRM2, and SPMPAR are also used.

Reference. Dennis, J.E. and Schnabel, R.B., ***Numerical Methods for Unconstrained Optimization and Nonlinear Equations***, Prentice-Hall, Englewood Cliffs, NJ, 1983.

UNCONSTRAINED MINIMUM OF THE SUM OF SQUARES OF NONLINEAR FUNCTIONS

Let $f_1(x), \ldots, f_m(x)$ be m real-valued functions of n real variables $x = (x_1, \ldots, x_n)$ where $m \geq n$. The problem under consideration is to find a point x which minimizes the function $\phi(x) = \sum_{i=1}^{m} f_i(x)^2$. Assume that each $f_i(x)$ is differentiable and that an initial guess $a = (a_1, \ldots, a_n)$ to a minimum of $\phi(x)$ is given. Then the following subroutine is available for finding a point which minimizes $\phi(x)$.

CALL LMDIFF$(F, m, n, X,$ FVEC,EPS,TOL,INFO,IWK,WK,$\ell)$

X is an array of dimension n and FVEC an array of dimension m. On input X contains the starting point $a = (a_1, \ldots, a_n)$. When LMDIFF terminates, X contains the final estimate $x = (x_1, \ldots, x_n)$ of a minimum of ϕ and FVEC contains the values of the functions $f_1, \ldots, f_m$ at the output point in X.

The argument F is the name of a user defined subroutine that has the format:

CALL $F(m, n, X,$ FVEC,IFLAG)

Here X is an array of dimension n, FVEC an array of dimension m, and IFLAG an integer variable. The array X contains a point $x = (x_1, \ldots, x_n)$. Normally F evaluates the functions $f_1, \ldots, f_m$ at this point and stores the results in FVEC. However, if x does not lie in the domain of $f_1, \ldots, f_m$ then this cannot be done. In this case, the argument IFLAG (which will have been assigned a nonnegative value by LMDIFF) should be reset by F to a negative value. This will signal LMDIFF to terminate. F must be declared in the calling program to be of type EXTERNAL.

EPS is an input argument which specifies the relative accuracy of F. If it is estimated that the subroutine F produces results accurate to k significant decimal digits then one may set EPS $= 10^{-k}$. It is required that EPS ≥ 0. If EPS $= 0$ then it is assumed that F produces results accurate to machine precision.

TOL is an input argument which specifies the desired accuracy to be attained. The Euclidean norm $||x|| = \sqrt{\Sigma_i x_i^2}$ is employed. If $\bar{x}$ denotes an actual minimum of ϕ, then LMDIFF terminates when an iterate x is generated for which it is estimated that

(1) $\phi(x) \leq (1 + \text{TOL})^2 \phi(\bar{x})$ or

(2) $||D(x - \bar{x})|| \leq \text{TOL} \cdot ||D\bar{x}||$

is satisfied. In (2) x and $\bar{x}$ are regarded as column vectors, and D is a diagonal matrix generated by LMDIFF whose entries are scaling factors. For convenience, criterion (1) is called the F-convergence (or ϕ-convergence) test and criterion (2) is called the x-convergence test. It is required that TOL ≥ 0. In order for the convergence tests to work properly, it is recommended that TOL always be smaller than 10^{-5}.

IWK is an array of dimension n and WK is an array of dimension ℓ. IWK and WK are work spaces. It is assumed that $\ell \geq mn + 5n + m$.

INFO is an integer variable that reports the status of the results. When LMDIFF terminates, INFO has one of the following values:

INFO < 0 This occurs when the user terminates the execution of LMDIFF by resetting the argument IFLAG in the subroutine F to a negative value. Then INFO = the negative value of IFLAG.

INFO $= 0$ (Input error)$1 \leq n \leq m$, EPS ≥ 0, TOL ≥ 0,or $\ell \geq mn + 5n + n$ is violated.

INFO $= 1$ The F-convergence test has been satisfied.

INFO $= 2$ The x-convergence test has been satisfied.

INFO $= 3$ The F-convergence and x-convergence tests have been satisfied.

INFO $= 4$ The gradient of ϕ is 0 at point X.

INFO $= 5$ The number of calls to the subroutine F has reached or exceeded $200(n + 1)$.

INFO $= 6$ TOL is too small. No further reduction in the value of $\phi(x)$ is possible.

INFO $= 7$ TOL is too small. No further improvement in the accuracy of X is possible.

When LMDIFF terminates, if INFO $\neq 0$ then X contains the final iterate that was generated. Also, if INFO ≥ 1 then FVEC contains the values of the functions $f_1, \ldots, f_m$ at this iterate. If INFO $= 4$ then X should be examined very closely. The gradient of ϕ can be 0 when X is a local minimum or maximum, or when X is a saddle point. If INFO $= 5$ then it may (or may not) be helpful to continue the procedure by recalling LMDIFF with the current point in X as the new starting point. Since TOL is a relative tolerance, this setting can occur when $\phi(0) = 0$.

Algorithm. A modified form of the Levenberg-Marquardt algorithm is employed.

Programming. LMDIFF is a slightly modified version of the MINPACK-1 subroutine LMDIF1. The MINPACK-1 subroutines LMDIF, SPMPAR, ENORM, FDJAC2, LMPAR, QRFAC, and QRSOLV are employed. The subroutines were written by Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom (Argonne National Laboratory).

References.

(1) More, J. J., Garbow, B. S., and Hillstrom, K. E., ***User Guide for MINPACK***-1, Argonne National Laboratory Report ANL-80-74, Argonne, Illinois, 1980.

(2) More, J. J., "The Levenberg-Marquardt Algorithm: Implementation and Theory," ***Numerical Analysis***, G. A. Watson (ed.), Springer-Verlag, 1977.

LINEAR PROGRAMMING

Let $A = (a_{ij})$ be an $m \times n$ matrix, B an array containing $b_1, \ldots, b_m$, and C an array containing $c_1, \ldots, c_n$ where a_{ij}, b_i, c_j are real. Consider the problem of finding nonnegative values $x_1, \ldots, x_n$ which maximize or minimize the function $c_1x_1 + \cdots + c_nx_n$ subject to the constraints:

$$a_{11}x_1 + \cdots + a_{1n}x_n\{\leq, =, \geq\}b_1$$

$$\vdots$$

$$a_{m1}x_1 + \cdots + a_{mn}x_n\{\leq, =, \geq\}b_m$$

In each constraint

$$a_{i1}x_1 + \cdots + a_{in}x_n\{\leq, =, \geq\}b_i$$

only one of the relations $\leq, =, \geq$ is used, but the relation may vary from constraint to constraint. The following subroutines are available for solving this problem.

CALL SMPLX(A, B, C, ka, m, n, IND,IBASIS, X, z, ITER,MXITER,
NUMLE,NUMGE,BI,WK,IWK)
CALL SSPLX(TA,ITA,JTA, B, C, m, n, IND,IBASIS, X, z, ITER,MXITER,
NUMLE,NUMGE,BI,WK,IWK)

It is assumed that $m \geq 2, n \geq 2$, and that each $b_i \geq 0$. If SMPLX is called then ka is the number of rows in the dimension statement for A in the calling program. Otherwise, if SSPLX is called then TA, ITA, JTA are arrays containing the transpose matrix A^t in sparse form.

The constraints $a_{i1}x_1 + \cdots + a_{in}x_n\{\leq, =, \geq\}b_i$ are assumed to be ordered so that the $\leq$ constraints are followed by the $\geq$ constraints, and the $=$ constraints come last. NUMLE and NUMGE have the values:

NUMLE = the number of $\leq$ constraints.
NUMGE = the number of $\geq$ constraints.

It is assumed that NUMLE ≥ 0, NUMGE ≥ 0, and NUMLE + NUMGE $\leq m$.

When SMPLX or SSPLX is called, the routine attempts to maximize $\Sigma_j c_j x_j$ subject to the constaints. A modified form of the primal simplex algorithm is employed. Frequently the procedure requires less than $5m$ iterations to perform the task. The argument MXITER has the value:

MXITER = the maximum number of iterations that may be performed.

This argument is provided by the user. The related argument ITER is a variable that is set by the routine. When the routine terminates, ITER has for its value the number of iterations that were performed.

IND is a variable and IBASIS an array of dimension m. IBASIS contains the indices i of the current basic variables x_i, and IND is used for input/output purposes. On input IND is normally set by the user to 0. If IND $\neq$ 0 then the routine selects its own

beginning basis and stores the appropriate indices in IBASIS. [The remainder of this paragraph may be skipped by anyone not acquainted with the simplex algorithm.] If the user wishes to use his own beginning basis, then IND must be set to 1 and the indices of the initial basic variables stored in IBASIS. ***It is not required that the initial basis be selected so that the basic variables are nonnegative***. The initial basic variables may be original, slack, surplus, or artificial variables. Slack and surplus variables are automatically provided for the $\leq$ and $\geq$ constraints, and artificial variables for the $=$ constraints. The routine defines x_{n+i} to be the slack, surplus, or artificial variable for the i^{th} constraint $a_{i1}x_1 + \cdots + a_{in}x_n\{\leq, =, \geq\}b_i$. If IND = 0 then the slack, surplus, and artificial variables are the initial basic variables that are employed.

On output IND reports the status of the results. The routine assigns IND one of the following values:

IND = 0 The problem was solved.
IND = 1 The problem has no solution.
IND = 2 MXITER iterations were performed. More iterations are needed.
IND = 3 Sufficient accuracy cannot be maintained to solve the problem.
IND = 4 The problem has an unbounded solution.
IND = 5 An input error was detected. (See below)
IND = 6 A possible solution was obtained. The routine is not certain if the solution is correct.

X is an array of dimension n + NUMLE + NUMGE and z is a variable. If IND = 0 or IND = 6, then z has for its value the maximum value obtained for $\Sigma_j c_j x_j$ and X contains the values obtained for the original, slack, and surplus variables. If IND $\neq$ 5 then IBASIS contains the indices of the basic variables currently in effect when the routine terminates.

BI is an array of dimension m^2 that is used for storing the inverse of the basis matrix. The order of the column vectors of the basis matrix corresponds to the order of the basic variables given in IBASIS. If IND $\neq 3, 5$ on output then BI contains the inverse of the basis matrix currently in effect when the routine terminates.

WK is an array of dimension $2m$ or larger, and IWK is an array of dimension $2m + n$ or larger. WK and IWK are work spaces.

Input Errors. IND = 5 occurs on output when one of the following conditions is violated:

(1) $n > 2$ and ka $> m > 2$.
(2) NUMLE + NUMGE $\leq m$.
(3) Each $b_i > 0$.
(4) The basis matrix specified by the user in IBASIS (when IND = 1 on input) is nonsingular and sufficiently well conditioned so that its inverse can be computed.

Remarks.

(1) A, B, C,TA,ITA,JTA are not modified by the routines.
(2) The routines maximize $\Sigma_j c_j x_j$. This function can be minimized by maximizing $\Sigma_j(-c_j)x_j$ and then changing the sign of the result.

(3) SMPLX and SSPLX generate the same results. For efficiency, SSPLX should be used when A is sparse.

Algorithm. A three step procedure is used. The first step eliminates the negative variables. Then phases (1) and (2) of the primal simplex algorithm are invoked. Negative variables are eliminated as follows: Let $x_{B1}, \ldots, x_{Bm}$ be the basic variables and y_{ij} the components of the simplex tableau.

(1) Compute $d_j = \Sigma'_i y_{ij}$ for each nonbasic variable x_j where the sum Σ'_i is for all i where $x_{Bi} < 0$. If all $d_j \geq 0$ then the problem has no feasible solution. Otherwise, select k so that $d_k = min_j d_j$. Then x_k is the variable to be made basic.

(2) If $x_{Bj} \geq 0$ and $y_{jk} > 0$ for some j then go to (3). Otherwise, select a negative variable x_{Br} to become nonbasic where $x_{Br}/y_{rk} = \max\{x_{Bj}/y_{jk} : x_{Bj} < 0 \text{ and } y_{jk} < 0\}$. Then update the basis and go to (5).

(3) Compute $\epsilon = \min\{x_{Bj}/y_{jk} : x_{Bj} \geq 0 \text{ and } y_{jk} > 0\}$ and check if a negative variable x_{Bj} exists that satisfies the conditions:

($\star$) $y_{jk} < 0$ and $\epsilon \geq x_{Bj}/y_{jk}$

If such a variable exists then go to (4). Otherwise, select a nonnegative variable x_{Br} to become nonbasic where $y_{rk} > 0$ and $x_{Br}/y_{rk} = \epsilon$. Then update the basis and go to (1).

(4) Select a negative variable x_{Br} to become nonbasic where $x_{Br}/y_{rk} = \max\{x_{Bj}/y_{jk} : x_{Bj} < 0$ and x_{Bj} satisfies($\star$)$\}$. Then update the basis and go to (5).

(5) Check if there are any remaining negative variables. If not, then we are finished. Otherwise go to (1).

Programming. SMPLX and SSPLX employ the subroutines SMPLX1, SSPLX1, and CROUT1. These routines were written by A. H. Morris. The function SPMPAR is also used.

THE ASSIGNMENT PROBLEM

Let $C = (c_{i,j})$ be an $n \times n$ matrix (the cost matrix). The problem under consideration is to find an $n \times n$ matrix $x = (x_{i,j})$ which minimizes $T = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j}$ and satisfies:

(1) $\sum_{i=1}^{n} x_{i,j} = 1$ for $j = 1, \ldots, n$

(2) $\sum_{j=1}^{n} x_{i,j} = 1$ for $i = 1, \ldots, n$

(3) Each $x_{i,j} = 0$ or 1

Each x which satisfies (1)–(3) is called an ***assignment***. For each such x, from (1) and (3) we note that for each j there exists a unique integer $\pi(j)$ such that $x_{\pi(j),j} = 1$. Also, (2) and (3) assert that π is a permutation of $\{1, \ldots, n\}$. Conversely, for any permutation π there corresponds an assignment x defined by $x_{\pi(j),j} = 1$ and $x_{i,j} = 0$ for $i \neq \pi(j)$. Thus, the problem is to find a permutation π of $\{1, \ldots, n\}$ which minimizes $T = \sum_{j=1}^{n} c_{\pi(j),j}$. The following subroutine is available for solving this problem when all $c_{i,j}$ are integers.

CALL ASSGN(n, C, JC, T, IWK, IERR)

C is a 2-dimensional integer array of dimension $n \times (n + 1)$, JC an integer array of dimension n, and T an integer variable. It is assumed that $n > 2$ and that the first n columns of C contain the cost matrix $(c_{i,j})$. [The $(n + 1)^{st}$ column of C is a work space for the routine.] When ASSGN is called, the desired permutation π is obtained and the values $\pi(1), \ldots, \pi(n)$ stored in JC. Also T is assigned the minimized value $\Sigma_j c_{\pi(j),j}$.

IWK is an array of dimension $7n + 2$ or larger that is a work space for the routine.

IERR is a variable that is set by the routine. If JC and T are obtained then IERR is assigned the value 0. Otherwise, if the problem cannot be solved because of integer overflow, then IERR = 1.

Remarks

(1) C is destroyed by the routine.

(2) ASSGN minimizes $T = \Sigma_j c_{\pi(j),j}$. This function can be maximized by minimizing $\Sigma_j(-c_{\pi(j),j})$ and then changing the sign of the result.

Programming ASSGN calls the subroutine ASSGN1. ASSGN1 was written by Giorgio Carpaneto and Paolo Toth (University of Bologna, Italy), and modified by A. H. Morris. The function IPMPAR is also used.

Reference Carpaneto, G., and Toth, P., "Algorithm 548, Solution of the Assignment Problem," ***ACM Trans. Math. Software*** 6 (1980) pp. 104–111.

0–1 KNAPSACK PROBLEM

Given $n \geq 2$ items, each having a profit $p_j > 0$ and weight $w_j > 0$, and $m \geq 1$ containers (knapsacks), each having a capacity $k_i > 0$. Let $x_{ij} = 1$ if item j is assigned to knapsack i. Otherwise, let $x_{ij} = 0$. Then the problem is to find an assignment x_{ij} of the items to the knapsacks which maximizes $\sigma = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$ subject to

(1) $\sum_{j=1}^{n} w_j x_{ij} \leq k_i$ for $i = 1, \ldots, m$, and

(2) $\sum_{i=1}^{m} x_{ij} \leq 1$ for $j = 1, \ldots, n$.

Condition (2) states that each item may be assigned to a (single) knapsack or be rejected. It can be assumed, without loss of generality, that

(a) the knapsacks are ordered so that $k_1 \leq \cdots \leq k_m$,
(b) $\min\{w_1, \ldots, w_n\} \leq k_1$,
(c) $\max\{w_1, \ldots, w_n\} \leq k_m$, and
(d) $\sum_{j=1}^{n} w_j > k_n$.

Then the following subroutine is available for solving this problem when all p_j, w_j, and k_i are positive integers.

CALL MKP(n, m, P, W, K,NBCK,L, σ,TEMP,ITEMP,NUM)

P and W are integer arrays containing $p_1, \ldots, p_n$ and $w_1, \ldots, w_n$, and K an integer array containing $k_1, \ldots, k_m$. L is an integer array of dimension n or larger, and σ an integer variable. When MKP is called, if no input errors are detected then the maximum value for σ is obtained. Also, $L(j) = i$ if item j has been assigned to knapsack i ($j = 1, \ldots, n$), and $L(j) = 0$ if item j does not appear in the solution (i.e., if $x_{ij} = \cdots = x_{nj} = 0$).

A depth-first tree search employing backtracking is used. If no bound is placed on the number of back tracks that may be performed, then the exact maximum σ is assured. However, if the number of back tracks must be restricted, then only an approximation to the maximum may be obtained. The argument NBCK is available for limiting the backtracking. NBCK is a variable. On input, if NBCK $= -1$ then no restrictions are placed on the backtracking. Otherwise, if NBCK $\neq -1$, then it is assumed that NBCK is the maximum number of back tracks that are permitted. When the routine terminates, NBCK = the number of back tracks that were actually performed.

TEMP is a real array of dimension n or larger, and ITEMP an integer array of dimension NUM. It is assumed that NUM $\geq 5m + 14n + 4mn + 3$. TEMP and ITEMP are work spaces for the routine. It should be noted that TEMP is the only argument of MKP that is not of integer type.

Error return. If an input error is detected then σ is set to one of the following values:

$\sigma = -1$ if $m < 1$ or $n < 2$.

$\sigma = -2$ if some p_j, w_j, or k_i is not positive.
$\sigma = -3$ if $\min\{w_1, \ldots, w_n\} > k_1$; i.e., if a knapsack cannot contain any item.
$\sigma = -4$ if $\max\{w_1, \ldots, w_n\} > k_m$; i.e., if an item cannot fit in any knapsack.
$\sigma = -5$ if $\Sigma_j w_j \leq k_m$; i.e., if knapsack m can contain all the items.
$\sigma = -7$ if the knapsacks are not ordered so that $k_1 \leq \cdots \leq k_m$.
$\sigma = -8$ if NUM $< 5m + 14n + 4mn + 3$.

Backtracking. For NBCK $= -1$, the time required for finding the exact maximum depends primarily on the value of m, and can increase quite dramatically for very small increases in the value of m. It is recommended that this setting never be used when $m > 10$. Instead, if $m > 10$ then a setting of NBCK ≤ 50 frequently suffices.

Programming. MKP employs the subroutines MKP1, SIGMA1, PI1, PARC, SKNP, and SKNP1. The interface subroutine MKP was written by A.H. Morris. The remaining subroutines were written by Silvano Martello (University of Bologna) and Paolo Toth (University of Florence) and modified by A.H. Morris. The subroutine RISORT is also used.

Reference. Martello, S. and Toth, P.,"Algorithm 632. A Program for the 0-1 Multiple Knapsack Problem," ***ACM Trans. Math Software*** **11** (1985), pp. 135–140.

INVERSION OF THE LAPLACE TRANSFORM

Let $f(t)$ be a complex-valued function that is continuous for $t > 0$ except possibly at a countable set of values t_k having no finite limit points. Then for complex z the Laplace transform $F(z)$ of $f(t)$ is defined by

$$F(z) = \int_0^\infty e^{-zt} f(t)\, dt$$

when the integral converges. If the integral converges for $Re(z) > c$ but does not exist for $Re(z) < c$, then c is called the ***abscissa of convergence of*** $F(z)$. If $\int_0^\infty |f(t)|e^{-at}\, dt < \infty$ for some real constant a, then $F(z)$ is analytic for all z where $Re(z) > a$. Also, for any point t for which $f(t)$ is continuous and sufficiently well-behaved, the value $f(t)$ can be obtained from F by the inversion formula

$$f(t) = \frac{1}{2\pi i} \lim_{T\to\infty} \int_{\alpha - iT}^{\alpha + iT} e^{\lambda t} F(\lambda)\, d\lambda \qquad (i = \sqrt{-1})$$

for any $\alpha > a$. If $f(t)$ is real for $t > 0$, then $\overline{F(z)} = F(\bar{z})$. Given a transform $F(z)$ where $\overline{F(z)} = F(\bar{z})$, then the following subroutine is available for computing $f(t)$ for $t > 0$.

CALL LAINV(MO,FUN,t,AERR,RERR,Y,c,ERROR,NUM,IERR)

It is assumed that $\overline{F(z)} = F(\bar{z})$. The argument FUN is the name of a user defined subroutine for computing $F(z)$. FUN has the format:

CALL FUN(x, y, A, B)

A and B are variables. For real arguments x and y, A and B are assigned the values $A = \mathrm{Re}[F(x + iy)]$ and $B = \mathrm{Im}[F(x + iy)]$. FUN must be declared in the calling program to be of type EXTERNAL.

IERR is a variable that is used both for input and output purposes. If IERR ≥ 0 on input then it is assumed that the abscissa of convergence is known and that c = the abscissa of convergence. Otherwise, if IERR< 0 then the abscissa of convergence must be computed. In this case, c is a variable. LAINV sets c to the value that is obtained for the abscissa of convergence.

MO is an integer which specifies the search procedure to be used for finding the abscissa of convergence c when IERR < 0 on input. A one-pass procedure is employed when MO $\neq 0$, and a two-pass procedure when MO $= 0$. Normally, the one-pass procedure should be used. However, if all the singularities of $F(z)$ are known to be real, then the two-pass procedure (MO $= 0$) will be more efficient.

It is assumed that $t > 0$ and that Y is a variable. When LAINV is called Y is set to the value obtained for $f(t)$.

AERR and RERR are the absolute and relative error tolerances to be used in computing $F(t)$ (AERR ≥ 0 and RERR > 0). If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that $f(t)$ is to be computed to machine

accuracy. LAINV attempts to find a value Y which satisfies $|Y - f| < \max\{ AERR, RERR \cdot f(t) \}$.

ERROR and NUM are variables that are set by the routine. When LAINV terminates, ERROR is a rough estimate of the absolute error $|Y - f(t)|$ and NUM is the number of calls that were made to the subroutine FUN.

When LAINV terminates, IERR reports the status of the results. IERR is assigned one of the following values:

IERR = 0 Y was obtained to the desired accuracy.

IERR = 1 Y was obtained, but it may not be accurate because of inaccuracy in the computation of c. This setting occurs only when IERR < 0 on input.

IERR = 2 Y could not be obtained, possibly because too much accuracy was requested. Increase AERR and RERR, and rerun the problem.

IERR = 3 Y could not be obtained, possibly because of inaccuracy in the computation of c or too much accuracy was requested. Increase AERR and RERR, and rerun the problem. This setting occurs only when IERR < 0 on input.

IERR = 4 (Input error) The argument t is not positive. Y and ERROR are assigned the values 0 and 1.

IERR = 5 The abscissa of convergence c could not be found in the interval $[-10^4, 10^4]$. Y, c, and ERROR are assigned the values 0, 0, and 1. This setting occurs only when IERR < 0 on input.

IERR = 6 The argument t is too large for $f(t)$ to be computed. Y and ERROR are assigned the values 0 and 1.

Remarks.

(1) Accuracy decreases when t is near a discontinuity of $f(t)$.

(2) The calculation may lose accuracy or fail when $F(t)$ is oscillatory.

Algorithm. Given c, $f(t)$ is computed by a modification of the subroutine DLAINV developed by R. Piessens and R. Huysmans, where the real Wynn ϵ-algorithm has been replaced with the complex Wynn ϵ-algorithm.

When IERR < 0, c is calculated by the subroutine ABCON or the subroutine ABCON1. In ABCON, which is a two-pass search procedure, the abscissa x_1 of the rightmost singularity in the strip $-10^4 \le x \le 10^4$, $|y| \le .01$ is first determined. Then the abscissa of the rightmost singularity in the half-plane $\mathrm{Re}(z) \ge x_1$ is found. In ABCON1 these calculations are combined into a single-pass procedure.

In ABCON and ABCON1, the function $F(z)/(z - x_1 + 1)^2$ is integrated along paths C_1 and C_2 defined as follows: C_1 is the straight line segment from $(x_1, 0)$ to $(x_1, .01)$, followed by the straight line segment from $(x_1, .01)$ to $(\infty, .01)$, and C_2 is the straight line segment from (x_1, ∞) to $(x_1, 0)$, followed by the straight line segment from $(x_1, 0)$ to $(\infty, 0)$. The integral along C_1 vanishes if no singularity lies to the right of x_1 in the strip $|y| \le .01$, and the integral along C_2 vanishes if no singularity lies in $\mathrm{Re}(z) \ge x_1$. Otherwise, these integrals are nonzero in most applications. Simpson's rule suffices for integrating along the

finite line segment from $(x_1, 0)$ to $(x_1, .01)$.

Example. Let $F(z) = 1/(1+z^2)$, in which case $f(t) = \sin t$. The following code may be used for computing $f(t)$ at $t = 1, 1.1, 1.2, \ldots, 1.9$ and storing the results in the array W.

```
      REAL W(10)
      EXTERNAL FT
C
      AERR = 1.E-30
      RERR = 1.E-12
      IERR = -1
      T = 1.0
      DO 10 I = 1,10
         CALL LAINV (1, FT, T, AERR, RERR, W(I), C, ERR, N, IERR)
         IF (IERR .GT. 1) STOP
         T = T + 0.1
10    CONTINUE
```

Here FT may be defined by:

```
      SUBROUTINE FT(X, Y, A, B)
      COMPLEX Z,W
C
      Z = CMPLX(X,Y)
      W = 1.0/(1.0 + Z**2)
      A = REAL(W)
      B = AIMAG(W)
      RETURN
      END
```

Programming. LAINV employs the subroutines ABCON, ABCON1, SRCH, ACOND, XCOND, LAINV1, CQEXT, QAGI1, QAGIE1, QELG, QK15I1, QPSRT, CDIVID, CREC and functions ACONDF, ACONDG, XCONDX, XCONDY, SPMPAR, EXPARG. LAINV and ABCON were written by Andrew H. van Tuyl (NSWC) and modified by A.H. Morris. ABCON1 was written by A.H. Morris. LAINV1 was written by Robert Piessens and Rudi Huysmans (University of Leuven, Herverlee, Belgium) and modified by Andrew van Tuyl. QAGI1 is a modification of QAGI by Andrew van Tuyl and A.H. Morris.

Reference. Piessens, R. and Huysmans, R., "Algorithm 619. Automatic Numerical Inversion of the Laplace Transform," ***ACM Trans. Math Software*** **10** (1984), pp. 348–353.

FAST FOURIER TRANSFORM

Let n be a positive integer and $\theta_j = 2\pi j/n$ for $j = 0, 1, \ldots, n-1$. For any complex valued functions f and g defined on the points θ_j let $(f,g) = \sum_{j=0}^{n-1} f(\theta_j)\,\overline{g(\theta_j)}$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_j. Also $e^{ij\theta}(j = 0, 1, \ldots, n-1)$ form an orthogonal set of functions where each $e^{ij\theta}$ has norm $\sqrt{n}$.[1] Thus, if f is a function that is approximated by $f(\theta) = \sum_{j=0}^{n-1} c_j e^{ij\theta}$ then each $c_j = \frac{1}{n}(f(\theta), e^{ij\theta})$. The mapping $f(\theta_j) \mapsto c_j$ given by

$$c_j = \frac{1}{n}\sum_{k=0}^{n-1} f(\theta_k) e^{-2\pi ijk/n}$$

is called the ***discrete Fourier transform*** and its inverse

$$f(\theta_j) = \sum_{k=0}^{n-1} c_k e^{2\pi ijk/n}$$

the ***inverse discrete Fourier transform***. The following subroutines are available for computing these transforms.

CALL FFT$(C, n, \ell,$ IERR)
CALL FFT1$(A, B, n, \ell,$ IERR)

Let $c_j = a_j + ib_j (j = 0, 1, \ldots, n-1)$ be the data to be transformed. If FFT is called then C is a complex array containing $c_0, c_1, \ldots, c_{n-1}$ (where $C(j+1) = c_j$ for $j < n$). Otherwise, if FFT1 is called then A and B are real arrays containing $a_0, a_1, \ldots, a_{n-1}$ and $b_0, b_1, \ldots, b_{n-1}$ respectively.

The argument ℓ may have the values 1 or -1, and IERR is a variable. When FFT or FFT1 is called, if there are no input errors then IERR is set to 0 and

$$\hat{c}_j = \sum_{k=0}^{n-1} c_k e^{2\pi \ell ijk/n}$$

is computed. The results $\hat{c}_j = \hat{a}_j + i\hat{b}_j$ replace the original data $c_j = a_j + ib_j$ in C (or A and B).

Restrictions on the argument n. When FFT and FFT1 are called, n is factored by the routine into its prime factors. It is assumed that the largest prime factor of n is < 23. If $n = \mu^2 n$ where n is the square free portion of n, then it is further assumed that $n < 210$ whenever n is a product of two or more primes.

[1] Throughout this section $i = \sqrt{-1}$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1 if $n < 1$.
IERR = 2 if n has too many factors.
IERR = 3 if n has a prime factor greater than 23 or the square free portion of n is greater than 210.
IERR = 4 if $\ell \neq \pm 1$.

The setting IERR = 2 can occur only when $n > 4251528$.

Remark. The complex array C is interpreted by FFT as a real array of dimension $2n$. If this association is not permitted by the FORTRAN being employed then use FFT1.

Programming. FFT and FFT1 are interface routines for the subroutine SFFT, which was written by Richard C. Singleton (Stanford Research Institute).

Reference. Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," ***IEEE Trans. Audio and Electroacoustics***, vol. AU–17 (1969), pp. 93–103.

MULTIVARIATE FAST FOURIER TRANSFORM

Let $n_1, \ldots, n_m$ be positive integers. For any $J = (j_1, \ldots, j_m)$ where $j_\nu = 0, \ldots, n_\nu - 1$ $(\nu = 1, \ldots, m)$ let θ_J denote the point $(2\pi j_1/n_1, \ldots, 2\pi j_m/n_m)$. Also, for any complex valued functions f and g defined on the points θ_J let $(f,g) = \Sigma_J f(\theta_J)\,\overline{g(\theta_J)}$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_J. Also the functions $\phi_J(\theta) = \exp(ij_1\theta^1)\cdots\exp(ij_m\theta^m)$ form an orthogonal set where each ϕ_J has the norm $\sqrt{n_1\cdots n_m}$.[1] Thus, if f is a function that is approximated by $f = \Sigma_J c_J\phi_J$ then each $c_J = \frac{1}{n_1\cdots n_m}(f,\phi_J)$. The mapping $f(\phi_J) \mapsto c_J$ given by

$$c_j = \tfrac{1}{n_1\cdots n_m}\textstyle\sum_K f(\theta_K)\,\exp(-2\pi i j_1k_1/n_1)\cdots\exp(-2\pi i j_mk_m/n_m)$$

is called the ***discrete multivariate Fourier transform*** and its inverse

$$f(\theta_J) = \textstyle\sum_K c_K\,\exp(2\pi i j_1k_1/n_1)\cdots\exp(2\pi i j_mk_m/n_m)$$

the ***inverse discrete multivariate Fourier transform***. The sums Σ_K are for all $K = (k_1, \ldots, k_m)$ where $k_\nu = 0, 1, \ldots, n_\nu - 1$ $(\nu = 1, \ldots, m)$. The following subroutines are available for computing these transforms.

CALL MFFT$(C, N, m, \ell, \text{IERR})$
CALL MFFT1$(A, B, N, m, \ell, \text{IERR})$

Let $c_J = a_J + ib_J$ be the data to be transformed where $J = (j_1, \ldots, j_m)$ for $j_\nu = 0, 1, \ldots, n_\nu - 1$ $(\nu = 1, \ldots, m)$. If MFFT is called then C is a 1-dimensional complex array containing the values c_J where $c_J = C(1 + j_1 + j_2n_1 + j_3n_1n_2 + \cdots + j_mn_1\cdots n_{m-1})$. Otherwise, if MFFT1 is called then A and B are 1-dimensional real arrays containing the data a_J and b_J respectively.

Note. If MFFT is used and $m = 2$ or 3, then instead of having to store the m-dimensional data c_J into a 1-dimensional array C, the data may be stored in C where C is defined to be an m-dimensional array. If $m = 2$ then C may be declared to be of dimension $n_1 \times n_2$, in which case $C(j_1 + 1, j_2 + 1) = c_J$ for all $J = (j_1, j_2)$. Similarly, if $m = 3$ then C may be declared to be of dimension $n_1 \times n_2 \times n_3$, in which case $C(j_1 + 1, j_2 + 1, j_3 + 1) = c_J$ for all $J = (j_1, j_2, j_3)$. Similar comments hold for A and B if MFFT1 is employed.

N is an array containing the integers $n_1, \ldots, n_m$. The argument ℓ may have the values 1 and -1, and IERR is a variable. When MFFT or MFFT1 is called, if there are no input errors then IERR is set to 0 and the transform

$$\hat{c}_j = \textstyle\sum_K c_K\exp(2\pi\ell i j_1k_1/n_1)\cdots\exp(2\pi\ell i j_mk_m/n_m)$$

is computed. The results $\hat{c}_J = \hat{a}_J + i\hat{b}_J$ replace the original data $c_J = a_J + ib_J$ in C (or A and B).

[1] Throughout this section $i = \sqrt{-1}$ and $\theta = (\theta^1, \ldots, \theta^m)$ denotes an arbitrary point.

Restrictions on the arguments $n_1, \ldots, n_m$. When MFFT and MFFT1 are called, each n_ν is factored by the routine into its prime factors. It is assumed that the largest prime factor of n_ν is ≤ 23. If $n_\nu = \mu_\nu^2 \bar{n}_\nu$ where $\bar{n}_\nu$ is the square free portion of n_ν, then it is further assumed that $\bar{n}_\nu \leq 210$ whenever $\bar{n}_\nu$ is a product of two or more primes.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1 if some $n_\nu < 1$.
IERR = 2 if some n_ν has too many factors.
IERR = 3 if some n_ν has a prime factor greater than 23 or the square free portion of some n_ν is greater than 210.
IERR = 4 if $\ell \neq \pm 1$.
IERR = 5 if $m \leq 0$.

The setting IERR = 2 can occur only when some $n_\nu > 4251528$.

Remark. The complex array C of dimension $n_1 \cdots n_m$ is interpreted by MFFT as a real array of dimension $2n_1 \cdots n_m$. If this association is not permitted by the FORTRAN being employed then use MFFT1.

Programming. MFFT and MFFT1 are interface routines for the subroutine SFFT, which was written by Richard C. Singleton (Stanford Research Institute).

Reference. Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," ***IEEE Trans. Audio and Electroacoustics***, vol. AU-17 (1969), pp 93–103.

DISCRETE COSINE AND SINE TRANSFORMS

Let n be a positive integer and $\theta_\nu = (\nu + 1/2)\pi/n$ for $\nu = 0, 1, \ldots, n-1$. For any real valued functions f and g defined on the points θ_ν let $(f,g) = \sum_{\nu=0}^{n-1} f(\theta_\nu)\, g(\theta_\nu)$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_ν. Also $\cos j\theta$ $(j = 0, 1, \ldots, n-1)$ form an orthogonal set of functions where $\cos j\theta$ has norm $\sqrt{n}$ when $j = 0$ and norm $\sqrt{n/2}$ when $j \geq 1$. Thus, if f is a function that is approximated by $f(\theta) = a_0 + 2\sum_{j=1}^{n-1} a_j \cos j\theta$ then each $a_j = \frac{1}{n}(f(\theta), \cos j\theta)$. The mapping $f(\theta_\nu) \mapsto a_j$ is called the ***discrete cosine transform*** and its inverse $a_j \mapsto f(\theta_\nu)$ the ***inverse discrete cosine transform***.

Alternatively, the functions $\sin j\theta$ for $j = 1, \ldots, n$ also form an orthogonal set where $\sin j\theta$ has norm $\sqrt{n/2}$ when $j < n$ and norm $\sqrt{n}$ when $j = n$. Thus, if f is a function that is approximated by $f(\theta) = 2\sum_{j=1}^{n-1} b_j \sin j\theta + b_n \sin n\theta$ then each $b_j = \frac{1}{n}(f(\theta), \sin j\theta)$. The mapping $f(\theta_\nu) \mapsto b_j$ is called the ***discrete sine transform*** and its inverse $b_j \mapsto f(\theta_\nu)$ the ***inverse discrete sine transform***.

The subroutines COSQB and COSQF are available for computing the discrete cosine transform and its inverse, and the subroutines SINQB and SINQF are available for computing the discrete sine transform and its inverse. The subroutine COSQI provides information that is needed for the cosine and sine transform routines.

CALL COSQI(n, WK)

WK is an array of dimension $3n + 15$ or larger that is a work space for the routines COSQB, COSQF, SINQB, and SINQF. COSQI stores in WK information needed for the fast Fourier computation of the discrete cosine and sine transforms and their inverses. A preliminary call must be made to COSQI before COSQB, COSQF, SINQB, and SINQF can be used. After this preliminary call, COSQI need only be recalled when n is modified.

Programming. COSQI employs the subroutines RFFTI and RFFTII. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL COSQB(n, X, WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $f(\theta_0), f(\theta_1), \ldots, f(\theta_{n-1})$. When COSQB is called, $4na_j$ is computed and stored in $X(j+1)$ for $j = 0, 1, \ldots, n-1$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before COSQB can be used.

Programming. COSQB employs the subroutines COSB1, RFFTB, RFFTB1, RADB2, RADB3, RADB4, RADB5, and RADBG. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL COSQF(n, X, WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $a_0, a_1, \ldots, a_{n-1}$. When COSQF is called, $f(\theta_\nu)$ is computed and stored in $X(\nu+1)$ for $\nu = 0, 1, \ldots, n-1$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before COSQF can be used.

Example. Assume that X contains the data $f(\theta_0), \ldots, f(\theta_{n-1})$. When the statements

CALL COSQI(n,WK)
CALL COSQB(n, X,WK)
CALL COSQF(n, X,WK)

are called, COSQB stores $4na_0, \ldots, 4na_{n-1}$ in X and COSQF then sets $X(\nu+1) = 4nf(\theta_\nu)$ for $\nu = 0, 1, \ldots, n-1$. Thus, the terms of the original sequence X are multiplied by $4n$.

Programming. COSQF employs the subroutines COSQF1, RFFTF, RFFTF1, RADF2, RADF3, RADF4, RADF5, and RADFG. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL SINQB(n, X, WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $f(\theta_0), \ldots, f(\theta_{n-1})$. When SINQB is called, $4nb_j$ is computed and stored in $X(j)$ for $j = 1, \ldots, n$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before SINQB can be used.

Programming. SINQB calls the subroutine COSQB. SINQB was written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL SINQF(n, X, WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $b_1, \ldots, b_n$. When SINQF is called, $f(\theta_\nu)$ is computed and stored in $X(\nu + 1)$ for $\nu = 0, 1, \ldots, n - 1$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before SINQF can be used.

Example. Assume that X contains the data $b_1, \ldots, b_n$. When the statements

CALL COSQI(n,WK)
CALL SINQF(n, X,WK)
CALL SINQB(n, X,WK)

are called, SINQF stores $f(\theta_0), \ldots, f(\theta_{n-1})$ in X and SINQB then sets $X(j) = 4nb_j$ for $j = 1, \ldots, n$. Thus, the terms of the original sequence X are multiplied by $4n$.

Programming. SINQF calls the subroutine COSQF. The routine SINQF was written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

RATIONAL MINIMAX APPROXIMATION OF FUNCTIONS

Let $a < b$ and $g(x)$ be a continuous nonvanishing function on the interval $[a, b]$. For any continuous function $f(x)$, let $\|f\|$ denote the weighted norm $max\{|f(x)|/|g(x)| : a \leq x \leq b\}$. Also let $\phi(x)$ be a continuous strictly monotonic mapping on $[a, b]$. Then for any nonnegative integers ℓ and m, the subroutine CHEBY is available for finding a rational function

$$R(x) = \frac{p_0 + p_1\phi(x) + \cdots + p_\ell\phi(x)^\ell}{q_0 + q_1\phi(x) + \cdots + q_m\phi(x)^m}$$

which minimizes $\|R - f\|$. The subroutine performs the calculations in double precision. It is assumed that the error curve $\delta(x) = (R(x) - f(x))/g(x)$ satisfies $|\delta(x_i)| = \|R - f\|$ at precisely $\ell + m + 2$ critical points $x_0 < x_1 < \cdots < x_n (n = \ell + m + 1)$, and that $\delta(x_{i+1}) = -\delta(x_i)$ for each $i < n$.

CALL CHEBY$(a, b, F, G, \text{PHI}, \epsilon, \text{ITER}, \text{MXITER}, \ell, m, P, Q,$
$\text{ERROR}, \text{IERR}, \text{WK})$

The arguments a and b are double precision real numbers. F, G, and PHI are functions whose arguments and values are double precision real numbers. The functions must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL. The functions evaluate $f(x), g(x)$, and $\phi(x)$ respectively.

The argument ϵ is a double precision tolerance that is supplied by the user. If λ denotes the estimated value of $\|R - f\|$, then the routine converges when the error curve $\delta(x)$ satisfies $\lambda(1 - \epsilon) \leq |\delta(x_i)| \leq \lambda(1 + \epsilon)$ for each x_i. Thus ϵ specifies the relative agreement that must be attained between $\|f - R\|$ and the $|\delta(x_i)|$. Normally the setting $\epsilon = 10^{-3}$ will give satisfactory results. It is required that $0 < \epsilon < 10^{-2}$.

The Remes-type algorithm designed by Cody, Fraser, and Hart is employed. This algorithm normally requires less than 20 iterations. The argument MXITER = the maximum number of iterations that may be performed. This argument is set by the user. The related argument ITER is an integer variable that is set by the routine. When CHEBY terminates, ITER will have for its value the number of iterations that were actually performed.

P is a double precision array of dimension $\ell + 1$, Q a double precision array of dimension $m + 1$, ERROR a double precision variable, and IERR an integer variable. When CHEBY terminates, if the rational function approximation $R(x)$ has been obtained then IERR is assigned the value 0 and ERROR is the estimated error $\|R - f\|$. The coefficient p_i of the numerator of $R(x)$ is stored in $P(i + 1)$ for $i = 0, 1, \ldots, \ell$, and the coefficient q_j of the denominator is stored in $Q(j + 1)$ for $j = 0, 1, \ldots, m$. The coefficient q_0 will always have the value 1.

Let $k = \ell + m + 2$. Then WK is a double precision array of dimension $k(k + 5)$ or larger that is used for a work space.

Error Return. IERR is assigned one of the following values when the desired minimizing rational function $R(x)$ is not obtained.

IERR = 1 An input error was detected. Either $\ell < 0, m < 0, \epsilon \leq 0, \epsilon \geq 10^{-2}$, or $g(x) = 0$ for some point x.

IERR = 2 MXITER iterations were performed. More iterations are needed to obtain $R(x)$.

IERR = 3 The system of linear equations that define the coefficients p_i and q_j was found to be singular. This indicates that for the current values of ℓ and m, the numerator and denominator of $R(x)$ may have common factors.

IERR = 4 A nonmonotonic sequence of critical points x_i was obtained. Modify ℓ and/or m.

IERR = 5 The value of the error curve $\delta(x)$ at some critical point x_i appears to be too large. This indicates that $R(x)$ may have poles, and that m (or possibly a or b) may have to be modified.

IERR = 6 CHEBY completely failed to find (or roughly approximate) $R(x)$. All information in P, Q, and ERROR should be ignored.

If IERR = 2,3,4, or 5 then P and Q contain the coefficients of the most recent rational function approximation $R(x)$ obtained, and ERROR is an estimate of the error $\|R - f\|$ of the approximation.

Remark. The two most common weighting functions employed are $g(x) = 1$ and $g(x) = f(x)$. If $g(x) = 1$ then the absolute error is minimized in constructing $R(x)$. If $g(x) = f(x)$ then the relative error is minimized.

Programming. CHEBY employs the subroutines CHEBY1, CERR, and DPSLV. These routines were written by A. H. Morris. CHEBY, CHEBY1, and CERR are slightly modified translations of the ALGOL 60 procedures Chebychev, lineq, del, and surmis given in the reference.

Reference. Cody, W. J., Fraser, W., and Hart, J. F.,"Rational Chebychev Approximation using Linear Equations," ***Numerische Mathematik*** **12** (1968), pp. 242-251.

L_p APPROXIMATION OF FUNCTIONS

For any continuous real-valued function $f(x)$ defined on the interval $[a,b]$, let $||f||_p$ denote the L_p norm defined by

$$||f||_p = \left(\int_a^b |f(x)|^p\,dx\right)^{1/p} \quad \text{if} \quad 0 < p < \infty$$
$$||f||_p = \max\{|f(x)| : a \le x \le b\} \quad \text{if} \quad p = \infty.$$

If $p = \infty$ then the norm is also known as the ***Chebyshev*** norm. For any continuous function f, $0 < p \le \infty$, and $\epsilon > 0$, the subroutine ADAPT is available for finding a continuous piecewise polynomial function ϕ that satisfies $||f - \phi||_p \le \epsilon$.

CALL ADAPT(F, a, b, ϵ, k, ERROR,XKNOTS, C, IND, μ, n, ℓ, ANORM, DX,MO,m,XBREAK,KDIFF,DLEFT,DRIGHT)

It is assumed that the polynomials which form the approximation ϕ are of degree $\le n$. The argument n must satisfy $1 \le n \le 19$, and IND is a variable. When ADAPT is called, if there are no input errors and ϕ is successfully constructed, then IND is set to 0, a sequence of points $a = x_1 < \cdots < x_{k-1} < x_k = b$ is selected, and ϕ takes the form

$$\phi(x) = c_{i0} + c_{i1}(x - x_i) + \cdots + c_{in}(x - x_i)^n \quad x_i \le x \le x_{i+1}$$

for $i = 1, \ldots, k-1$. The points $x_1, \ldots, x_k$ are called the ***knots*** (or ***nodes***) of ϕ.

The argument μ is the maximum number of polynomials that may be used in forming ϕ. ERROR and k are variables, XKNOTS an array of dimension $\mu + 1$ or larger, and C a 2-dimensional array of dimension $\mu \times (n+1)$. ADAPT sets k to the number of knots that are generated. The knots $x_1, \ldots, x_k$ are stored in the XKNOTS array, and the coefficients $c_{i0}, \ldots, c_{in}$ are stored in $C(i,1), \ldots, C(i,n+1)$ for $i = 1, \ldots, k-1$. ERROR is a rough estimate of the error $||f - \phi||_p$.

The argument ℓ specifies the degree of smoothness that the approximation ϕ must satisfy. It is assumed that $0 \le \ell < 10$ and $n > 2\ell$. If $\ell = 0$ then it is only required that ϕ be continuous on the interval $[a,b]$. Otherwise, if $\ell \ge 1$ then it is assumed that f is of class C^ℓ on $[a,b]$ except at possibly a finite number of points (called ***break points***), and it is required that ϕ be of class C^ℓ on $[a,b]$ except possibly at the break points.

The argument m specifies the number of break points of f. It is assumed that $m \le 20$. If $m = 0$ then the arguments XBREAK,KDIFF,DLEFT, and DRIGHT can be ignored. Otherwise, if $m \ge 1$ then it is assumed that XBREAK,KDIFF,DLEFT, and DRIGHT are arrays of dimension m or larger, and that

XBREAK(i) — the i^{th} break point, call it u_i,
KDIFF(i) — the smallest integer ν_i for which the ν_i^{th} derivative of f does not exist or is not continuous at u_i,
DLEFT(i) — the value from the left of the ν_i^{th} derivative at u_i, and
DRIGHT(i) — the value from the right of the ν_i^{th} derivative at u_i

for $i = 1, \ldots, m$. It is also assumed that $a < u_1 < \cdots < u_m < b$ and $n > 2\nu_i$ for each ν_i.

F is the name of a user defined function that has the value $F(x, D) = f(x)$ for $a \le x \le b$. If $\ell = 0$ then D can be ignored. (However, D must still be given as an argument of F.) Otherwise, if $\ell \ge 1$ then D is an array of dimension greater than or equal to ℓ. For any x not a break point in XBREAK, the user must set $D(j) =$ the j^{th} derivative of f at x for $j \le \ell$. However, if $x =$ XBREAK(i) then the user need only set $D(j) =$ the j^{th} derivative of f at x for $j <$ KDIFF(i). The function F must be declared in the calling program to be of type EXTERNAL.

The argument DX specifies the maximum distance to be permitted between the knots x_i, and the argument ANORM specifies the norm to be used. Set

ANORM $= \pm 1.0$ for L_1 approximation.
ANORM $= \pm 2.0$ for L_2 (least squares) approximation.
ANORM $= 3.0$ for L_∞ (minimax) approximation.
ANORM $= -p$ for L_p $(0 < p < \infty)$ approximation.

Before considering the argument MO, one should be briefly acquainted with how ADAPT operates. ADAPT employs the following procedure to construct ϕ.

(1) Set $I = [a, b]$ and $k = 1$. Let a be the first knot of ϕ.
(2) If the interior of I contains no break points then go to (3). Otherwise, if $I = [c, d]$ then partition I into the subintervals $[c, u]$ and $[u, d]$ where u is the smallest break point greater than c. Stack the right subinterval $[u, d]$ and reset I to $[c, u]$.
(3) Construct a polynomial ϕ_I on I using Hermite interpolation. If the length of the interval I is $\le$ DX and ϕ_I satisfactorily approximates f on I, then go to (4). Otherwise go to (5).
(4) Set k to be $k + 1$. Let ϕ_I be the $(k-1)^{st}$ polynomial forming ϕ and let the right end point of I be the k^{th} knot of ϕ. If the interval stack is empty then the procedure is finished. Otherwise, obtain from the stack the next interval I to be considered and return to (2).
(5) The polynomial ϕ_I cannot be used. Partition I into halves, stack the right subinterval and reset I to be the left subinterval. Then go to (3).

The argument MO specifies the accuracy criterion that the approximation ϕ is to satisfy on a subinterval $I = [c, d]$ of $[a, b]$. It is assumed that MO $= 0, 1, 2$. If the L_∞ norm is used then MO is ignored[1] and ϕ is required to satisfy $|f(x) - \phi(x)| \le \epsilon$ for $c \le x \le d$. Otherwise, if the L_p $(0 < p < \infty)$ norm is used then ϕ is required to satisfy:

$$\int_c^d |f(x) - \phi(x)|^p \, dx \le \frac{d-c}{b-a} \epsilon^p \qquad \text{for MO} = 0$$

$$\int_c^d |f(x) - \phi(x)|^p \, dx \le \epsilon^p \qquad \text{for MO} = 2$$

The setting MO $= 0$, which is the most commonly used setting, requires the total error $\|f - \phi\|_p \le \epsilon$. The alternate setting MO $= 2$ employs ϵ to control local accuracy. If ϕ consists of $k - 1$ polynomials then the total error $\|f - \phi\|_p = (k-1)^{1/p}\epsilon$. This setting can

[1] However, it is still required that MO $= 0, 1, 2$

be useful when f is rough. A (heuristic) compromise strategy is provided when MO = 1. At each step in the formation of ϕ, the MO = 1 strategy estimates the total number of subintervals that will finally be needed and adjusts the error requirement for the subinterval I accordingly. This strategy attempts to keep the total error to a minimum while relaxing the local accuracy criterion demanded by the MO = 0 setting.

Remarks. IND, k, μ, n, ℓ, MO, m, and KDIFF are integer arguments. All other arguments (including F) are double precision arguments.

Error Return. ADAPT assigns IND one of the following values:

IND = 0 The approximation was successfully constructed.

IND = −1 Either $a \geq b$ or one of the arguments ϵ, n, ℓ, ANORM, MO, m is assigned an incorrect value.

IND = −2 $[a, b]$ is too small an interval.

IND = −3 DX is less than $(b - a)/\mu$. Since only μ subintervals can be used and each subinterval must be of length $\leq$ DX, the interval $[a, b]$ cannot be covered. Make DX or μ larger.

IND = −4 The restriction $a \leq u_1 < \cdots < u_m \leq b$ on the break points is violated.

IND = −5 Either KDIFF(i) < 0 or KDIFF(i) > $(n - 1)/2$ for some i.

IND = 1 ADAPT selected $\mu + 1$ knots. More knots are needed to complete the problem.

IND = 2 A subinterval $I = [c, d]$ must be partitioned into subintervals $[c, u]$ and $[u, d]$ where u is a break point. However, this cannot be done either because the interval stack is full, or partitioning will produce too small an interval. (The stack can hold only 50 subintervals).

IND = 3 A subinterval must be partitioned because its length is greater than DX. However, this cannot be done since the interval stack is full.

IND = 4 A subinterval must be partitioned so that the accuracy criterion can be satisfied. However, this cannot be done either because the stack is full, or partitioning will produce too small an interval.

If an input error is detected (i.e., if IND < 0) then no computation is performed. Otherwise, if IND $\geq$ 0 then when ADAPT terminates k = the number of knots generated, XKNOTS contains the knots, C contains the coefficients of the polynomials generated, and ERROR contains the error estimate for $f - \phi$ over the interval covered.

Remarks.

(1) If the L_∞ norm is used then ϵ controls absolute accuracy, not relative acuracy. This should be kept in mind when ϵ is to be set for any L_p norm.

(2) ADAPT requires more time when $\ell \geq 2$ than when $\ell = 0$ or 1. However, the choice of the norm normally has little effect on the efficiency of the routine.

(3) ADAPT can yield excellent results even when the derivatives of f have singularities. The one major exception is when the first derivative of f is not bounded. Then the routine can be expected to fail.

Example. The following code can be used for approximating $f(x) = e^x$ on the interval $[0, 1]$.

```
      DOUBLE PRECISION F, A, B, EPS, ERROR, ANORM, DX
      DOUBLE PRECISION XKNOTS (11), C(10,20)
      INTEGER KDIFF(1)
      DOUBLE PRECISION XBREAK(1),DLEFT(1),DRIGHT(1)
      EXTERNAL F
      DATA MAX, A, B, DX/10, 0.D0, 1.D0, 1.D0/
      N = 8
      L = 1
      EPS = 1.D-12
      ANORM = 3.D0
      CALL ADAPT(F,A,B,EPS,K,ERROR,XKNOTS,C,IND,
     *   MAX,N,L,ANORM,DX,0,0,XBREAK,KDIFF,DLEFT,DRIGHT)
```

Here F may be defined by:

```
      DOUBLE PRECISION FUNCTION F(X,D)
      DOUBLE PRECISION X,D(1)
      F = DEXP(X)
      D(1) = F
      RETURN
      END
```

In the ADAPT statement XBREAK, KDIFF, DLEFT, and DRIGHT are ignored since $m = 0$.

Programming. ADAPT employs the subroutines ADAPT1, ADSET, ADTAKE, ADCOMP, NEWTON, ADCHK, ADPUT, ADTRAN and functions ERRINT, POLYDD. These routines exchange information in labeled common blocks. The block names are INPUTZ, RESULZ, KONTRL, and COMDIF. The routines were written by John R. Rice (Purdue University) and modified by A. H. Morris. The function DPMPAR is also used.

References.

(1) Rice, J. R.,"Algorithm 525. ADAPT, Adaptive Smooth Curve Fitting," ***ACM Trans. Math Software*** 4 (1978), pp. 82–94.

(2) _____, "Adaptive Approximation," ***J. Approx. Theory*** **16** (1976), pp. 329–337.

CALCULATION OF THE TAYLOR SERIES OF A COMPLEX ANALYTIC FUNCTION

Let $f(z) = \sum_{n\geq 0} a_n(z - z_0)^n$ denote the Taylor series of an analytic function f around a point z_0. Then the subroutines CPSC and DCPSC are available for obtaining the coefficients a_n of the series. CPSC obtains single precision results and DCPSC obtains double precision results.

CALL CPSC(f, z_0, n,IND,ϵ, R, A,ERR)

It is assumed that $f(z)$ is a user defined function whose arguments and values are complex numbers. The argument f must be declared in the calling program to be of types COMPLEX and EXTERNAL.

The argument z_0 is complex, n is an integer where $1 \leq n \leq 51$, and A is a complex array of dimension n or larger. IND may be any integer. If IND $= 0$ then a_j is computed and stored in $A(j + 1)$ for $j = 0, 1, \ldots, n - 1$. Otherwise, if IND $\neq 0$ then $f(z_0)$ and the derivatives $f'(z_0), \ldots, f^{(n-1)}(z_0)$ are computed and stored in A.

The argument ϵ specifies the relative accuracy of f. If it is estimated that f produces results accurate to k significant decimal digits then one may set $\epsilon = 10^{-k}$. It is assumed that $\epsilon \geq 0$. If $\epsilon = 0$ then the results of f are assumed to be correct to machine precision.

When CPSC is called, $f(z)$ is evaluated on circles of various radii around the point z_0. R is a real variable. On input, R is the radius of the first circle on which $f(z)$ is to be evaluated. After using this radius, the radius is repeatedly modified (first by factors of 2 or 1/2) until a suitable final radius r_c is obtained for deriving the values of the coefficients a_j. This radius, whose value depends on ϵ, is called the computational radius of the series $\Sigma a_j(z - z_0)^j$. When the routine terminates, R is assigned the value r_c.

ERR is a real array of dimension n or larger. On output, ERR(j) is the estimated absolute error of $A(j)$ for $j = 1, \ldots, n$.

Usage. Given a radius R, $f(z)$ is evaluated on k equidistant points on the circle of radius R around z_0 where

$k = 8$ when $1 \leq n \leq 6$,
$k = 16$ when $7 \leq n \leq 12$,
$k = 32$ when $13 \leq n \leq 25$,
$k = 64$ when $26 \leq n \leq 51$.

It is assumed that $f(z)$ has at least one nonzero coefficient a_j among the first $k/2$ coefficients, and at least one nonzero coefficient among the next $k/2$ coefficients. Thus, the routine should not be used to obtain coefficients of a low degree polynominal such as $f(z) = 1 - z^2$. In such cases, the results will normally be incorrect.

In general, the selection of the radius R of the first circle on which $f(z)$ is evaluated is not bothersome. A randomly selected value of moderate size, such as $R = 6.2738$, almost

always suffices. No difficulties normally arise when the initial radius R is greater than the radius of convergence of the series $\Sigma a_j(z - z_0)^j$. However, difficulties do arise when

(1) the routine attempts to evaluate f too close to (or exactly at) a singularity,
(2) f has a Taylor series expansion which contains one or more extremely large isolated terms (e.g., $f(z) = 10^8 + \sin z$),
(3) f has a branch point near z_0, or
(4) The initial radius R is too far from the computational radius r_c (see the error return section below).

The risk of (1) occurring is minimized by the random selection of an initial radius R. For (2) and (3), a severe loss of accuracy can occur when a large number of coefficients are requested. In these cases, any loss of accuracy is reported by the ERR array.

Error return. $A(j)$ is assigned the value 0 and ERR(j) is set to 10^{10} for $j = 1, \ldots, n$ when the initial radius R differs from r_c by a factor of 30000 or greater.

Programming. CPSC was written by Bengt Fornberg (California Institute of Technology) and modified by A.H. Morris. CPSC employs the function SPMPAR.

References.

(1) Fornberg, B.,"Numerical Differentiation of Analytic Functions," ***ACM Trans. Math Software*** **7**, 1981, pp. 512–526.
(2) _____,"Algorithm 579. CPSC: Complex Power Series Coefficients," ***ACM Trans. Math Software*** **7**, 1981, pp. 542-547.

CALL DCPSC(F, x_0, y_0, n,IND,ϵ, R,AR,AI,ERR)

F is the name of a user defined subroutine that has the format:
CALL $F(x, y, u, v)$
This subroutine is used for evaluating $f(z)$ at point z. The arguments x and y are the real and imaginary parts of z, and u and v are the real and imaginary parts of $f(z)$. The arguments x and y have double precision values, and u and v are double precision variables. F must be declared in the calling program to be of type EXTERNAL.

The arguments x_0 and y_0, which have double precision values, are the real and imaginary parts of z_0. The argument n is an integer where $1 \leq n \leq 51$, and AR and AI are double precision arrays of dimension n or larger. IND may be any integer. If $IND = 0$ then the real and imaginary parts of a_j are stored in AR($j + 1$) and AI($j + 1$) for $j = 0, 1, \ldots, n - 1$. Otherwise, if IND $\neq$ 0 then the real and imaginary parts of $f(z_0)$ and the derivatives $f'(z_0), \ldots, f^{(n-1)}(z_0)$ are stored in AR and AI respectively.

The argument ϵ, which has double precision values, specifies the relative accuracy of the subroutine F. If it is estimated that F produces results accurate to k decimal digits then one may set $\epsilon = 10^{-k}$. It is assumed that $\epsilon > 0$. If $\epsilon = 0$ then the results of F are assumed to be correct to machine precision.

When DCPSC is called, $f(z)$ is evaluated on circles of various radii around the point z_0. R is a double precision variable. On input, R is the radius of the first circle on which

$f(z)$ is to be evaluated. After using this radius, the radius is repeatedly modified (first by factors of 2 or 1/2) until a suitable final radius r_c is obtained for deriving the values of the coefficients a_j. This radius, whose value depends on ϵ, is called the computational radius of the series $\Sigma a_j(z - z_0)^j$. When the routine terminates R is assigned the value r_c.

ERR is a double precision array of dimension n or larger. On output, ERR(j) is the estimated absolute error of the complex value stored in AR(j) and AI(j) for $j = 1, \ldots, n$.

Usage. DCPSC is used in the same manner as CPSC. See the usage section for CPSC.

Error return. AR(j) and AI(j) are assigned the value 0 and ERR(j) is set to 10^{10} for $j = 1, \ldots, n$ when the initial radius R differs from r_c by a factor of 30000 or greater.

Programming. DCPSC is an adaptation by A.H. Morris of the subroutine CPSC, written by Bengt Fornberg (California Institute of Technology). DCPSC employs the function DPMPAR.

References.

(1) Fornberg, B.,"Numerical Differentiation of Analytic Functions," ***ACM Trans. Math Softwcre*** **7**, 1981, pp. 512–526.

(2) _____, "Algorithm 579. CPSC: Complex Power Series Coefficients," ***ACM Trans. Math Software*** **7**, 1981, pp. 542–547.

LINEAR INTERPOLATION

Let a be a real number and $(x_1, y_1), \ldots, (x_n, y_n)$ a sequence of points. The following function performs a linear interpolation at point a.

TRP(a, n, X, Y)

It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$ respectively. TRP$(a, n, X, Y) = b$ where b is the value of the interpolation at a.

Programmer. A. H. Morris.

LAGRANGE INTERPOLATION

Let $\{(x_i, y_i) : i = 1, \ldots, n\}$ be a set of $n \geq 2$ points where $x_1 < \cdots < x_n$, m be an integer where $2 \leq m \leq n$, and $\bar{x}_1, \ldots, \bar{x}_k$ be $k \geq 1$ points at which m point Lagrange interpolation is to be performed. The subroutine LTRP is available for performing this interpolation.

CALL LTRP$(m, X, Y, n, \text{XI}, \text{YI}, k, T, \text{IERR})$

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, XI an array containing $\bar{x}_1, \ldots, \bar{x}_k$, and YI an array of dimension k or larger. When LTRP is called, if no input errors are detected then interpolation is performed at each $\bar{x}_j$ and the result stored in YI(j) for $j = 1, \ldots, k$.

T is an array of dimension m or larger. The array is used as a temporary storage area by the routine.

Error Return. IERR is a variable that is set by the routine. If no input errors are detected then IERR is assigned the value 0. Otherwise, IERR is assigned one of the following values:

IERR = 1 if $m < 2$.
IERR = 2 if $m > n$.
IERR = 3 if $k < 1$.

When an error is detected LTRP immediately terminates.

Algorithm. If $\bar{x}_j = (x_i + x_{i+m})/2$ for some i, then $(x_i, y_i), \ldots, (x_{i+m-1}, y_{i+m-1})$ are the m data points used in the Lagrange interpolation at $\bar{x}_j$. Otherwise, the data points selected for the interpolation are those m points (x_i, y_i) whose abscissas are closest to $\bar{x}_j$.

Linear Interpolation. For $m = 2$, if the abscissae x_i are not equally spaced then LTRP can produce different results than the linear interpolating function TRP. If $\bar{x}_j$ lies in the interval $[x_i, x_{i+1})$ then TRP always uses the data points (x_i, y_i) and (x_{i+1}, y_{i+1}) to find the interpolated value at $\bar{x}_j$. However, the operation of LTRP is somewhat different. For example, if the point $\bar{x}_j$ in $[x_i, x_{i+1})$ is closer to x_{i-1} than to x_{i+1}, then (x_{i-1}, y_{i-1}) and (x_i, y_i) will be the data points employed in the interpolation. Thus, TRP will normally be the procedure that will be desired for linear interpolation.

Programming. Developed by A. H. Morris. The portion of the code for finding the subinterval containing $\bar{x}_j$ was written by Rondall E. Jones (Sandia Laboratories).

HERMITE INTERPOLATION

Let $x_1, \ldots, x_k$ be $k \geq 1$ distinct points. For each x_i assume that we are given $n_i \geq 1$ values $y_i, y_i', \ldots, y_i^{(n_i-1)}$. If $n = n_1 + \cdots + n_k$, then there exists a unique polynomial of degree $n-1$ which satisfies

$$\begin{aligned} p(x_i) &= y_i \\ p'(x_i) &= y_i' \\ &\vdots \\ p^{n_i-1}(x_i) &= y_i^{(n_i-1)} \end{aligned}$$

for each $i = 1, \ldots, k$. The subroutine HTRP is available for obtaining this polynomial.

CALL HTRP$(n, X, Y, A,$ WK,IERR)

X and Y are arrays of dimension n containing the following information: $X(j) = x_1$ for $j = 1, \ldots, n_1$ and $Y(1), \ldots, Y(n_1)$ contain the values $y_1, y_1', \ldots, y_1^{(n_1-1)}$. For $i = 2, \ldots, k$ let $m_i = n_1 + \cdots + n_{i-1}$. Then $X(m_i + j) = x_i$ for $j = 1, \ldots, n_i$ and $Y(m_i + 1), \ldots, Y(m_i + n_i)$ contain the values $y_i, y_i', \ldots, y_i^{(n_i-1)}$.

A is an array of dimension n and IERR an integer variable. When HTRP is called, if no errors are detected then IERR is assigned the value 0 and the coefficients a_j of the polynomial $p(x) = a_0 + \sum_{j=1}^{n-1} a_j(x - X(1)) \cdots (x - X(j))$ are computed and stored in $A(j+1)$ for $j = 0, 1, \ldots, n-1$.

WK is an array of dimension n or larger that is a work space for the routine.

Error Return. If an error is detected then IERR is assigned one of the following values:

IERR = 1 The argument n is not positive.

IERR = 2 There exists integers i and ℓ for which $X(i) = X(\ell)$ but $X(i) \neq X(j)$ for some j where $i < j < \ell$. In this case, the values i and ℓ are stored (in floating point form) in WK(1) and WK(2).

When an error is detected, the routine immediately terminates.

Example. If $p(0) = 2$, $p(-1) = 1$, and $p'(-1) = 2$ where $x_1 = 0$ and $x_2 = -1$, then HTRP stores $2, 1, -1$ in A. Hence, $p(x) = 2 + x - x(x+1)$ is the desired polynomial.

Remark. The Newton representation $a_0 + \sum_{j=1}^{n-1} a_j(x - X(1)) \cdots (x - X(j))$ of the polynomial $p(x)$ can be converted to the Taylor series representation $\sum_{j=0}^{n-1} c_j(x - \alpha)^j$ by the subroutine PCOEFF.

Programmer. A. H. Morris.

CONVERSION OF REAL POLYNOMIALS FROM NEWTON TO TAYLOR SERIES FORM

For $n \geq 1$ let $p(x) = a_0 + \sum_{j=1}^{n-1} a_j(x - x_1)\cdots(x - x_j)$. Then for any real number α, the subroutine PCOEFF is available for converting the polynomial $p(x)$ to the Taylor series form $\sum_{j=0}^{n-1} c_j(x - \alpha)^j$.

CALL PCOEFF(α, n, X, A, C, T)

X is a single precision real array containing $x_1, \ldots, x_{n-1}$, A a single precision real array containing $a_0, a_1, \ldots, a_{n-1}$ where a_j is stored in $A(j+1)$ for $j = 0, 1, \ldots, n-1$, and C a single precision real array of dimension n or larger. When PCOEFF is called then the coefficients c_j of the Taylor series representation are computed and stored in $C(j+1)$ for $j = 0, 1, \ldots, n-1$.

T is a ***double precision*** array of dimension n or larger. The array is a work space for the routine. (The conversion of the coefficients is done in double precision.)

Note. A and C may reference the same storage area, in which case the results c_j will overwrite the input data a_j.

Programmer. A. H. Morris.

LEAST SQUARES POLYNOMIAL FIT

Let $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of $m \geq 2$ points where $x_i \neq x_j$ for $i \neq j$. Then for any positive integer n where $n < m$, the subroutine PFIT is available for obtaining the (unique) n^{th} degree polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$ which minimizes $\sum_{i=1}^{m} (p(x_i) - y_i)^2$.

CALL PFIT$(n, m, X, Y, A,$ RNORM,PHI,WK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, and A an array of dimension $n+1$ or larger. RNORM and IERR are variables. When PFIT is called, if no input errors are detected then IERR is set to 0, the coefficients a_j of $p(x)$ are stored in $A(j+1)$ for $j = 0, 1, \ldots, n$, and RNORM is assigned the value $\sqrt{\Sigma_i (p(x_i) - y_i)^2}$.

PHI is an array of dimension $2(n+1)$ or larger, and WK is an array of dimension $4m$ or larger. PHI and WK are work spaces for the routine.

Error Return. IERR $= 1$ if $n < 1$ or $n \geq m$.

Algorithm. The abscissas x_i are first mapped into values in the interval $[-1, 1]$. Then the Forsythe procedure is used.

Programmer. A. H. Morris.

WEIGHTED LEAST SQUARES POLYNOMIAL FIT

Let $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of $m \geq 2$ points where $x_i \neq x_j$ for $i \neq j$, and let $w_i \geq 0 (i = 1, \ldots, m)$ be weights. It is assumed that $m_w \geq 2$ where m_w is the number of nonzero weights. For any positive integer n where $n < m_w$, the subroutine WPFIT is available for finding the (unique) n^{th} degree polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$ which minimizes $\sum_{i=1}^{m} w_i (p(x_i) - y_i)^2$.

CALL WPFIT$(n, m, X, Y, W, A,$ RNORM,PHI,WK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, W an array containing $w_1, \ldots, w_m$, and A an array of dimension $n + 1$ or larger. RNORM and IERR are variables. When WPFIT is called, if no input errors are detected then IERR is set to 0, the coefficients a_j of $p(x)$ are computed and stored in $A(j + 1)$ for $j = 0, 1, \ldots, n$, and RNORM is assigned the value $\sqrt{\Sigma_i w_i (p(x_i) - y_i)^2}$.

PHI is an array of dimension $2(n + 1)$ or larger, and WK is an array of dimension $4m$ or larger. PHI and WK are work spaces for the routine.

Error Return. IERR = 1 if $n < 1$ or $n \geq m_w$, and IERR = 3 if some w_i is negative.

Algorithm. The abscissas x_i corresponding to the positive weights are first mapped into values in the interval $[-1, 1]$. Then the Forsythe procedure is used.

Programmer. A. H. Morris.

CUBIC SPLINE INTERPOLATION

Given $x_1 < \cdots < x_n$. A function $f(x)$ is a ***cubic spline having the nodes(knots)*** $x_1, \ldots, x_n$ if f is a polynomial of degree ≤ 3 on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3.$$

Consequently, the function f is obtained by fitting the polynomials $f_1, \ldots, f_{n-1}$ together at the points $x_2, \ldots, x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1, \ldots, n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1, \ldots, n$.

Assume now that the ordinates $y_1, \ldots, y_n$ are given. Then there exist an infinitude of splines with nodes $x_1, \ldots, x_n$ that interpolate the points (x_i, y_i). In general, two independent conditions must be imposed to uniquely specify the interpolating spline f which is of interest. Frequently, f is restricted on the first interval $[x_1, x_2]$ by requiring that $f'(x_1)$ or $f''(x_1)$ has a given value, or that f is continuous at x_2. Also, f is restricted on the last interval $[x_{n-1}, x_n]$ by requiring that $f'(x_n)$ or $f''(x_n)$ has a given value, or that f is continuous at x_{n-1}. The subroutine CBSPL is available for obtaining the spline when these restrictions are employed. Alternatively, f may be required to satisfy the conditions

$$f''(x_1) = \alpha f''(x_2) + \beta \qquad |\alpha| < 1$$
$$f''(x_n) \quad \alpha f''(x_{n-1}) + \beta \qquad |\alpha| < 1$$

when $n \geq 4$. Then the subroutine SPLIFT is available for obtaining the spline.

CALL CBSPL$(X, Y, A, B, C, n, i_1, i_2, w_1, w_2, \mathrm{IERR})$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $x_1 < \cdots < x_n$ and $n \geq 3$. A, B and C are arrays of dimension n or larger, and IERR an integer variable. When CBSPL is called , if no input errors are detected then IERR is set to 0. Also, the coefficients a_i, b_i, c_i $(i = 1, \ldots, n-1)$ of the interpolating spline $f(x)$ are stored in A, B, C, and $A(n)$ is set to $f'(x_n)$.

The arguments i_1, i_2, w_1, w_2 specify the conditions that the spline $f(x)$ must satisfy. It is assumed that i_1 and i_2 have the values 0,1,2 where:

i_1	0	f is continuous at x_2.	i_2	0	f is continuous at x_{n-1}.
i_1	1	$f'(x_1)$ has the value w_1.	i_2	1	$f'(x_n)$ has the value w_n.
i_1	2	$f''(x_1)$ has the value w_1.	i_2	2	$f''(x_n)$ has the value w_n.

If $i_1 = 0$ then the argument w_1 is ignored, and if $i_2 = 0$ then w_2 is ignored.

Error Return. IERR = 1 if $n < 3$ and IERR = 2 if $x_i \geq x_{i+1}$ for some i.

Remarks.

( ) $B(n)$ and $C(n)$ are used for temporary storage.

(2) If $i_1 = i_2 = 0$ and $n = 3$, then it is also assumed that $f'(x_2) + f'(x_3) = 2(\frac{y_3 - y_2}{x_3 - x_2})$.

(3) After A, B and C have been obtained, then SCOMP or SCOMP1 may be used to evaluate the spline at any point x. SEVAL or SEVAL1 may be used if derivatives are also desired.

Programming. CBSPL is an adaptation by A. H. Morris of the subroutine CUBSPL, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, *A Practical Guide to Splines*, Springer-Verlag, 1978.

CALL SPLIFT(X, Y, DY, DDY, n, W, IERR, MO, $\alpha, \beta, \bar{\alpha}, \bar{\beta}$)

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $x_1 < \cdots < x_n$ and $n \geq 4$. DY and DDY are arrays of dimension n or larger, and IERR an integer variable. When SPLIFT is called, if there are no input errors then IERR is assigned the value 0, the first derivatives $f'(x_1), \ldots, f'(x_n)$ are computed and stored in DY(1), ..., DY(n), and the second derivatives $f''(x_1), \ldots, f''(x_n)$ are computed and stored in DDY(1), ..., DDY(n).

W is an array of dimension $3n$ or larger that is used for a storage area. On the first call to SPLIFT the argument MO must be set to 0. When SPLIFT is initially called, certain calculations which depend only on the values of $\alpha, \bar{\alpha}$, and $x_1, \ldots, x_n$ are performed and the results stored in W. On subsequent calls to SPLIFT, if only values of $\beta, \bar{\beta}$ or $y_1, \ldots, y_n$ are modified, then the information in W need not be recomputed. Set MO = 1 and the information in W will be reused.

Error Return If there is an input error then IERR is set as follows:

IERR = 1 if $|\alpha| \geq 1$ or $|\bar{\alpha}| \geq 1$.
IERR = 2 if $n < 4$.
IERR = 3 if the restriction $x_1 < \cdots < x_n$ is not satisfied.

Remarks.

!) After DY and DDY have been obtained, then SCOMP1 or SCOMP2 may be used to evaluate the spline at any point x. SEVAL1 or SEVAL2 may be used if derivatives are also desired.

(2) Given the values y'_1 and y'_n. Then there exists a unique interpolating cubic spline f that satisfies $f'(x_1) = y'_1$ and $f'(x_n) = y'_n$. This spline can be obtained by setting $\alpha = \bar{\alpha} = -1/2$ and

$$\beta = \frac{3}{x_2 - x_1}\left[\frac{y_2 - y_1}{x_2 - x_1} - y'_1\right] \qquad \bar{\beta} = \frac{-3}{x_n - x_{n-1}}\left[\frac{y_n - y_{n-1}}{x_n - x_{n-1}} - y'_n\right].$$

Programmer. Rondall E. Jones (Sandia Laboratories).

WEIGHTED LEAST SQUARES CUBIC SPLINE FITTING

Let $t_1 < \cdots < t_\ell$ be a sequence where $\ell \geq 2$, $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of $m \geq 4$ points where $t_1 \leq x_1 < \cdots < x_m \leq t_\ell$, and $w_1, \ldots, w_m$ be positive weights. Then the subroutine SPFIT is available for obtaining a cubic spline $f(x)$ with the nodes $t_1, \ldots, t_\ell$ which minimizes $\sum_{i=1}^{m} w_i \left(f(x_i) - y_i\right)^2$. This spline is represented by

$$f(x) = z_j + a_j(x - t_j) + b_j(x - t_j)^2 + c_j(x - t_j)^3$$

for $t_j \leq x < t_{j+1}$ $(j = 1, \ldots, \ell - 1)$. If the nodes are selected so that $\ell \leq m - 2$ and each interval (t_j, t_{j+1}) contains a data point x_{ν_j}, then this least squares approximation is unique.

CALL SPFIT$(X, Y, W, m, T, \ell, Z, A, B, C,$ WK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, W an array containing $w_1, \ldots, w_m$, and T an array containing $t_1, \ldots, t_\ell$. Z, A, B, C are arrays of dimension $\ell - 1$ or larger, and IERR is an integer variable. When SPFIT is called, if no input errors are detected then IERR is set to 0. Also, the coefficients z_j, a_j, b_j, c_j of the least squares approximating spline $f(x)$ are computed and stored in Z, A, B, C.

WK is an array of dimension $7\ell + 18$ or larger that is a work space for the routine.

Error Return. IERR is set to one of the following values when an input error is detected.

IERR = 1 if $\ell < 2$.
IERR = 2 if $t_1 < \cdots < t_\ell$ is not satisfied.
IERR = 3 if $m \geq 4$ and $t_1 \leq x_1 \leq \cdots \leq x_m \leq t_\ell$ are not satisfied.

If an error is detected, the routine immediately terminates.

Remarks.

(1) Occasionally, accuracy can be lost when $t_\ell = x_m$. Therefore, it is recommended that $t_\ell > x_m$.
(2) After A, B, C, and Z have been obtained, then SCOMP may be used to evaluate the spline at any point x. SEVAL may be used if derivatives are also desired.

Programming. SPFIT employs the subroutines BSPP, BSLSQ, BSPVB, BCHFAC, and BCHSLV. SPFIT was written by A. H. Morris.

LEAST SQUARES CUBIC SPLINE FITTING WITH EQUALITY AND INEQUALITY CONSTRAINTS

Let $t_1 < \cdots < t_\ell$ be a sequence where $\ell \geq 2$, and $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of points where $t_1 \leq x_1 < \cdots < x_m \leq t_\ell$. Then the subroutine CSPFIT is available for obtaining a cubic spline $f(x)$ with the nodes $t_1, \ldots, t_\ell$ which minimizes $\sum_i (f(x_i) - y_i)^2$ subject to a set of constraints. This spline is represented by

$$f(x) = a_j + b_j(x - t_j) + c_j(x - t_j)^2 + d_j(x - t_j)^3$$

for $t_j \leq x < t_{j+1}$ $(j = 1, \ldots, \ell - 1)$.

Notation. For any real x, $f^{(j)}(x)$ will denote the value $f(x)$ if $j = 0$, the j^{th} derivative of f at x if $j = 1$ or 2, and the right third derivative $f'''(x+)$ if $j = 3$.

CALL CSPFIT$(X, Y, m, T, \ell,$XCON$,R,$NDER$,k, Z, A, B, C,$WK$,$IWK$,$IND$)$

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, and T an array containing $t_1, \ldots, t_\ell$. The argument k is the number of constraints to be satisfied $(k \geq 0)$ and XCON, R, NDER are arrays of dimension $\max\{1, k\}$ or larger.

If $k \geq 1$ then XCON contains the points $\bar{x}_1, \ldots, \bar{x}_k$ at which the constraints are placed. The points may be given in any order and need not be distinct. It is assumed that $t_1 \leq \bar{x}_i \leq t_\ell$ for $i = 1, \ldots, k$. At $\bar{x}_i$ any of the following types of constraints

itype = 0	$f^{(j)}(\bar{x}_i) \leq r_i$
itype = 1	$f^{(j)}(\bar{x}_i) \geq r_i$
itype = 2	$f^{(j)}(\bar{x}_i) = r_i$
itype = 3	$f^{(j)}(\bar{x}_i) = f^{(j)}(r_i)$

can be imposed where $j = 0, 1, 2$, or 3. For the selected constraint, it is assumed that $R(i) = r_i$ and NDER$(i) =$ *itype* $+ 4j$. If an *itype* 3 constraint is imposed then it is further assumed that $t_1 \leq r_i \leq t_\ell$.

WK and IWK are arrays that are work spaces for the routine. On input, IWK(1) is the dimension of WK and IWK(2) the dimension of IWK. If $k = 0$ then one may set IWK(1) = NB and IWK(2) = 2 where NB $= 7(\ell + 7)$. Otherwise, let $\nu = \ell + 3$, m_e be the number of equality (i.e., *itype* 2 and 3) constraints, and m_i the number of inequality (*itype* 0 and 1) constraints. Then one must set

$$\text{IWK}(1) \geq \text{NB} + (\nu + k)\nu + 2(m_e + \nu) + (m_i + \nu) + (m_i + 2)(\nu + 6)$$
$$\text{IWK}(2) \geq m_i + 2\nu.$$

Remarks.

(1) If $k \geq 1$ then one can let the routine determine the amount of storage that is needed for WK and IWK. Set IWK(1) = NB and IWK(2) = 2, in which case errors will occur (see the IND = −6, −7 error values below).

(2) NDER(i) may be assigned a negative value $(i = 1, \ldots, k)$. If NDER$(i) < 0$ then the i^{th} constraint is ignored.

Z, A, B, C are arrays of dimension $\ell - 1$ or larger. When CSPFIT is called, if the cubic spline $f(x)$ is obtained then the coefficients z_j, a_j, b_j, c_j of the spline are stored in Z, A, B, C.

IND is a variable that reports the status of the results. When CSPFIT terminates, IND is assigned one of the following values:

IND = 0 The spline was obtained.
IND = 1 The equality (i.e., *itype* 2 and 3) constraints are contradictory. The spline was obtained where the equality constraints were satisfied in a least squares sense.
IND = 2 The spline could not be obtained. The constraints are contradictory.
IND = -1 $\ell < 2$.
IND = -2 $m \leq 0$ or $k < 0$.
IND = -3 $t_i \geq t_{i+1}$ for some i.
IND = -4 The assumption $t_1 \leq x_1 < \cdots < x_m \leq t_\ell$ is not satisfied.
IND = -5 The i^{th} constraint is incorrect for the value of i stored in IWK(2).
IND = -6 Insufficient storage was specified for the WK array. IWK(1) has been reset to the amount of storage needed by WK.
IND = -7 Insufficient storage was specified for the IWK array. IWK(2) has been reset to the amount of storage needed by IWK.

IWK(1) and IWK(2) are not modified when IND $\neq -5, -6, -7$.

Example. If one wants f to be convex on an interval $[t_i, t_{i+1}]$, then this is equivalent to requiring that $f''(x) \geq 0$ on the interval. Thus, since f is a cubic polynomial on the interval, it suffices to require that $f''(t_i) \geq 0$ and $f''(t_{i+1}) \geq 0$. For these two constraints, 9 is the appropriate value for the NDER array.

Programming. CSPFIT employs the subroutines BFIT and BSPP, and the functions and subroutines used by BFIT. CSPFIT was written by A. H. Morris.

CUBIC SPLINE EVALUATION

Given $x_1 < \cdots < x_n$. A function $f(x)$ is a ***cubic spline having the nodes (knots)*** $x_1, \ldots, x_n$ if f is a polynomial of degree ≤ 3 on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, the function f is obtained by fitting the polynomials $f_1, \ldots, f_{n-1}$ together at the points $x_2, \ldots, x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1, \ldots, n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1, \ldots, n$.

A cubic spline f given by the polynomials $f_1, \ldots, f_{n-1}$ is uniquely defined by any of the following three sets of data:

(1) the points (x_i, y_i) and coefficients a_i, b_i, c_i for $i = 1, \ldots, n-1$
(2) the points (x_i, y_i) and first derivatives $f'(x_i)$ for $i = 1, \ldots, n$
(3) the points (x_i, y_i) and second derivatives $f''(x_i)$ for $i = 1, \ldots, n$

The subroutines SCOMP, SCOMP1, SCOMP2 are available for computing the spline at any point x. SCOMP is used if data set (1) is given, SCOMP1 is used if data set (2) is given, and SCOMP2 is used if data set (3) is given.

CALL SCOMP$(X, Y, A, B, C, N, \mathrm{XI}, \mathrm{YI}, m, \mathrm{IERR})$

Let $N = n - 1$. Then N is the number of polynomials f_i that form the spline, X and Y are arrays containing the abscissas $x_1, \ldots, x_N$ and ordinates $y_1, \ldots, y_N$, and A, B, C are arrays containing the coefficients a_i, b_i, c_i $(i = 1, \ldots, N)$. It is assumed that $N \geq 1$ and that $x_1 < \cdots < x_N$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Note. SCOMP does not require f to be a spline. It is only required that $f_i(x)$ be a cubic polynomial $y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$ and that

$$\begin{aligned} f(x) &= f_1(x) \quad \text{for } x < x_1 \\ f(x) &= f_i(x) \quad \text{for } x_i \leq x < x_{i+1} \quad (1 \leq i < N) \\ f(x) &= f_N(x) \quad \text{for } x \geq x_N. \end{aligned}$$

In this case, SCOMP computes the value $f(x_j +)$ for $j = 1, \ldots, m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SCOMP1(X, Y, DY, n, XI, YI, m, IERR)

X, Y, DY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$, and first derivatives $f'(x_1), \ldots, f'(x_n)$ respectively. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP1 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SCOMP2(X, Y, DDY, n, XI, YI, m, IERR)

X, Y, DDY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$ and second derivatives $f''(x_1), \ldots, f''(x_n)$ respectively. It is assumed that $x_1 < \cdots < x_n$ for $n \geq 2$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP2 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Programmer. Rondall E. Jones (Sandia Laboratories).

CUBIC SPLINE EVALUATION AND DIFFERENTIATION

Given $x_1 < \cdots < x_n$. A function $f(x)$ is a ***cubic spline having the nodes (knots)*** $x_1, \ldots, x_n$ if f is a polynomial of degree ≤ 3 on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, the function f is obtained by fitting the polynomials $f_1, \ldots, f_{n-1}$ together at the points $x_2, \ldots, x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1, \ldots, n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1, \ldots, n$.

A cubic spline f given by the polynomials $f_1, \ldots, f_{n-1}$ is uniquely defined by any of the following three sets of data:

(1) the points (x_i, y_i) and coefficients a_i, b_i, c_i for $i = 1, \ldots, n-1$
(2) the points (x_i, y_i) and first derivatives $f'(x_i)$ for $i = 1, \ldots, n$
(3) the points (x_i, y_i) and second derivatives $f''(x_i)$ for $i = 1, \ldots, n$

The subroutines SEVAL, SEVAL1, SEVAL2 are available for computing the spline and its first and second derivatives at any point x. SEVAL is used if data set (1) is given, SEVAL1 is used if data set (2) is given, and SEVAL2 is used if data set (3) is given.

CALL SEVAL$(X, Y, A, B, C, N,$ XI,YI,DYI,DDYI$, m,$ IERR$)$

Let $N = n - 1$. Then N is the number of polynomials f_i that form the spline, X and Y are arrays containing the abscissas $x_1, \ldots, x_N$ and ordinates $y_1, \ldots, y_N$, and A, B, C are arrays containing the coefficients $a_i, b_i, c_i (i = 1, \ldots, N)$. It is assumed that $N \geq 1$ and that $x_1 < \cdots < x_N$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger, and IERR is a variable. When SEVAL is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and the values $f(\bar{x}_j), f'(\bar{x}_j), f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \ldots, m$.

Note. SEVAL does not require f to be a spline. It is only required that $f_i(x)$ be a cubic polynomial $y_i + a_i(x - x_i) + b_1(x - x_i)^2 + c_i(x - x_i)^3$ and that

$$\begin{aligned} f(x) &= f_1(x) \quad \text{for } x < x_1 \\ f(x) &= f_i(x) \quad \text{for } x_i \leq x < x_{i+1} \quad (1 \leq i < N) \\ f(x) &= f_N(x) \quad \text{for } x \geq x_N. \end{aligned}$$

In this case, SEVAL computes the values $f(\bar{x}_j+), f'(\bar{x}_j+), f''(\bar{x}_j+)$ for $j = 1, \ldots, m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SEVAL1(X, Y, DY, n, XI,YI,DYI,DDYI, m, IERR)

X, Y, DY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$, and first derivatives $f'(x_1), \ldots, f'(x_n)$ respectively. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger,and IERR is a variable. When SEVAL1 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and the values $f(\bar{x}_j), f'(\bar{x}_j), f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \ldots, m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SEVAL2(X, Y, DDY, n, XI,YI,DYI,DDYI, m, IERR)

X, Y, DDY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$, and second derivatives $f''(x_1), \ldots, f''(x_n)$ respectively. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$,YI, DYI, DDYI are arrays of dimension m or larger,and IERR is a variable. When SEVAL2 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geq 1$ then IERR is set to 0 and the values $f(\bar{x}_j), f'(\bar{x}_j), f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \ldots, m$.

Programmer. Rondall E. Jones (Sandia Laboratories).

INTEGRALS OF CUBIC SPLINES

Given $x_1 < \cdots < x_n$. A function $f(x)$ is a ***cubic spline having the nodes (knots)*** $x_1, \ldots, x_n$ if f is a polynomial of degree ≤ 3 on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, the function f is obtained by fitting the polynomials $f_1, \ldots, f_{n-1}$ together at the points $x_2, \ldots, x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1, \ldots, n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1, \ldots, n$.

A cubic spline f given by the polynomials $f_1, \ldots, f_{n-1}$ is uniquely defined by any of the following three sets of data:

(1) the points (x_i, y_i) and coefficients a_i, b_i, c_i for $i = 1, \ldots, n-1$
(2) the points (x_i, y_i) and first derivatives $f'(x_i)$ for $i = 1, \ldots, n$
(3) the points (x_i, y_i) and second derivatives $f''(x_i)$ for $i = 1, \ldots, n$

For any real α and β the functions CSINT, CSINT1, CSINT2 are available for computing the integral $\int_\alpha^\beta f(t)\,dt$. CSINT is used if data set (1) is given, CSINT1 is used if data set (2) is given, and CSINT2 is used if data set (3) is given.

CSINT$(X, Y, A, B, C, N, \alpha, \beta)$

Let $N = n - 1$. Then N is the number of polynomials f_i that form the spline, X and Y are arrays containing the abscissas $x_1, \ldots, x_N$ and ordinates $y_1, \ldots, y_N$, and A, B, C are arrays containing the coefficients $a_i, b_i, c_i (i = 1, \ldots, N)$. It is assumed that $N \geq 1$ and that $x_1 < \cdots < x_N$. Then CSINT has the value $\int_\alpha^\beta f(t)\,dt$.

Programming. CSINT calls the function INTRVL. CSINT was written by A. H. Morris.

CSINT1$(X, Y, \mathrm{DY}, n, \alpha, \beta)$

X, Y, DY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$, and first derivatives $f'(x_1), \ldots, f'(x_n)$ respectively. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. Then CSINT1$(X, Y, \mathrm{DY}, n, \alpha, \beta) = \int_\alpha^\beta f(t)\,dt$.

Programming. CSINT1 calls the function INTRVL. CSINT1 was written by A. H. Morris.

CSINT2$(X, Y, \mathrm{DDY}, n, \alpha, \beta)$

X, Y, DDY are arrays containing the abscissas $x_1, \ldots, x_n$, ordinates $y_1, \ldots, y_n$, and second derivatives $f''(x_1), \ldots, f''(x_n)$ respectively. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. Then CSINT2$(X, Y, \mathrm{DDY}, n, \alpha, \beta) = \int_\alpha^\beta f(t)\,dt$.

Programming. CSINT2 calls the function INTRVL. CSINT2 was written by A. H. Morris.

PERIODIC CUBIC SPLINE INTERPOLATION

Given $x_1 < \cdots < x_n$. A function $f(x)$ is a ***cubic spline having the nodes (knots)*** $x_1, \ldots, x_n$ if f is a polynomial of degree ≤ 3 on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3.$$

Consequently, the function f is obtained by fitting the polynomials $f_1, \ldots, f_{n-1}$ together at the points $x_2, \ldots, x_{n-1}$.

Normally, a cubic spline f is defined for $x \notin [x_1, x_n]$ by letting $f(x) = f_1(x)$ for $x < x_1$ and $f(x) = f_{n-1}(x)$ for $x > x_n$. However, if

$$\begin{aligned} f(x_1) &= f(x_n) \\ f'(x_1+) &= f'(x_n-) \\ f''(x_1+) &= f''(x_n-) \end{aligned}$$

are satisfied then f may be defined for $x \notin [x_1, x_n]$ by requiring that $f(x + p) = f(x)$ for all x where $p = x_n - x_1$. The function f is called a ***periodic spline*** when this is done.

Now if the values $y_1, \ldots, y_{n-1}$ are given, then there exists a unique periodic cubic spline f having the nodes $x_1, \ldots, x_n$ and satisfying $f(x_i) = y_i$ for $i = 1, \ldots, n-1$. The following subroutine is available for obtaining the coefficients a_i, b_i, c_i of this spline.

CALL PDSPL$(X, Y, A, B, C, n$,WK,IERR)

X is an array containing $x_1, \ldots, x_n$ and Y an array containing $y_1, \ldots, y_{n-1}$. It is assumed that $x_1 < \cdots < x_n$ and $n \geq 3$. A, B, C are arrays of dimension $n - 1$ or larger, and IERR a variable. When PDSPL is called, if no input errors are detected then IERR is set to 0 and the coefficients a_i, b_i, c_i of the interpolating periodic cubic spline are stored in A, B, C.

WK is an array of dimension $n - 2$ or larger that is a work space for the routine.

Error Return. IERR = 1 if $n < 3$ and IERR = 2 if $x_i \geq x_{i+1}$ for some i.

Remark. After A, B, and C have been obtained, then PSCMP may be used to evaluate the periodic spline at any point x. PSEVL may be used if derivatives are also desired.

Programmer. A. H. Morris.

LEAST SQUARES PERIODIC CUBIC SPLINE FITTING

Let $t_1 < \cdots < t_\ell$ be a sequence where $\ell \geq 2$ and $\{ (x_i, y_i) : i = 1, \ldots, m \}$ a set of points where $t_1 \leq x_1 < \cdots < x_m \leq t_\ell$. Then the following subroutine is available for obtaining a periodic cubic spline $f(x)$ with the nodes $t_1, \ldots, t_\ell$ which minimizes $\sum_i (f(x_i) - y_i)^2$. This spline is represented by

$$f(x) = z_j + a_j(x - t_j) + b_j(x - t_j)^2 + c_j(x - t_j)^3$$

for $t_j \leq x < t_{j+1}$ $(j = 1, \ldots, \ell - 1)$.

CALL PDFIT$(X, Y, m, T, \ell, Z, A, B, C$,WK,IWK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, and T an array containing $t_1, \ldots, t_\ell$. Z, A, B, C are arrays of dimension $\ell - 1$ or larger. When PDFIT is called, if the desired periodic cubic spline is obtained then the coefficients z_j, a_j, b_j, c_j of the spline are stored in Z, A, B, C.

WK and IWK are arrays that are work spaces for the routine. On input IWK(1) is the dimension of WK and IWK(2) the dimension of IWK. It is required that IWK(1) $\geq (\ell + 6)(\ell + 15) + 10$ and IWK(2) $\geq 2\ell + 6$.

IERR is a variable that reports the status of the results. When PDFIT terminates, IERR has one of the following values:

IERR = 0 The desired spline was obtained.
IERR = 1 The spline could not be obtained.
IERR = 2 Either $m \leq 0$ or $\ell < 2$.
IERR = 3 $t_j > t_{j+1}$ for some j.
IERR = 4 The assumption $t_1 \leq x_1 < \cdots < x_m \leq t_\ell$ is not satisfied.
IERR = 5 Insufficient storage was specified for the WK array. IWK(1) has been reset to the amount of storage needed by WK.
IERR = 6 Insufficient storage was specified for the IWK array. IWK(2) has been reset to the amount of storage needed by IWK.

If an error is detected, then IERR ≥ 2 and the routine terminates.

Remark. After A, B, C, and Z have been obtained, then PSCMP may be used to evaluate the periodic spline at any point x. PSEVL may be used if derivatives are also desired.

Programming. PDFIT employs the subroutines BSPP, BFIT, BFIT0, BNDACC, BNDSL, BSPVB, BSPVD, LSEI, and the subroutines and functions used by LSEI. PDFIT was written by A. H. Morris.

PERIODIC CUBIC SPLINE EVALUATION AND DIFFERENTIATION

Let $x_1 < \cdots < x_n$ be a sequence where $n \geq 2$, and f a periodic cubic spline where

$$f(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

for $x_i \leq x < x_{i+1}$ $(i = 1, \ldots, n-1)$. Then the subroutine PSCMP is available for evaluating the spline for all points x, and the subroutine PSEVL is available for computing its derivatives.

CALL PSCMP$(X, Y, A, B, C, n$,XI,YI,m,IERR)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_{n-1}$, and A, B, C arrays containing the coefficients a_i, b_i, c_i $(i = 1, \ldots, n-1)$. It is assumed that $x_1 < \cdots < x_n$ and $n \geq 3$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the periodic spline is to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When PSCMP is called, if no input errors are detected then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Error Return. IERR = 1 if $n < 3$ or $m \leq 0$.

Programmer. A. H. Morris.

CALL PSEVL$(X, Y, A, B, C, n$,XI,YI,DYI,DDYI,m,IERR)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_{n-1}$, and A, B, C arrays containing the coefficients a_i, b_i, c_i $(i = 1, \ldots, n-1)$. It is assumed that $x_1 < \cdots < x_n$ and $n \geq 3$.

Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which the periodic spline and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger, and IERR a variable. When PSEVL is called, if no input errors are detected then IERR is set to 0 and $f(\bar{x}_j)$, $f'(\bar{x}_j)$, $f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \ldots, m$.

Error Return. IERR = 1 if $n < 3$ or $m \leq 0$.

Programmer. A. H. Morris.

N-DIMENSIONAL CUBIC SPLINE CLOSED CURVE FITTING

Given $m \geq 2$ points $x_i = (x_{i1}, \ldots, x_{in})$ where $n \geq 2$. One procedure for fitting a closed curve to the points is to first let $s_1 = 0, s_i = s_{i-1} + ||x_i - x_{i-1}||$ for $i = 2, \ldots, m$, and $s_{m+1} = s_m + ||x_1 - x_m||$,[1] next let $t_i = s_i/s_{m+1}$ for $i = 1, \ldots, m+1$, and then to find for each $j \leq n$ the periodic cubic spline $\gamma_j(t)$ having the knots $t_1, \ldots, t_{m+1}$ where $\gamma_j(t_i) = x_{ij}$ for $i \leq m$. The mapping $\gamma(t) = (\gamma_1(t), \ldots, \gamma_n(t))$ then defines on the interval $[0, 1]$ a closed curve which traverses the points $(t_1, x_1), \ldots, (t_m, x_m), (1, x_1)$ and satisfies $\gamma'(0) = \gamma'(1)$ and $\gamma''(0) = \gamma''(1)$. For $t < 0$ and $t > 1$, $\gamma(t)$ is defined by periodicity (having period 1). The subroutine CSLOOP is available for obtaining the derivatives $\gamma_j'(t_i)$ which characterize this curve, the subroutine LOPCMP is available for computing the curve, and the subroutine LOPDF is available for differentiating the curve.

CALL CSLOOP$(m, n, X, kx, T, \mathrm{DX}, kdx, \mathrm{WK}, \mathrm{IERR})$

X is an $m \times n$ matrix whose i^{th} row contains the point $x_i = (x_{i1}, \ldots, x_{in})$, where $m \geq 2$ and $n \geq 2$. It is assumed that the points $x_1, \ldots, x_m$ are indexed in the order that they are to be traversed by the curve $\gamma(t)$. It is also assumed that $x_i \neq x_{i+1}$ for $i = 1, \ldots, m-1$ and that $x_m \neq x_1$.

DX is a 2–dimensional array containing at least m rows and n columns. The arguments kx and kdx have the following values:

kx = the number of rows in the dimension statement for X in the calling program

kdx = the number of rows in the dimension statement for DX in the calling program

It is required that $kx \geq m$ and $kdx \geq m$.

IERR is a variable and T an array of dimension m or larger. When CSLOOP is called, if no input errors are detected then IERR is assigned the value 0 and $t_1, \ldots, t_m$ are computed and stored in T. Also, the derivatives $\gamma_j'(t_i)$ are computed and stored in DX, where the i^{th} row of DX contains $\gamma'(t_i) = (\gamma_1'(t_i), \ldots, \gamma_n'(t_i))$.

WK is an array of dimension $4(m-1)$ or larger that is a work space for the routine.

Error Return. IERR reports the following input errors:

IERR = 1 if $m < 2$ or $n < 2$

IERR = 2 if $x_i = x_{i+1}$ for some i.

IERR = 3 if $x_1 = x_m$.

Remark. After T and DX are obtained, LOPCMP may be used to compute the curve and LOPDF may be used to differentiate the curve.

Programming. CSLOOP calls the subroutine CSLOP1 and function SNRM2. CSLOOP and CSLOP1 were written by A. H. Morris.

[1] $||x|| = \sqrt{\Sigma_i x_i^2}$ for any $x = (x_1, \ldots, x_m)$.

CALL LOPCMP$(m, n, T, X, kx, \mathrm{DX}, kdx, \ell, \mathrm{TI}, Z, kz)$

T is an array containing the knots $t_1, \ldots, t_m$, X an $m \times n$ matrix whose i^{th} row contains the point x_i, and DX an $m \times n$ matrix whose i^{th} row contains the derivative $\gamma'(t_i)$. The arguments kx and kdx have the following values:

kx = the number of rows in the dimension statement for X in the calling program

kdx = the number of rows in the dimension statement for DX in the calling program

It is assumed that $m \geq 2$, $n \geq 2$, $kx \geq m$, and $kdx \geq m$.

Let $\bar{t}_1, \ldots, \bar{t}_\ell$ be the points at which the curve γ is to be evaluated. TI is an array containing $\bar{t}_1, \ldots, \bar{t}_\ell$, and Z a 2–dimensional array containing at least ℓ rows and n columns. The argument kz is the row dimension for Z in the calling program. It is assumed that $\ell \geq 1$ and $kz \geq \ell$. When LOPCMP is called, $\gamma(\bar{t}_i) = (\gamma_1(\bar{t}_i), \ldots, \gamma_n(\bar{t}_i))$ is computed and stored in the i^{th} row of Z for $i = 1, \ldots, \ell$.

Programmer. A. H. Morris.

CALL LOPDF$(m, n, T, X, kx, \mathrm{DX}, kdx, t_0, Z, \mathrm{DZ}, \mathrm{DDZ})$

T is an array containing the knots $t_1, \ldots, t_m$, X an $m \times n$ matrix whose i^{th} row contains the point x_i, DX an $m \times n$ martrix whose i^{th} row contains the derivative $\gamma'(t_i)$. The arguments kx and kdx have the following values:

kx = the number of rows in the dimension statement for X in the calling program

kdx = the number of rows in the dimension statement for DX in the calling program

It is assumed that $m \geq 2$, $n \geq 2$, $kx \geq m$, and $kdx \geq m$.

Z, DZ, and DDZ are arrays of dimension n. For any real t_0, $\gamma(t_0) = (\gamma_1(t_0), \ldots, \gamma_n(t_0))$, $\gamma'(t_0) = (\gamma_1'(t_0), \ldots, \gamma_n'(t_0))$, and $\gamma''(t_0) = (\gamma_1''(t_0), \ldots, \gamma_n''(t_0))$ are computed and stored in Z, DZ, and DDZ respectively.

Programmer. A. H. Morris.

SPLINE UNDER TENSION INTERPOLATION

Given real σ and $x_1 < \cdots < x_n$. A function $f(x)$ is a ***spline having the tension factor*** σ ***and the nodes (knots)*** $x_1, \ldots, x_n$ if $f(x)$ and its first two derivatives are continuous on $[x_1, x_n]$, and $f''(x) - \bar{\sigma}^2 f(x) = a_i x + b_i$ on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$. Here $\bar{\sigma} = |\sigma|(n-1)/(x_n - x_1)$ and a_i, b_i are constants. For $x_i \le x \le x_{i+1}$ $f(x)$ can be represented by

$$f(x) = A_i \sinh \bar{\sigma}(x - x_i) + B_i \sinh \bar{\sigma}(x_{i+1} - x) - (a_i x + b_i)/\bar{\sigma}^2$$

when $\sigma \neq 0$, and by a cubic polynomial when $\sigma = 0$.

Assume now that n ordinates $y_1, \ldots, y_n$ are given. Then there exist an infinitude of splines $f(x)$ having tension σ for which $f(x_i) = y_i$ $(i = 1, \ldots, n)$. However, if values y_1' and y_n' are given then only one of these splines will satisfy $f'(x_1) = y_1'$ and $f'(x_n) = y_n'$. For convenience, denote this spline by f_σ. If $\sigma = 0$ then it is clear that f_σ is the standard cubic spline. Also it can be verified that when $\sigma \to \infty$, f_σ converges uniformly on $[x_1, x_n]$ to the piecewise linear function $\ell(x)$ where $\ell(x) = y_i + m_i(x - x_i)$ for $x_i \le x \le x_{i+1}$ $(i = 1, \ldots, n-1)$. Here $m_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$. The following subroutine is available for obtaining the spline f_σ.

CALL CURV1$(n, X, Y,$ SLP1,SLPN,IND,DDY,TEMP,σ, IERR)

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $n \ge 2$ and $x_1 < \cdots < x_n$.

SLP1 and SLPN are assigned the values y_1' and y_n'. The user may omit values for either or both of these arguments. IND specifies the information that is provided.

IND = 0 Values are supplied for SLP1 and SLPN.
IND = 1 A value is supplied for SLP1 but not for SLPN.
IND = 2 A value is supplied for SLPN but not for SLP1.
IND = 3 Values are not supplied for SLP1 and SLPN.

If a value is not supplied by the user, then the routine provides a value.

DDY is an array of dimension n or larger, and IERR is an integer variable. When CURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the second derivatives $f_\sigma''(x_1), \ldots, f_\sigma''(x_n)$ are computed and stored in DDY.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$.
IERR = 2 if $x_1 < \cdots < x_n$ is not satisfied.

When either of these errors is detected, the routine immediately terminates.

Remarks.
(1) After DDY is obtained then CURV2 may be used to evaluate the spline at any point x.
(2) X, Y, n, SLP1, SLPN, IND, σ are not modified by CURV1.

Programming. CURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. CURV1, CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

Reference. Cline, A. K.,"Scalar and Planar Valued Curve Fitting using Splines under Tension," ***Comm. ACM*** **17** (1974), pp. 218–220.

SPLINE UNDER TENSION EVALUATION

Given σ and $x_1 < \cdots < x_n$. A function $f(x)$ is a ***spline having the tension factor σ and the nodes (knots)*** $x_1, \ldots, x_n$ if $f(x)$ and its first two derivatives are continuous on $[x_1, x_n]$, and $f''(x) - \bar{\sigma}^2 f(x) = a_i x + b_i$ on the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$. Here $\bar{\sigma} = |\sigma|(n-1)/(x_n - x_1)$ and a_i, b_i are constants. If $f(x) = f_i(x)$ for $x_i \leq x \leq x_{i+1}$ then $f_i(x)$ can be represented by

$$f_i(x) = A_i \sinh \bar{\sigma}(x - x_i) + B_i \sinh \bar{\sigma}(x_{i+1} - x) - (a_i x + b_i)/\bar{\sigma}^2$$

when $\sigma \neq 0$, and by a cubic polynomial when $\sigma = 0$. For $x < x_1$ we let $f(x) = f_1(x)$, and for $x > x_n$ we let $f(x) = f_{n-1}(x)$.

Assume now that $f(x_i) = y_i$ for $i = 1, \ldots, n$. Then for a fixed σ, $f(x)$ is uniquely defined by the points (x_i, y_i) and the second derivatives $f''(x_i)(i = 1, \ldots, n)$. When this data is available, the following function may be used to compute the spline at any point t.

CURV2$(t, n, X, Y, \mathrm{DDY}, \sigma)$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$, and DDY is an array containing the second derivatives $f''(x_1), \ldots, f''(x_n)$. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. CURV2$(t, n, X, Y$ DDY$, \sigma) = f(t)$ for any real t.

Remark. After DDY has been obtained, CURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified then the derivative information in DDY will have to be recomputed before CURV2 can be used with the new tension factor.

Programming. CURV2 employs the function INTRVL and subroutine SNHCSH. CURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

Reference. Cline, A. K., "Scalar and Planar Valued Curve Fitting using Splines under Tension," *Comm. ACM* **17** (1974), pp. 218-220.

DIFFERENTIATION AND INTEGRATION OF SPLINES UNDER TENSION

Let $f(x)$ be a spline having the tension factor σ and the nodes $x_1, \ldots, x_n$. Assume that $f(x_i) = y_i$ for $i = 1, \ldots, n$. If the second derivatives $f''(x_1), \ldots, f''(x_n)$ are known then the following functions may be used for differentiating and integrating the spline.

CURVD$(t, n, X, Y, \mathrm{DDY}, \sigma)$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$, and DDY is an array containing the second derivatives $f''(x_1), \ldots, f''(x_n)$. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. For any real t, the derivative $f'(t)$ is computed and assigned to be the value of CURVD$(t, n, X, Y, \mathrm{DDY}, \sigma)$.

Programming. CURVD employs the function INTRVL and subroutine SNHCSH. CURVD was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

CURVI$(a, b, n, X, Y, \mathrm{DDY}, \sigma)$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$, and DDY is an array containing the second derivatives $f''(x_1), \ldots, f''(x_n)$. It is assumed that $n \geq 2$ and $x_1 < \cdots < x_n$. CURVI$(a, b, n, X, Y, \mathrm{DDY}, \sigma) = \int_a^b f(t)dt$ for any real a and b.

Note. It is not required that $a \leq b$.

Programming. CURVI employs the function INTRVL and subroutine SNHCSH. CURVI was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

TWO DIMENSIONAL SPLINE UNDER TENSION CURVE FITTING

Given a sequence of points $(x_1, y_1), \ldots, (x_n, y_n)$. One procedure for fitting a curve to the points is to let $s_1 = 0$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for $i = 2, \ldots, n$, and then to find two splines $x(s)$ and $y(s)$ with tension σ that satisfy $x(s_i) = x_i$ and $y(s_i) = y_i$ for $i = 1, \ldots, n$. If θ_1 and θ_n are the desired angles for the curve $s \mapsto (x(s), y(s))$ at the points (x_1, y_1) and (x_n, y_n), then the splines $x(s)$ and $y(s)$ can be selected so that $x'(s_i) = \cos\theta_i$ and $y'(s_i) = \sin\theta_i$ for $i = 1, n$. The curve $s \mapsto (x(s), y(s))$ then passes through the points (x_i, y_i) and has the required slopes at the end points. The subroutine KURV1 is available for obtaining the second derivatives $x''(s_i), y''(s_i) (i = 1, \ldots, n)$ which characterize this curve, and the subroutine KURV2 is available for computing the curve.

CALL KURV1$(n, X, Y,$ SLP1,SLPN,IND,DDX,DDY,TEMP$, S, \sigma,$ IERR$)$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $n \geq 2$ and that the points (x_i, y_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ for $i = 1, \ldots, n-1$.

SLP1 and SLPN are assigned the values θ_1 and θ_n. These angles are measured counterclockwise (in radians) from the positive x-axis. The user may omit values for SLP1 and/or SLPN. IND specifies the information that is provided.

IND = 0 Values are supplied for SLP1 and SLPN.
IND = 1 A value is supplied for SLP1 but not for SLPN.
IND = 2 A value is supplied for SLPN but not for SLP1.
IND = 3 Values are not supplied for SLP1 and SLPN.

If a value is not supplied by the user, then the routine provides a value.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s)$ and $y(s)$ approximate cubic splines. Otherwise, if $|\sigma|$ is large, say $|\sigma| > 100$, then the resulting curve approximates the polygonal line from (x_1, y_1) to (x_n, y_n).

IERR is an integer variable and S, DDX, DDY are arrays of dimension n or larger. When KURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1, \ldots, s_n$ are computed and stored in S. Also, the second derivatives $x''(s_1), \ldots, x''(s_n)$ and $y''(s_1), \ldots, y''(s_n)$ are computed and stored in DDX and DDY.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$.
IERR = 2 if $(x_i, y_i) = (x_{i+1}, y_{i+1})$ for some i.

When either of these errors is detected, the routine immediately terminates.

Remark. After S, DDX, DDY are obtained, KURV2 may be used to compute the curve.

Programming. KURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. KURV1,

CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL KURV2(t, XT, YT, n, X, Y, DDX, DDY, S, σ)

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$, S is an array containing $s_1, \ldots, s_n$, and DDX and DDY are arrays containing the second derivatives $x''(s_1), \ldots, x''(s_n)$ and $y''(s_1), \ldots, y''(s_n)$.

Now consider the change of variables $t = s/s_n$, and let $t \mapsto (\bar{x}(t), \bar{y}(t))$ denote the curve in terms of the new parameter t. XT and YT are real variables. For any $0 \le t \le 1$, KURV2 computes the point $(\bar{x}(t), \bar{y}(t))$ on the curve and assigns XT the value $\bar{x}(t)$ and YT the value $\bar{y}(t)$.

Remark. After DDX and DDY have been obtained, KURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified then the derivative information in DDX and DDY will have to be recomputed before KURV2 can be used with the new tension factor.

Programming. KURV2 employs the function INTRVL and subroutine SNHCSH. KURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

TWO DIMENSIONAL SPLINE UNDER TENSION CLOSED CURVE FITTING

Given $n \geq 2$ points $(x_1, y_1), \ldots, (x_n, y_n)$. One procedure for fitting a closed curve to the points is to let $s_1 = \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2}$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for $i = 2, \ldots, n$, and then to find periodic splines $x(s)$ and $y(s)$ with tension σ that pass through the points $(s_1, x_1), \ldots, (s_n, x_n), (s_1 + s_n, x_1)$ and $(s_1, y_1), \ldots, (s_n, y_n), (s_1 + s_n, y_1)$. The mapping $s \mapsto (x(s), y(s))$ then defines a closed curve that passes through the points (x_i, y_i). The subroutine KURVP1 is available for obtaining the second derivatives $x''(s_i), y''(s_i)(i = 1, \ldots, n)$ which characterize this curve, and the subroutine KURVP2 is available for computing the curve.

CALL KURVP1$(n, X, Y, \text{DDX}, \text{DDY}, \text{TEMP}, S, \sigma, \text{IERR})$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $n \geq 2$ and that the points (x_i, y_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ for $i = 1, \ldots, n-1$.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s)$ and $y(s)$ approximate periodic cubic splines. Otherwise, if $|\sigma|$ is large, say $|\sigma| > 100$, then the curve approximates the closed polygonal path that traverses the points (x_i, y_i).

IERR is an integer variable and S, DDX, DDY are arrays of dimension n or larger. When KURVP1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1, \ldots, s_n$ are computed and stored in S. Also, the second derivatives $x''(s_1), \ldots, x''(s_n)$ and $y''(s_1), \ldots, y''(s_n)$ are computed and stored in DDX and DDY.

TEMP is an array of dimension $2n$ or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$.
IERR = 2 if $(x_i, y_i) = (x_{i+1}, y_{i+1})$ for some i.

When either of these errors is detected, the routine immediately terminates.

Remark. After S, DDX, DDY are obtained, KURVP2 may be used to compute the curve.

Programming. KURVP1 employs the subroutines TERMS and SNHCSH. KURVP1 and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL KURVP2$(t, \text{XT}, \text{YT}, n, X, Y, \text{DDX}, \text{DDY}, S, \sigma)$

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$, S is an array containing $s_1, \ldots, s_n$, and DDX and DDY are arrays containing the second derivatives $x''(s_1), \ldots, x''(s_n)$ and $y''(s_1), \ldots, y''(s_n)$.

Now consider the change of variables $t = (s - s_1)/s_n$, and let $t \mapsto (\bar{x}(t), \bar{y}(t))$ denote the curve in terms of the new parameter t. Then $t \mapsto (\bar{x}(t), \bar{y}(t))$ maps 0 and 1 onto the point (x_1, y_1), and $t \mapsto (\bar{x}(t), \bar{y}(t))$ is a periodic function (with period 1).

XT and YT are real variables. For any real t, KURVP2 computes the point $(\bar{x}(t), \bar{y}(t))$ on the curve and assigns XT the value $\bar{x}(t)$ and YT the value $\bar{y}(t)$.

Remark. After DDX and DDY have been obtained, KURVP2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified then the derivative information in DDX and DDY will have to be recomputed before KURVP2 can be used with the new tension factor.

Programming. KURVP2 employs the function INTRVL and subroutine SNHCSH. KURVP2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

THREE DIMENSIONAL SPLINE UNDER TENSION CURVE FITTING

Given $n \geq 2$ points $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$. One procedure for fitting a curve to the points is to let $s_1 = 0$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2}$ for $i = 2, \ldots, n$, and then to find splines $x(s), y(s), z(s)$ with tension σ that satisfy $x(s_i) = x_i, y(s_i) = y_i$, and $z(s_i) = z_i$ for $i = 1, \ldots, n$. If (x'_1, y'_1, z'_1) and (x'_n, y'_n, z'_n), are the desired slopes for the curve $s \mapsto (x(s), y(s), z(s))$ at the points (x_1, y_1, z_1) and (x_n, y_n, z_n), then the splines $x(s), y(s), z(s)$ can be selected so that $x'(s_i) = x'_i$, $y'(s_i) = y'_i$, and $z'(s_i) = z'_i$ for $i = 1, n$. The curve $s \mapsto (x(s), y(s), z(s))$ then passes through the points (x_i, y_i, z_i) and has the required slopes at the end points. The subroutine QURV1 is available for obtaining the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1, \ldots, n)$ which characterize this curve, and the subroutine QURV2 is available for computing the curve.

CALL QURV1$(n, X, Y, Z,$SLP1X,SLP1Y,SLP1Z,SLPNX,SLPNY,
SLPNZ,IND,DDX,DDY,DDZ,TEMP$, S, \sigma,$IERR$)$

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and Z an array containing $z_1, \ldots, z_n$. It is assumed that $n \geq 2$ and that the points (x_i, y_i, z_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i, z_i) \neq (x_{i+1}, y_{i+1}, z_{i+1})$ for $i = 1, \ldots, n-1$.

SLP1X, SLP1Y, SLP1Z and SLPNX, SLPNY, SLPNZ are assigned the values x'_1, y'_1, z'_1 and x'_n, y'_n, z'_n. The user may omit values for SLP1X, SLP1Y, SLP1Z and/or SLPNX, SLPNY, SLPNZ. The argument IND specifies the information that is provided.

- IND = 0 Values are supplied for SLP1X,SLP1Y,SLP1Z and SLPNX,SLPNY, SLPNZ.
- IND = 1 Values are supplied for SLP1X,SLP1Y,SLP1Z but not for SLPNX, SLPNY, SLPNZ.
- IND = 2 Values are supplied for SLPNX,SLPNY,SLPNZ but not for SLP1X, SLP1Y, SLP1Z.
- IND = 3 No values are supplied for SLP1X, SLP1Y, SLP1Z and SLPNX, SLPNY, SLPNZ.

If a value is not supplied by the user, then the routine provides a value.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s), y(s), z(s)$ approximate cubic splines. Otherwise, if $|\sigma|$ is large, say $|\sigma| > 100$, then the resulting curve approximates the polygonal line from (x_1, y_1, z_1) to (x_n, y_n, z_n).

IERR is an integer variable and S, DDX, DDY, DDZ are arrays of dimension n or larger. When QURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1, \ldots, s_n$ are computed and stored in S. Also, the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1, \ldots, n)$ are computed and stored in DDX, DDY, DDZ.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$.
IERR = 2 if $(x_i, y_i, z_i) = (x_{i+1}, y_{i+1}, z_{i+1})$ for some i.

When either of these errors is detected, the routine immediately terminates.

Remark. After S, DDX, DDY, DDZ are obtained, QURV2 may be used to compute the curve.

Programming. QURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. QURV1, CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL QURV2(t, XT,YT,ZT, n, X, Y, Z, DDX,DDY,DDZ, S, σ)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and Z an array containing $z_1, \ldots, z_n$. S is an array containing $s_1, \ldots, s_n$ and DDX, DDY, DDZ are arrays containing the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1, \ldots, n)$.

Now consider the change of variables $t = s/s_n$ and let $t \mapsto (\bar{x}(t), \bar{y}(t), \bar{z}(t))$ denote the curve in terms of the new parameter t. XT, YT, ZT are real variables. For any $0 \le t \le 1$, QURV2 computes the point $(\bar{x}(t), \bar{y}(t), \bar{z}(t))$ on the curve and assigns XT, YT, ZT the values $\bar{x}(t), \bar{y}(t), \bar{z}(t)$.

Remark. After DDX, DDY, DDZ have been obtained, QURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified then the derivative information in DDX, DDY, DDZ will have to be recomputed before QURV2 can be used with the new tension factor.

Programming. QURV2 employs the function INTRVL and subroutine SNHCSH. QURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin).

B-SPLINES

For $k \geq 1$ let $f(t) = (t - x)^{k-1}$ when $t > x$ and $f(t) = 0$ when $t \leq x$. Then for any sequence $t_i \leq \cdots \leq t_{i+k}$ where $t_i < t_{i+k}$, let $B_{ik}(x) = (t_{i+k} - t_i) f[t_i, \ldots, t_{i+k}]$ where $f[t_i, \ldots, t_{i+k}]$ is the k^{th} order divided difference of $f(t)$. The function B_{ik} is called a ***B-spline of order k***. For $k = 1$ it follows that

$$B_{i1}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1} \\ 0 & \text{otherwise.} \end{cases}$$

More generally, for $k \geq 2$ $B_{ik}(x) = 0$ when $x \notin [t_i, t_{i+k})$. For $t_i \leq x < t_{i+k}$

$$B_{ik}(x) = \left(\frac{t_{i+k} - x}{t_{i+k} - t_i}\right)^{k-1} \qquad \text{when } t_i = \cdots = t_{i+k-1} \text{ and}$$

$$B_{ik}(x) = \left(\frac{x - t_i}{t_{i+k} - t_i}\right)^{k-1} \qquad \text{when } t_{i+1} = \cdots = t_{i+k}.$$

Otherwise, if no point appears more than $k - 1$ times in the sequence $\{t_i, \ldots, t_{i+k}\}$ then

$$B_{ik}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(x).$$

From these relations it follows that $B_{ik}(x) > 0$ when $t_i < x < t_{i+k}$.

Now let $\xi_1 < \cdots < \xi_{\ell+1}$ be a sequence of points where $\ell \geq 1$. If m_j is an integer where $1 \leq m_j \leq k - 1$ for $j = 2, \ldots, \ell$, let $P_k[\xi_1, \ldots, \xi_{\ell+1}; m_2, \ldots, m_\ell]$ denote the collection of piecewise polynomials $p(x)$ where $p(x)$ is a polynomial of order $\leq k$ (degree $\leq k - 1$) on each interval $[\xi_j, \xi_{j+1})$ $(j = 1, \ldots, \ell)$, and $p(x)$ is of class C^{k-1-m_j} at each point ξ_j $(j = 2, \ldots, \ell)$. Then $\xi_1, \ldots, \xi_{\ell+1}$ are called the ***knots*** or ***break points*** for these piecewise polynomials, $\xi_2, \ldots, \xi_\ell$ the ***interior knots***, and $m_2, \ldots, m_\ell$ the ***multiplicities*** of the interior knots.

If $n = k + m_2 + \cdots + m_\ell$, consider any sequence $t_1 \leq \cdots \leq t_{n+k}$ where

(1) $t_1 \leq \cdots \leq t_k \leq \xi_1$,

(2) $t_{k+1}, \ldots, t_n$ are the interior knots, where each interior knot ξ_j appears exactly m_j times, and

(3) $\xi_{\ell+1} \leq t_{n+1} \leq \cdots \leq t_{n+k}$.

Then the B-splines $B_{1k}, \ldots, B_{nk}$ are piecewise polynomials in $P_k[\xi_1, \ldots, \xi_{\ell+1}; m_2, \ldots, m_\ell]$, forming a basis for this vector space. Consequently, any piecewise polynomial $p(x)$ in the space can be represented uniquely in the form $p = \sum_{i=1}^{n} a_i B_{ik}$. This representation is called a ***B-spline representation*** for $p(x)$. If $p = \sum_{i=1}^{n} a_i B_{ik}$ then we note that $p(x) = 0$ if $x < t_1$ or $x \geq t_{n+k}$.

Remark. If $t_j < x < t_{j+1}$ where $k \leq j < n$, then $B_{ik}(x) \neq 0$ only when $i = j + 1 - k, \ldots, j$. Also $\sum_i B_{ik}(x) = 1$.

FINDING THE INTERVAL THAT CONTAINS A POINT

Let $t_1 \leq \cdots \leq t_m$ be a sequence where $t_1 < t_m$. Then for any $t_1 \leq x < t_m$, the function INTRVL finds the interval $[t_i, t_{i+1})$ that contains x.

INTRVL(x, T, m)

T is an array containing $t_1, \ldots, t_m$ and x a real number. If $t_1 \leq x < t_m$ then INTRVL has the value i where $t_i \leq x < t_{i+1}$. Otherwise,

$$\text{INTRVL}(x, T, m) = \begin{cases} 1 & \text{if } x < t_1 \\ \ell & \text{if } x \geq t_m \end{cases}$$

where ℓ is the integer such that $t_\ell < t_{\ell+1}$ and $t_{\ell+1} = \cdots = t_m$.

Programmer. A. H. Morris.

EVALUATION AND DIFFERENTIATION OF PIECEWISE POLYNOMIALS FROM THEIR B-SPLINE REPRESENTATIONS

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_k < t_{n+1}$ and $t_i < t_{i+k}$ for $i = 1, \ldots, n$. If $f(x) = \sum_{i=1}^{n} a_i B_{ik}(x)$ then the following subroutine is available for evaluating $f(x)$ and computing its derivatives.

CALL BVAL$(T, A, n, k, x, j, w,$WK$)$

T is an array containing $t_1, \ldots, t_{n+k}$ and A an array containing $a_1, \ldots, a_n$. The argument j is a nonnegative integer, x is a real number at which $f(x)$ or its j^{th} derivative $f^{(j)}(x)$ is to be computed, and w is a variable. If $x \neq t_{n+k}$ then w is assigned the value $f(x+)$ if $j = 0$ and the value $f^{(j)}(x+)$ if $j \geq 1$. Otherwise, if $x = t_{n+k}$ then $w = f(t_{n+k}-)$ if $j = 0$ and $w = f^{(j)}(t_{n+k}-)$ if $j \geq 1$.

WK is an array of dimension $3k$ or larger that is a work space for the routine.

Remark. The left limits $f(t_{n+k}-)$ and $f^{(j)}(t_{n+k}-)$ are the only limits of interest when $x = t_{n+k}$ since $f(x) = 0$ for all $x \geq t_{n+k}$.

Programming. BVAL employs the function INTRVL. BVAL is a modified version by A. H. Morris of the function BVALUE, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

EVALUATION OF THE INDEFINITE INTEGRAL OF A PIECEWISE POLYNOMIAL FROM ITS B-SPLINE REPRESENTATION

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_k < t_{n+1}$ and $t_i < t_{i+k}$ for $i = 1, \ldots, n$. If $f = \sum_{i=1}^{n} a_i B_{i,k}$ then the following subroutine is available for computing $F(x) = \int_{t_1}^{x} f(t)\, dt$.

CALL BVALI$(T, A, n, k, x, w, \mathrm{WK})$

T is an array containing $t_1, \ldots, t_{n+k}$ and A an array containing $a_1, \ldots, a_n$. The argument x is a real number and w is a variable. When BVALI is called, w is assigned the value $F(x)$.

WK is an array of dimension $n + 3(k+1)$ or larger that is a work space for the routine.

Algorithm. For $m = n + k$, let $t_{m+1}, \ldots, t_{m+k}$ be any sequence where $t_m < t_{m+1} \leq \cdots \leq t_{m+k}$. If $x \leq t_m$ then $F(x) = \sum_{i=1}^{m-1} b_i B_{i,k+1}(x)$ is used where $b_i = \frac{1}{k} \sum_{j=1}^{i} a_j (t_{j+k} - t_j)$ for $i = 1, \ldots, n$ and $b_i = b_n$ for $i = n+1, \ldots, m-1$.

Programming. BVALI employs the subroutine BVALI0 and the function INTRVL. BVALI was written by A. H. Morris.

CONVERSION OF PIECEWISE POLYNOMIALS FROM B-SPLINE TO TAYLOR SERIES FORM

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1, \ldots, n$. Further assume that $t_k < t_{n+1}$ and let $f(x) = \sum_{i=1}^{n} a_i B_{ik}(x)$ for $t_k \leq x < t_{n+1}$. If $\xi_1 < \cdots < \xi_{\ell+1}$ are the distinct points in the sequence $\{t_k, \ldots, t_{n+1}\}$ then the piecewise polynomial f can be represented in the form $f(x) = \sum_{i=1}^{k} c_{ij}(x - \xi_j)^{i-1}$ for $\xi_j \leq x < \xi_{j+1}$ $(j = 1, \ldots, \ell)$. The following subroutine is available for obtaining the coefficients c_{ij} of this representation.

CALL BSPP$(T, A, n, k, \text{BREAK}, C, L, \text{WK})$

T is an array containing $t_1, \ldots, t_{n+k}$ and A an array containing $a_1, \ldots, a_n$. BREAK is an array of dimension $\ell + 1$ or larger, C a 2-dimensional array of dimension $k \times \ell$, and L a variable. When BSPP is called then L is assigned the value ℓ (which is computed by the routine), the break points $\xi_1 < \cdots < \xi_{\ell+1}$ are found and stored in BREAK, and the coefficients c_{ij} are computed and stored in C. The j^{th} column of the matrix C then contains the coefficients of the j^{th} polynomial forming f $(j = 1, \ldots, \ell)$.

WK is an array of dimension $k(k + 1)$ or larger that is a work space for the routine.

Remarks.

(1) Since $\ell \leq n - k + 1$, BREAK may be declared to be of dimension $n - k + 2$ and C to be of dimension $k \times (n - k + 1)$.

(2) After C is obtained, then PPVAL may be used to evaluate $f(x)$ for any $x \in [\xi_1, \xi_{\ell+1}]$. However, PPVAL cannot be used for evaluating $f(x)$ if $x < \xi_1$ or $x > \xi_{\ell+1}$. BVAL must be used in this case.

Programming. BSPP is a modified version by A. H. Morris of the subroutine BSPLPP, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

EVALUATION OF PIECEWISE POLYNOMIALS FROM THEIR TAYLOR SERIES REPRESENTATIONS

Given $x_1 < \cdots < x_{\ell+1}$. If f is a piecewise polynomial where $f(x) = \sum_{i=1}^{k} c_{ij}(x - x_j)^{i-1}$ for $x_j \le x < x_{j+1}$ $(j = 1, \ldots, \ell)$, then the following subroutine is available for computing f at any point x.

CALL PPVAL$(X, C, k, \ell, \mathrm{XI}, \mathrm{YI}, m)$

X is an array containing the knots $x_1, \ldots, x_\ell$ and C a $k \times \ell$ matrix containing the coefficients c_{ij}. It is assumed that $k \ge 1$ and $\ell \ge 1$. Let $\bar{x}_1, \ldots, \bar{x}_m$ be the points at which f is to be evaluated. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$ and YI an array of dimension m or larger. When PPVAL is called, $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Remarks.

(1) X need not contain the knot $x_{\ell+1}$.

(2) It is not required that f be continuous at an interior knot x_i. If x_i appears in XI then $f(x_i+)$ is computed.

(3) It is not required that the output points $\bar{x}_j$ in XI be in the interval $[x_1, x_{\ell+1})$. If $\bar{x}_j < x_1$ then $\sum_i c_{i1}(x - x_1)^{i-1}$ is evaluated at $\bar{x}_j$. Otherwise, if $\bar{x}_j \ge x_{\ell+1}$ then $\sum_i c_{i\ell}(x - x_\ell)^{i-1}$ is evaluated at $\bar{x}_j$.

Programming. PPVAL is an adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

PIECEWISE POLYNOMIAL INTERPOLATION

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1, \ldots, n$. Consider a set of points $\{(x_i, y_i) : i = 1, \ldots, n\}$ where $t_k \leq x_1 < \cdots < x_n \leq t_{n+1}$. Then we wish to find a piecewise polynomial $f = \sum_{i=1}^{n} a_i B_{ik}$ which satisfies $f(x_i) = y_i$ for $i = 1, \ldots, n$.[1] This problem has a unique solution when $t_i < x_i < t_{i+k}$ for $1 \leq i \leq n$. The following subroutine is available for obtaining the coefficients $a_1, \ldots, a_n$ of the interpolating piecewise polynomial.

CALL BSTRP$(X, Y, T, n, k, A, \text{WK}, \text{IFLAG})$

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and T an array containing $t_1, \ldots, t_{n+k}$. A is an array of dimension n or larger, and IFLAG an integer variable. On an initial call to the routine the user may assign IFLAG any nonzero value. In this case, if no errors are detected then IFLAG is reset by the routine to 0 and the B-spline coefficients $a_1, \ldots, a_n$ are computed and stored in A. The routine may be recalled with IFLAG = 0 on input when only Y is modified. In this case, no error checking is performed and IFLAG= 0 on output. Also the B-spline coefficients $a_1, \ldots, a_n$ of the new interpolating piecewise polynomial are computed and stored in A.

WK is an array of dimension $(2k - 1)n$ or larger that is used for temporary storage by the routine. When BSTRP terminates, WK contains information needed for subsequent calls to the routine.

Error Return. IFLAG is assigned the value 1 if any of the conditions

$$x_1 < \cdots < x_n$$
$$t_k \leq x_1 < t_{k+1}$$
$$t_i < x_i < t_{i+k} \text{ for } 1 < i < n$$
$$t_n < x_n \leq t_{n+1}$$

is violated. When an error is detected, the routine immediately terminates.

Remarks.

(1) It is recommended that $t_1 = \cdots = t_k$ and $t_{n+1} = \cdots = t_{n+k}$.
(2) After the B-spline representation $\sum_i a_i B_{ik}$ is obtained, then the subroutine BVAL can be used to evaluate or differentiate the piecewise polynomial.

Example Given $n > 4$ data points (x_i, y_i), then for $k = 4$ one may set $t_1 = \cdots = t_k = x_1$, $t_{i+k} = x_{i+2}$ for $i = 1, \ldots, n - k$, and $x_n = t_{n+1} = \cdots = t_{n+k}$. Then $x_3, \ldots, x_{n-2}$ are the interior knots for the interpolating piecewise polynomial f. Here we have cubic spline interpolation where the data points x_2 and x_{n-1} are not knots for f.

[1] If $x_n = t_{n+1}$ then by $f(x_n)$ we mean $f(x_n -)$

Programming. BSTRP calls the subroutines BSPVB, BANFAC, and BANSLV. BSTRP is a modified version by A. H. Morris of the subroutine SPLINT, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

WEIGHTED LEAST SQUARES PIECEWISE POLYNOMIAL FITTING

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1, \ldots, n$. Consider a set of points $\{(x_i, y_i) : i = 1, \ldots, m\}$ where $t_k \leq x_1 \leq \cdots \leq x_m \leq t_{n+1}$ and $m \geq \max\{2, k\}$. Let $w_i > 0$ $(i = 1, \ldots, m)$ be weights. Then the following subroutine is available for finding a piecewise polynomial $f = \sum_{i=1}^{n} a_i B_{ik}$ which minimizes $\sum_{i=1}^{m} w_i (f(x_i) - y_i)^2$.[1]

CALL BSLSQ$(X, Y, W, m, T, n, k, A, \text{WK}, Q, \text{IERR})$

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, W an array containing $w_1, \ldots, w_m$, and T an array containing $t_1, \ldots, t_{n+k}$. A is an array of dimension n or larger. When BSLSQ is called, if no input errors are detected then the coefficients $a_1, \ldots, a_n$ of a least squares approximation f are computed and stored in A.

WK is an array of dimension n or larger, and Q an array of dimension kn or larger. WK and Q are work spaces for the routine.

IERR is a variable that reports the status of the results. When BSLSQ terminates, IERR has one of the following values:

IERR = 0 The coefficients were obtained. The least squares approximation is unique.

IERR = 1 The coefficients for a least squares approximation were obtained. The approximation is not unique.

IERR = −1 (Input error) Either $t_k \leq x_1 \leq \cdots \leq x_m \leq t_{n+1}$ or $m \geq \max\{2, k\}$ is violated.

Selection of $t_k \leq \cdots \leq t_{n+1}$ given the data (x_i, y_i). It is recommended that the knots t_ν be selected so that there are data points $x_{i_1} < \cdots < x_{i_n}$ satisfying $t_\nu < x_{i_\nu} < t_{\nu+k}$ for $\nu = 1, \ldots, n$. If these conditions are satisfied then the least squares approximation is unique.

Remark. After the B-spline representation is obtained, then the subroutine BVAL can be used to evaluate or differentiate the piecewise polynomial.

Programming. BSLSQ calls the subroutines BSPVB, BCHFAC, and BCHSLV. BSLSQ is a modified version by A. H. Morris of the subroutine L2APPR, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

[1] If $x_i = t_{n+1}$ then by $f(x_i)$ we mean $f(x_i -)$.

LEAST SQUARES PIECEWISE POLYNOMIAL FITTING WITH EQUALITY AND INEQUALITY CONSTRAINTS

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_i < t_{i+k}$ $(i = 1, \ldots, n)$. Consider a set of points $\{(x_i, y_i) : i = 1, \ldots, m\}$ where $t_k \leq x_1 < \cdots < x_m \leq t_{n+1}$. Then the subroutine BFIT is available for obtaining a piecewise polynomial $f = \sum_{i=1}^{n} a_i B_{ik}$ which minimizes $\sum_i (f(x_i) - y_i)^2$ subject to a set of constraints.

Notation. If $t_k \leq x < t_{n+1}$ then $f^{(j)}(x)$ will denote the value $f(x+)$ if $j = 0$, and the right j^{th} derivative $f^{(j)}(x+)$ if $j \geq 1$. Otherwise, if $x = t_{n+1}$ then $f^{(j)}(x) = f(t_{n+1}-)$ if $j = 0$ and $f^{(j)}(x) = f^{(j)}(t_{n+1}-)$ if $j \geq 1$.

CALL BFIT(T, n, k, X, Y, m,XCON,C,NDER,ℓ,IND,A, r,WK,IWK)

T is an array containing $t_1, \ldots, t_{n+k}$, X an array containing $x_1, \ldots, x_m$, and Y an array containing $y_1, \ldots, y_m$. The argument ℓ is the number of constraints to be satisfied $(\ell \geq 0)$ and XCON, C, NDER are arrays of dimension $\max\{1, \ell\}$ or larger.

If $\ell \geq 1$ then XCON contains the points $\bar{x}_1, \ldots, \bar{x}_\ell$ at which the constraints are placed. The points $\bar{x}_i$ may be given in any order and need not be distinct. It is assumed that $t_k \leq \bar{x}_i \leq t_{n+1}$ for $i = 1, \ldots, \ell$. At $\bar{x}_i$ any of the following types of constraints

itype = 0	$f^{(j)}(\bar{x}_i) \leq c_i$
itype = 1	$f^{(j)}(\bar{x}_i) \geq c_i$
itype = 2	$f^{(j)}(\bar{x}_i) = c_i$
itype = 3	$f^{(j)}(\bar{x}_i) = f^{(j)}(c_i)$

can be imposed where j is a nonnegative integer. For the selected constraint, it is assumed that $C(i) = c_i$ and NDER$(i) =$ *itype* $+ 4j$. If an *itype* 3 constraint is imposed then it is further assumed that $t_k \leq c_i \leq t_{n+1}$.

WK and IWK are arrays that are work spaces for the routine. On input, IWK(1) is the dimension of WK and IWK(2) the dimension of IWK. If $\ell = 0$ then one may set IWK(1) = NB and IWK(2) = 2 where NB $= (n+3)(k+1) + k^2$. Otherwise, let $\nu = n + 1$, m_e be the number of equality (i.e., *itype* 2 and 3) constraints, and m_i the number of inequality (*itype* 0 and 1) constraints. Then one must set

$$\text{IWK}(1) \geq \text{NB} + (\nu + \ell)\nu + 2(m_e + \nu) + (m_i + \nu) + (m_i + 2)(\nu + 6)$$
$$\text{IWK}(2) \geq m_i + 2\nu.$$

Remarks.

(1) If $\ell > 1$ then one can let the routine determine the amount of storage that is needed for WK and IWK. Set IWK(1) = NB and IWK(2) = 2, in which case errors will occur (see the IND = −6, −7 error values below).

(2) NDER(i) may be assigned a negative value ($i = 1, \ldots, \ell$). If NDER(i) = 0 then the i^{th} constraint is ignored

IND is a variable used for input/output purposes, A an array of dimension n or larger, and r a variable. IND must be set to 0 on an initial call to BFIT. When the routine is called, if f is found then IND is assigned the value 0 or 1, the coefficients. $a_1, \ldots, a_n$ are stored in A, and r is assigned the value $\sqrt{\sum_i (f(x_i) - y_i)^2}$.

After an initial call to BFIT, if only the constraints are modified, then the information in WK concerning X and Y can be reused. In this case, one may set IND to a nonzero value and recall BFIT to solve the new problem. The input setting IND $\neq 0$ can be used only when T, n, k, X, Y, m, and WK have not been modified since the last call to BFIT.

When BFIT terminates, IND reports the status of the results. IND is assigned one of the following values:

IND = 0 The piecewise polynomial f was obtained.

IND = 1 The equality (i.e., *itype* 2 and 3) constraints are contradictory. The solution f was obtained where the equality constraints were satisfied in a least squares sense.

IND = 2 The problem could not be solved. The constraints are contradictory.

IND = −1 $k < 1$ or $k > n$.

IND = −2 $m \leq 0$ or $\ell < 0$.

IND = −3 $t_i > t_{i+1}$ for some i.

IND = −4 The assumption $t_k \leq x_1 < \cdots < x_m \leq t_{n+1}$ is not satisfied.

IND = −5 The i^{th} constraint is incorrect for the value of i stored in IWK(2).

IND = −6 Insufficient storage was specified for the WK array. IWK(1) has been reset to the amount of storage needed by WK.

IND = −7 Insufficient storage was specified for the IWK array. IWK(2) has been reset to the amount of storage needed by IWK.

IWK(1) and IWK(2) are not modified when IND $\neq -5, -6, -7$.

Example. If one wants f to be convex on an interval $[t_i, t_{i+1})$, then this is equivalent to requiring that $f''(x) \geq 0$ on the interval. Thus, if $k = 4$ then it suffices to require that $f''(t_i+) \geq 0$ and $f''(t_{i+1}-) > 0$. For these two constraints, 9 is the appropriate value for the NDER array.

Programming. BFIT employs the routines BFIT0, BNDACC,BNDSL, BSPVB, BSPVD, LSEI, and the subroutines and functions used by LSEI. BFIT is a modification by A. H. Morris of the subroutine FC, written by Richard J. Hanson (Sandia Laboratories). The subroutines BNDACC and BNDSL were written by Richard J. Hanson and Charles L. Lawson (Jet Propulsion Laboratory), and are described in reference (2).

References

(1) Hanson, R.J., ***Constrained Least Squares Curve Fitting to Discrete Data using B-Splines — A User's Guide***, Report SAND 78-1291, Sandia Laboratories, Albuquerque, New Mexico, 1979.

(2) Lawson, C.L., and Hanson, R.J., ***Solving Least Squares Problems***, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

BICUBIC SPLINES AND BISPLINES UNDER TENSION

Given a real value σ and a set $X = \{x_1, \ldots, x_m\}$ where $x_1 < \cdots < x_m$. Let $S_\sigma(X)$ denote the collection of splines having the tension factor σ and knots $x_1, \ldots, x_m$. If $\sigma = 0$ then $S_\sigma(X)$ is the collection of cubic splines with the knots $x_1, \ldots, x_m$.

Now $S_\sigma(X)$ is a vector space, where each of its splines $f(x)$ is uniquely characterized by the values $f(x_1), \ldots, f(x_m)$ and derivatives $f'(x_1)$ and $f'(x_m)$. Let $\psi_i(x)$ $(i = 1, \ldots, m)$ be the spline in $S_\sigma(X)$ where

$$\begin{aligned}
&\psi_i(x_i) = 1\\
&\psi_i(x_k) = 0 \quad \text{for } k \neq i\\
&\psi_i'(x_1) = \psi_i'(x_m) = 0,
\end{aligned}$$

and let ψ_{m+1}, ψ_{m+2} be the splines where

$$\begin{aligned}
&\psi_{m+1}(x_i) = 0 \qquad \psi_{m+2}(x_i) = 0 \qquad (i = 1, \ldots, m)\\
&\psi_{m+1}'(x_1) = 1 \qquad \psi_{m+2}'(x_1) = 0\\
&\psi_{m+1}'(x_m) = 0 \qquad \psi_{m+2}'(x_m) = 1.
\end{aligned}$$

Then $\{\psi_1, \ldots, \psi_{m+2}\}$ is a basis for $S_\sigma(X)$ and $f = \sum_{i=1}^{m} f(x_i)\psi_i + f'(x_1)\psi_{m+1} + f'(x_m)\psi_{m+2}$ for each spline $f(x)$ in $S_\sigma(X)$.

Let $Y = \{y_1, \ldots, y_n\}$ where $y_1 < \cdots < y_n$ and let $\{\overline{\psi}_1, \ldots, \overline{\psi}_{n+2}\}$ be the corresponding basis for $S_\sigma(Y)$. Then a function of the form

$$F(x, y) = \sum_{i=1}^{m+2} \sum_{j=1}^{n+2} a_{ij}\psi_i(x)\overline{\psi}_j(y)$$

is called a ***bispline with the tension factor*** σ (also, ***bicubic spline*** when $\sigma = 0$). A bispline $F(x, y)$ has the following properties:

(1) $F(x, y)$ is a C^2 mapping on $[x_1, x_n] \times [y_1, y_n]$.

(2) For each fixed y the mapping $x \mapsto F(x, y)$ is a spline in $S_\sigma(X)$, and for each fixed x the mapping $y \mapsto F(x, y)$ is a spline in $S_\sigma(Y)$.

(3) F is uniquely characterized by the values $F(x_i, y_j)$, $D_1F(x_k, y_j)$, $D_2F(x_i, y_\ell)$, and $D_1D_2F(x_k, y_\ell)$ where $i = 1, \ldots, m$, $j = 1, \ldots, n$, $k = 1, m$, and $\ell = 1, n$.[1]

If $\sigma = 0$ then a bicubic spline $F(x, y)$ has the value

$$F(x, y) = \sum_{k=0}^{3} \sum_{\ell=0}^{3} c_{k\ell}^{ij}(x - s_i)^k(y - t_j)^\ell$$

for $s_i = x < s_{i+1}$ and $t_j = y < t_{j+1}$ where $c_{k\ell}^{ij}$ are constants $(1 \le i \le m-1,\ 1 \le j \le n)$

[1] D_1F and D_2F denote the partial derivatives of F.

WEIGHTED LEAST SQUARES BICUBIC SPLINE FITTING

Given $u_1 < \cdots < u_m$ and $v_1 < \cdots < v_n$ where $m, n \geq 2$. Let $\{(x_r, y_s, z_{rs}) : r = 1, \ldots, \mu; s = 1, \ldots, \nu\}$ be a set of points where $\mu, \nu \geq 4$, $u_1 \leq x_1 \leq \cdots \leq x_\mu \leq u_m$, and $v_1 \leq y_1 \leq \cdots \leq y_\nu \leq v_n$. Also let $w_1, \ldots, w_\mu$ and $\overline{w}_1, \ldots, \overline{w}_\nu$ be positive weights. Then the following subroutine is available for finding a bicubic spline $f(x, y)$ having the knots u_i, v_j $(i = 1, \ldots, m, j = 1, \ldots, n)$ which minimizes $\sum_{r=1}^{\mu} \sum_{s=1}^{\nu} w_r \overline{w}_s [f(x_r, y_s) - w_{rs}]^2$.

CALL SPFIT2(X,WX,μ,Y,WY,ν,Z,kz,U,m,V,n,F,S,T,WK,NUM,IERR)

X is an array containing $x_1, \ldots, x_\mu$, WX an array containing $w_1, \ldots, w_\mu$, Y an array containing $y_1, \ldots, y_\nu$, WY an array containing $\overline{w}_1, \ldots, \overline{w}_\nu$, Z the $\mu \times \nu$ matrix (z_{rs}), U an array containing $u_1, \ldots, u_m$, and V an array containing $v_1, \ldots, v_n$. It is assumed that $kz \geq \mu$, where kz is the number of rows in the dimension statement for Z in the calling program.

F is a 3-dimensional array of dimension $m \times n \times 4$, and IERR a variable. When SPFIT2 is called, if no input errors are detected then IERR is set to 0 and the values $f(u_i, v_j)$, $D_1 f(u_i, v_j)$, $D_2 f(u_i, v_j)$, $D_1 D_2 f(u_i, v_j)$ of a weighted least squares bispline approximation $f(x, y)$ are computed and stored in F. Specifically,

$$\begin{aligned} F(i, j, 1) &= f(u_i, v_j) \\ F(i, j, 2) &= D_2 f(u_i, v_j) \\ F(i, j, 3) &= D_1 f(u_i, v_j) \\ F(i, j, 4) &= D_1 D_2 f(u_i, v_j) \end{aligned}$$

for each $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

S is an array of dimension $m + 6$, T an array of dimension $n + 6$, and WK an array of dimension NUM where NUM $\geq \max\{(m + 2)(n + 2), (m + 2)\nu + 5\max\{m + 2, n + 2\}\}$. S, T, and WK are work spaces for the routine.

Error Return. IERR has one of the following values if an input error is detected:

IERR = 1 Either $u_1 < \cdots < u_n$ or $m < 2$ is violated.
IERR = 2 Either $u_1 < x_1 < \cdots < x_\mu < u_m$ or $\mu < 4$ is violated.
IERR = 3 Either $v_1 < \cdots < v_n$ or $n < 2$ is violated.
IERR = 4 Either $v_1 < y_1 < \cdots < y_\nu < v_n$ or $\nu < 4$ is violated.
IERR = 5 NUM is too small. The minimum permissible value for NUM has been stored in WK(1). Reset NUM $\geq$ WK(1).

Remark. After the array F is obtained, then the subroutine CURF1 can be used to evaluate or differentiate the bispline $f(x, y)$.

f, $D_1 f$, and $D_2 f$ denote the partial derivatives of f.

Programming. SPFIT2 employs the routines CSSP, BSLSQ2, BSLSQ1, BSPVB, BCHFAC, and BCHSV1. SPFIT2 was written by A. H. Morris.

EVALUATION AND DIFFERENTIATION OF BICUBIC SPLINES

Given $u_1 < \cdots < u_m$ and $v_1 < \cdots < v_n$ where $m, n \geq 2$. Then a bicubic spline $f(x, y)$ having the knots (u_i, v_j) is uniquely defined by either of the following sets of data:[1]

(1) the values $f(u_i, v_j)$ and derivatives $D_1 f(u_i, v_j), D_2 f(u_i, v_j), D_1 D_2 f(u_i, v_j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$

(2) the values $f(u_i, v_j)$ and derivatives $D_1^2 f(u_i, v_j), D_2^2 f(u_i, v_j), D_1^2 D_2^2 f(u_i, v_j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$

The subroutines CSURF and CSURF1 are available for computing $f(x, y)$ or its derivative $D_1^k D_2^\ell f(x, y)$ if data set (1) is given, and the subroutines CSRF and CSRF2 are available for computing $f(x, y)$ or its derivative if data set (2) is given.

CALL CSURF$(k, \ell, X, \mu, Y, \nu, Z, kz, U, m, V, n, W, kw,$
$W_1, kw_1, W_2, kw_2, W_{12}, kw_{12})$

U is an array containing $u_1, \ldots, u_m$, V an array containing $v_1, \ldots, v_n$, and W, W_1, W_2, W_{12} the $m \times n$ matrices $\big(f(u_i, v_j)\big)$, $\big(D_1 f(u_i, v_j)\big)$, $\big(D_2 f(u_i, v_j)\big)$, $\big(D_1 D_2 f(u_i, v_j)\big)$. The arguments kw, kw_1, kw_2, and kw_{12} have the following values:

kw = the number of rows in the dimension statement for W in the calling program

kw_1 = the number of rows in the dimension statement for W_1 in the calling program

kw_2 = the number of rows in the dimension statement for W_2 in the calling program

kw_{12} = the number of rows in the dimension statement for W_{12} in the calling program

It is assumed that $kw \geq m$, $kw_1 \geq m$, $kw_2 \geq m$, and $kw_{12} \geq m$.

The arguments k and ℓ are nonnegative integers. Let (x_r, y_s) $(r = 1, \ldots, \mu,\ s = 1, \ldots, \nu)$ be the points where $D_1^k D_2^\ell f$ is to be computed. X is an array containing $x_1, \ldots, x_\mu$, Y an array containing $y_1, \ldots, y_\nu$, and Z a 2-dimensional array of dimension $kz \times \nu$ where $kz \geq \mu$. When CSURF is called, the values $D_1^k D_2^\ell f(x_r, y_s)$ are computed and stored in Z for $r = 1, \ldots, \mu$ and $s = 1, \ldots, \nu$.

Programming. CSURF calls the function INTRVL. CSURF was written by A. H. Morris.

CALL CSURF1$(k, \ell, X, \mu, Y, \nu, Z, kz, U, m, V, n, W)$

U is an array containing $u_1, \ldots, u_m$, V an array containing $v_1, \ldots, v_n$, and W a 3-dimensional array of dimension $m \times n \times 4$. It is assumed that

$$\begin{aligned} W(i, j, 1) &= f(u_i, v_j) \\ W(i, j, 2) &= D_2 f(u_i, v_j) \\ W(i, j, 3) &= D_1 f(u_i, v_j) \\ W(i, j, 4) &= D_1 D_2 f(u_i, v_j) \end{aligned}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

[1] $D_1 f$ and $D_2 f$ denote the partial derivatives of f, and $D_1^k D_2^\ell f = \partial^{k+\ell} f / \partial^k x \partial^\ell y$ $(k, \ell = 0, 1, \ldots)$. If $k = 3$ or $\ell = 3$ then $D_1^k D_2^\ell f(x, y)$ denotes the right limit $D_1^k D_2^\ell f(x+, y+)$.

The arguments k and ℓ are nonnegative integers. Let (x_r, y_s) $(r = 1, \ldots, \mu,\ s = 1, \ldots, \nu)$ be the points where $D_1^k D_2^\ell f$ is to be computed. X is an array containing $x_1, \ldots, x_\mu$, Y an array containing $y_1, \ldots, y_\nu$, and Z a 2-dimensional array of dimension $kz \times \nu$ where $kz \geq \mu$. When CSURF1 is called, the values $D_1^k D_2^\ell f(x_r, y_s)$ are computed and stored in Z for $r = 1, \ldots, \mu$ and $s = 1, \ldots, \nu$.

Programming. CSURF1 is a driver for the subroutine CSURF. The function INTRVL is used. CSURF1 was written by A. H. Morris.

CALL CSRF$(k, \ell, X, \mu, Y, \nu, Z, kz, U, m, V, n, W, kw,$
$W_1, kw_1, W_2, kw_2, W_{12}, kw_{12})$

U is an array containing $u_1, \ldots, u_m$, V an array containing $v_1, \ldots, v_n$, and W, W_1, W_2, W_{12} the $m \times n$ matrices $\big(f(u_i, v_j)\big)$, $\big(D_1^2 f(u_i, v_j)\big)$, $\big(D_2^2 f(u_i, v_j)\big)$, $\big(D_1^2 D_2^2 f(u_i, v_j)\big)$. The arguments kw, kw_1, kw_2, and kw_{12} have the following values:

kw = the number of rows in the dimension statement for W in the calling program

kw_1 = the number of rows in the dimension statement for W_1 in the calling program

kw_2 = the number of rows in the dimension statement for W_2 in the calling program

kw_{12} = the number of rows in the dimension statement for W_{12} in the calling program

It is assumed that $kw \geq m$, $kw_1 \geq m$, $kw_2 \geq m$, and $kw_{12} \geq m$.

The arguments k and ℓ are nonnegative integers. Let (x_r, y_s) $(r = 1, \ldots, \mu,\ s = 1, \ldots, \nu)$ be the points where $D_1^k D_2^\ell f$ is to be computed. X is an array containing $x_1, \ldots, x_\mu$, Y an array containing $y_1, \ldots, y_\nu$, and Z a 2-dimensional array of dimension $kz \times \nu$ where $kz \geq \mu$. When CSRF is called, the values $D_1^k D_2^\ell f(x_r, y_s)$ are computed and stored in Z for $r = 1, \ldots, \mu$ and $s = 1, \ldots, \nu$.

Programming. CSRF calls the function INTRVL. CSRF was written by A. H. Morris.

CALL CSRF2$(k, \ell, X, \mu, Y, \nu, Z, kz, U, m, V, n, W, kw, \mathrm{DDW})$

U is an array containing $u_1, \ldots, u_m$, V an array containing $v_1, \ldots, v_n$, W the $m \times n$ matrix $\big(f(u_i, v_j)\big)$, and DDW a 3-dimensional array of dimension $m \times n \times 3$ where

$$\begin{array}{ll} W(i,j,1) & D_2^2 f(u_i, v_j) \\ W(i,j,2) & D_1^2 f(u_i, v_j) \\ W(i,j,3) & D_1^2 D_2^2 f(u_i, v_j) \end{array}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The argument kw has the following value:

kw = the number of rows in the dimension statement for W in the calling program

It is assumed that $kw \geq m$.

The arguments k and ℓ are nonnegative integers. Let (x_r, y_s) $(r = 1, \ldots, \mu, s = 1, \ldots, \nu)$ be the points where $D_1^k D_2^\ell f$ is to be computed. X is an array containing $x_1, \ldots, x_\mu$, Y an array containing $y_1, \ldots, y_\nu$, and Z a 2-dimensional array of dimension $kz \times \nu$ where $kz \geq \mu$. When CSRF2 is called, the values $D_1^k D_2^\ell f(x_r, y_s)$ are computed and stored in Z for $r = 1, \ldots, \mu$ and $s = 1, \ldots, \nu$.

Programming. CSRF2 is a driver for the subroutine CSRF. The function INTRVL is used. CSRF2 was written by A. H. Morris.

BISPLINE UNDER TENSION SURFACE INTERPOLATION

Given $x_1 < \cdots < x_m$ and $y_1 < \cdots < y_n$. Also assume that we are given the values z_{ij} $(i = 1, \ldots, m; j = 1, \ldots, n)$ and a tension factor σ. Then the subroutine SURF is available for finding a bispline $F(x, y)$ with tension σ that satisfies $F(x_i, y_j) = z_{ij}$ for each i, j. Boundary conditions can be imposed on the surface $F(x, y)$ if desired.

CALL SURF$(m, n, X, Y, Z, kz,$ OPT,DDZ,WK,$\sigma,$ IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_n$, and Z the $m \times n$ matrix (z_{ij}). The argument kz is the number of rows in the dimension statement for Z in the calling program. It is assumed that $m \geq 2, n \geq 2$, and $kz \geq m$.

OPT is an array, called the ***option vector***, which permits the user to specify any boundary conditions that are to be imposed on the surface. If no boundary conditions are to be specified then OPT may be declared to be of dimension 1 and OPT(1) must be assigned the value 0. The details concerning the specification of boundary conditions in OPT are given below.

DDZ is a 3-dimensional array of dimension $m \times n \times 3$ and IERR is a variable. When SURF is called, if no input errors are detected then IERR is assigned the value 0 and the partial derivatives $D_1^2 F(x_i, y_j), D_2^2 F(x_i, y_j), D_1^2 D_2^2 F(x_i, y_j)$ $(i = 1, \ldots, m; j = 1, \ldots, n)$ are computed and stored in DDZ. DDZ$(i, j, 1) = D_2^2 F(x_i, y_j)$, DDZ$(i, j, 2) = D_1^2 F(x_i, y_j)$, and DDZ$(i, j, 3) = D_1^2 D_2^2 F(x_i, y_j)$ for each i, j.

WK is an array of dimension $m + 2n$ or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $m < 2$ or $n < 2$.
IERR = 2 if $x_1 < \cdots < x_m$ or $y_1 < \cdots < y_n$ is not satisfied.
IERR = 3 if OPT contains an error.

When an error is detected, the routine immediately terminates.

Remark. After DDZ is obtained, then the function SURF2 may be used to evaluate the bispline $F(x, y)$. Also, if $\sigma \neq 0$ then the subroutine CSRF2 may be used to evaluate or differentiate $F(x, y)$.

The option vector OPT. If no boundary conditions are to be imposed then OPT may be declared to be of dimension 1 and OPT(1) must have the value 0. Otherwise, OPT is an array containing the information key_1, data_1, key_2, data_2, $\ldots$, key_s, data_s, 0. The last entry in OPT is the value 0. Each group of data key_i, data_i $(i = 1, \ldots, s)$ is called an ***option***. Each key_i is an integer and data_i is a list of partial derivative values that the surface $F(x, y)$ is required to satisfy. The following options are available:

key = 1 The values $D_1 F(x_1, y_j)$ $(j = 1, \ldots, n)$ must be satisfied.
key = 2 The values $D_1 F(x_m, y_j)$ $(j = 1, \ldots, n)$ must be satisfied.
key = 3 The values $D_2 F(x_i, y_1)$ $(i = 1, \ldots, m)$ must be satisfied.

key = 4 The values $D_2F(x_i, y_n)$ $(i = 1, \ldots, m)$ must be satisfied.
key = 5 The value $D_1D_2F(x_1, y_1)$ must be satisfied.
key = 6 The value $D_1D_2F(x_m, y_1)$ must be satisfied.
key = 7 The value $D_1D_2F(x_1, y_n)$ must be satisfied.
key = 8 The value $D_1D_2F(x_m, y_n)$ must be satisfied.

The order of the options in OPT is arbitrary. If an unrecognized key is used then the error indicator IERR is assigned the value 3 and the routine terminates.

Example. Assume that we have an array DY1 containing values $D_2F(x_i, y_1)$ $(i = 1, \ldots, m)$ which are to be satisfied, and that we also want $D_1D_2F(x_m, y_n) = -1.3$ to be satisfied. Then OPT must be of dimension $\geq m + 4$ and OPT can be defined as follows:

```
      OPT(1) = 3.0            (First option)
      DO 10 I = 1,M
10    OPT (I + 1) = DY1(I)
      OPT (M + 2) = 8.0       (Second option)
      OPT (M + 3) = -1.3
      OPT (M + 4) = 0.0       (Terminates the option vector)
```

Background. The evaluation of $D_1^2F(x_i, y_j)$, $D_2^2F(x_i, y_j)$, and $D_1^2D_2^2F(x_i, y_j)$ reduce to the evaluation of second derivatives of splines. Specifically, for each $i \leq m$ $D_2^2F(x_i, y_1), \ldots, D_2^2F(x_i, y_n)$ are the second derivatives that characterize the spline $y \mapsto F(x_i, y)$, and for each $j \leq n$ $D_1^2F(x_1, y_j), \ldots, D_1^2F(x_m, y_j)$ are the second derivatives that characterize the spline $x \mapsto F(x, y_j)$. Also $D_1D_2^2F(x_1, y_j)$ and $D_1D_2^2F(x_m, y_j)$ $(j = 1, \ldots, n)$ are the second derivatives that characterize the splines $y \mapsto D_1F(x_1, y)$ and $y \mapsto D_1F(x_m, y)$. For each $j \leq m$, after one obtains the values $D_2^2F(x_i, y_j)$ through which the spline $x \mapsto D_2^2F(x, y_j)$ will pass and the end slopes $D_1D_2^2F(x_1, y_j)$ and $D_1D_2^2F(x_m, y_j)$ which this spline must have, then the second derivatives that characterize this spline can be computed. $D_1^2D_2^2F(x_1, y_j)$, $\ldots, D_1^2D_2^2F(x_m, y_j)$ are the second derivatives that characterize $x \mapsto D_2^2F(x, y_j)$.

Programming. SURF employs the subroutines CEEZ, TERMS, and SNHCSH. SURF was written by A. K. Cline and R. J. Renka (University of Texas at Austin), and modified by A. H. Morris.

BISPLINE UNDER TENSION EVALUATION

Given $x_1 < \cdots < x_m$ and $y_1 < \cdots < y_n$, and let $F(x,y)$ be a bispline with tension σ. If the partial derivatives $D_1^2F(x_i,y_j)$, $D_2^2F(x_i,y_j)$, $D_1^2D_2^2F(x_i,y_j)$ are known for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, then the function SURF2 may be used for evaluating $F(x,y)$ at a single point, and the subroutine NSURF2 may be used for evaluating $F(x,y)$ on a grid of points.

SURF2$(s,t,m,n,X,Y,Z,kz,\mathrm{DDZ},\sigma)$

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_n$, and Z an $m \times n$ matrix containing the values $F(x_i,y_j)$. The argument kz is the number of rows in the dimension statement for Z in the calling program. It is assumed that $m \geq 2$, $n \geq 2$, and $kz \geq m$.

DDZ is a 3-dimensional array of dimension $m \times n \times 3$ containing the partial derivatives where

$$\mathrm{DDZ}(i,j,1) = D_2^2F(x_i,y_j)$$
$$\mathrm{DDZ}(i,j,2) = D_1^2F(x_i,y_j)$$
$$\mathrm{DDZ}(i,j,3) = D_1^2D_2^2F(x_i,y_j)$$

for each i,j. $\mathrm{SURF2}(s,t,m,n,X,Y,Z,kz,\mathrm{DDZ},\sigma) = F(s,t)$ for any point (s,t).

Remark. After DDZ has been obtained, SURF2 may be repeatedly called to evaluate the bispline at different points so long as the tension factor σ remains fixed. However, if σ is modified then the derivative information in DDZ will have to be recomputed before SURF2 can be used with the new tension factor.

Programming. SURF2 employs the function INTRVL and subroutine SNHCSH. SURF2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL NSURF2$(s_{\min},s_{\max},m_s,t_{\min},t_{\max},n_t,W,kw,m,n,$
$X,Y,Z,kz,\mathrm{DDZ},\mathrm{WORK},\sigma)$

The arguments $s_{\min}$ and $s_{\max}$ are the lower and upper limits of the x-coordinates of the grid on which $F(x,y)$ is to be evaluated, and the arguments $t_{\min}$ and $t_{\max}$ are the lower and upper limits of the y-coordinates. The purpose of the routine is to evaluate the bispline at the points (s_i,t_j) where

$$s_i = s_{\min} + (i-1)\,\frac{s_{\max} - s_{\min}}{m_s - 1}$$
$$t_j = t_{\min} + (j-1)\,\frac{t_{\max} - t_{\min}}{n_t - 1}$$

for $i = 1, \ldots, m_s$ and $j = 1, \ldots, n_t$. It is assumed that $m_s \geq 1$ and $n_t \geq 1$.

W is a 2-dimensional array of dimension $kw \times n_t$ where $kw \geq m_s$. When NSURF2 is called $W(i,j)$ is assigned the value $F(s_i,t_j)$ for $i = 1, \ldots, m_s$ and $j = 1, \ldots, n_t$.

The arguments $m, n, X, Y, Z, kz, \mathrm{DDZ}, \sigma$ are the same as in SURF2. WORK is an array of dimension $4m_s$ or larger that is used for a work space.

Programming. NSURF2 employs the subroutine SNHCSH. NSURF2 was written by A. K. Cline (University of Texas at Austin).

BIVARIATE B-SPLINE PIECEWISE POLYNOMIAL INTERPOLATION

For $m \geq k \geq 1$ and $n \geq \ell \geq 1$ let $s_1 \leq \cdots \leq s_{m+k}$ and $t_1 \leq \ldots \leq t_{n+\ell}$ be sequences where $s_i < s_{i+k}$ $(i = 1, \ldots, m)$ and $t_j < t_{j+\ell}$ $(j = 1, \ldots, n)$. Let $B_{1k}, \ldots, B_{mk}$ be the B-splines for the knots $s_1 \leq \cdots \leq s_{m+k}$ and $B_{1\ell}, \ldots, B_{n\ell}$ the B-splines for the knots $t_1 \leq \cdots \leq t_{n+\ell}$. Given $s_k \leq x_1 < \cdots < x_m \leq s_{m+1}$, $t_\ell \leq y_1 < \cdots < y_n \leq t_{n+1}$, and the values z_{ij} $(i = 1, \ldots, m, j = 1, \ldots, n)$. Then we wish to find a piecewise polynomial of the form

$$f(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} B_{ik}(x) B_{j\ell}(y)$$

which satisfies $f(x_i, y_j) = z_{ij}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.[1] This problem has a unique solution when $s_i < x_i < s_{i+k}$ and $t_j < y_j < t_{j+\ell}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The following subroutine is available for obtaining the coefficients a_{ij} of this piecewise polynomial.

CALL BSTRP2$(X, Y, Z, kz, S, m, k, T, n, \ell, A, ka$,WK,NUM,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_n$, Z the $m \times n$ matrix (z_{ij}), S an array containing $s_1, \ldots, s_{m+k}$, and T an array containing $t_1, \ldots, t_{n+\ell}$. It is assumed that $kz \geq m$, where kz is the number of rows in the dimension statement for Z in the calling program.

A is a 2-dimensional array of dimension $ka \times n$ where $ka \geq m$, and IERR is a variable. When BSTRP2 is called, if no errors are detected then IERR is set to 0 and the desired coefficients a_{ij} are computed and stored in A.

WK is an array of dimension NUM where NUM $\geq mn + \max\{2km, 2\ell n\}$. WK is a work space for the routine.

Error Return. IERR has one of the following values if an input error is detected.

IERR = 1 $s_k \leq x_1 < \cdots < x_m \leq s_{m+1}$ or $s_i < x_i < s_{i+k}$ $(i = 1, \ldots, m)$ is violated.

IERR = 2 $t_\ell \leq y_1 < \cdots < y_n \leq t_{n+1}$ or $t_j < y_j < t_{j+\ell}$ $(j = 1, \ldots, n)$ is violated.

IERR = 3 NUM $< mn + \max\{2km, 2n\ell\}$.

Remarks.

(1) One may set $A = Z$, thereby replacing the data z_{ij} with a_{ij}.

(2) After the coefficients a_{ij} are obtained, then the subroutine BVAL2 can be used to evaluate or differentiate the piecewise polynomial $f(x, y)$.

Programming. BSTRP2 employs the subroutines BSTRP, BSTRP1, BSPVB, BANFAC, and BANSLV. BSTRP2 was written by A. H. Morris.

[1] If $x_m = s_{m+1}$ then by $f(x_m, y)$ we mean $f(x_m-, y)$, and if $y_n = t_{n+1}$ then by $f(x, y_n)$ we mean $f(x, y_n-)$.

BIVARIATE B-SPLINE PIECEWISE POLYNOMIAL LEAST SQUARES FITTING

For $m \geq k \geq 1$ and $n \geq \ell \geq 1$ let $s_1 \leq \cdots \leq s_{m+k}$ and $t_1 \leq \cdots \leq t_{n+\ell}$ be sequences where $s_i < s_{i+k}$ $(i = 1, \ldots, m)$ and $t_j < t_{j+\ell}$ $(j = 1, \ldots, n)$. Let $B_{1k}, \ldots, B_{mk}$ be the B-splines for the knots $s_1 \leq \cdots \leq s_{m+k}$ and $B_{1\ell}, \ldots, B_{n\ell}$ the B-splines for the knots $t_1 \leq \cdots \leq t_{n+\ell}$. Given $s_k \leq x_1 \leq \cdots \leq x_\mu \leq s_{m+1}$, $t_\ell \leq y_1 \leq \cdots \leq y_\nu \leq t_{n+1}$, and the values z_{rs} $(r = 1, \ldots, \mu,\ s = 1, \ldots, \nu)$ where $\mu \geq \max\{2, k\}$ and $\nu \geq \max\{2, \ell\}$. Also let $w_1, \ldots, w_\mu$ and $\overline{w}_1, \ldots, \overline{w}_\nu$ be positive weights. Then the following subroutine is available for finding a piecewise polynomial of the form[1]

$$f(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} B_{ik}(x) B_{j\ell}(y)$$

which minimizes $\sum_{r=1}^{\mu} \sum_{s=1}^{\nu} w_r \overline{w}_s [f(x_r, y_s) - z_{rs}]^2$.

CALL BSLSQ2(X,WX,μ,Y,WY,ν,Z,kz,S,m,k,T,n,ℓ,
A,ka,WK,NUM,IERR)

X is an array containing $x_1, \ldots, x_\mu$, WX an array containing $w_1, \ldots, w_\mu$, Y an array containing $y_1, \ldots, y_\nu$, WY an array containing $\overline{w}_1, \ldots, \overline{w}_\nu$, Z the $\mu \times \nu$ matrix (z_{rs}), S an array containing $s_1, \ldots, s_{m+k}$, and T an array containing $t_1, \ldots, t_{n+\ell}$. It is assumed that $kz \geq \mu$, where kz is the number of rows in the dimension statement for Z in the calling program.

A is a 2-dimensional array of dimension $ka \times n$ where $ka \geq m$. When BSLSQ2 is called, if no input errors are detected then the coefficients a_{ij} of a weighted least squares piecewise polynomial approximation $f(x, y)$ are computed and stored in A.

WK is an array of dimension NUM where NUM $\geq m\nu + \max\{(k+1)m, (\ell+1)n\}$. WK is a work space for the routine.

IERR is a variable that reports the status of the results. When BSLSQ2 terminates, IERR has one of the following values:

IERR = 0 The coefficients were obtained. The least squares approximation is unique.

IERR = 1 The coefficients for a least squares approximation were obtained. The approximation is not unique.

IERR = −1 (Input error) Either $s_k < x_1 \leq \cdots \leq x_\mu \leq s_{m+1}$ or $\mu \geq \max\{2, k\}$ is violated.

IERR = −2 (Input error) Either $t_\ell \leq y_1 \leq \cdots \leq y_\nu \leq t_{n+1}$ or $\nu \geq \max\{2, \ell\}$ is violated.

IERR = −3 (Input error) NUM is too small. The minimum permissible value

[1] If $x_\mu = s_{m+1}$ then by $f(x_\mu, y)$ we mean $f(x_\mu{}^-, y)$, and if $y_\nu = t_{n+1}$ then by $f(x, y_\nu)$ we mean $f(x, y_\nu{}^-)$.

for NUM, namely $m\nu + \max\{(k+1)m, (\ell+1)n\}$, has been stored in WK(1). Reset NUM $\geq$ WK(1).

Remark. After the coefficients $a_{i,j}$ are obtained, then the subroutine BVAL2 can be used to evaluate or differentiate the piecewise polynomial $f(x, y)$.

Programming. BSLSQ2 employs the routines BSLSQ1, BSPVB, BCHFAC, and BCHSV1. BSLSQ2 was written by A. H. Morris.

EVALUATION AND DIFFERENTIATION OF BIVARIATE PIECEWISE POLYNOMIALS FROM THEIR B-SPLINE REPRESENTATIONS

For $m \geq k \geq 1$ and $n \geq \ell \geq 1$ let $s_1 \leq \cdots \leq s_{m+k}$ and $t_1 \leq \cdots \leq t_{n+\ell}$ be sequences where $s_i < s_{i+k}$ $(i = 1, \ldots, m)$ and $t_j < t_{j+\ell}$ $(j = 1, \ldots, n)$. Let $B_{1k}, \ldots, B_{mk}$ be the B-splines for the knots $s_1 \leq \cdots \leq s_{m+k}$ and $B_{1\ell}, \ldots, B_{n\ell}$ the B-splines for the knots $t_1 \leq \cdots \leq t_{n+\ell}$. If

$$f(x,y) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} B_{ik}(x) B_{j\ell}(y)$$

then the following subroutine is available for computing $f(x,y)$ and its partial derivatives.

CALL BVAL2$(A, ka, S, m, k, T, n, \ell, x, y, \mu, \nu, w, \mathrm{WK})$

A is the $m \times n$ matrix (a_{ij}), S an array containing $s_1, \ldots, s_{m+k}$, and T an array containing $t_1, \ldots, t_{n+\ell}$. It is assumed that $ka \geq m$, where ka is the number of rows in the dimension statement for A in the calling program.

The arguments μ and ν are nonnegative integers, (x,y) is the point at which f or its partial derivative $\partial^{\mu+\nu} f / \partial^{\mu} x \partial^{\nu} y$ is to be evaluated, and w is a variable. When BVAL2 is called, then w is assigned the value $\sum a_{ij} B_{ik}^{(\mu)}(x) B_{j\ell}^{(\nu)}(y)$. Here $B_{ik}^{(\mu)}$ denotes B_{ik} if $\mu = 0$ and the μ^{th} derivative of B_{ik} if $\mu \geq 1$. If $x \neq s_{m+k}$ then by $B_{ik}^{(\mu)}(x)$ we mean the right limit $B_{ik}^{(\mu)}(x+)$, and if $x = s_{m+k}$ then by $B_{ik}^{(\mu)}(x)$ we mean the left limit $B_{ik}^{(\mu)}(x-)$. Similar comments hold for $B_{j\ell}^{(\nu)}(y)$.

WK is an array of dimension $\ell + \max\{3k, 3\ell\}$ or larger that is a work space for the routine.

Programming. BVAL2 employs the function INTRVL and subroutine BVAL. BVAL2 was written by A. H. Morris.

SURFACE INTERPOLATION FOR ARBITRARILY POSITIONED DATA POINTS

Let $\{ (x_i, y_i) : i = 1, \ldots, n \}$ be a set of $n \geq 3$ distinct points which are not collinear and $z_1, \ldots, z_n$ a sequence of values. Then the problem is to find a smooth mapping $z = F(x, y)$ for which $z_i = F(x_i, y_i)$ for $i = 1, \ldots, n$.

A triangle based procedure for constructing a smooth interpolating mapping $z = F(x, y)$ contains the following components:

(1) An algorithm for forming a triangular grid for the convex hull of the data points (x_i, y_i) and a procedure for locating any point in the grid. If extrapolation is to be permitted, then the region outside the grid must also be partitioned and a procedure provided for locating any point in the region.

(2) A procedure for estimating the first (and possibly higher order) partial derivatives of $F(x, y)$ at the data points (x_i, y_i).

(3) A smooth interpolating routine for computing $F(x, y)$ on each triangular cell of the grid. If extrapolation is to be permitted, then a routine must also be provided for computing $F(x, y)$ on each cell of the partitioned region outside of the grid.

With regard to (2), it should be noted that no derivative estimation procedure is known that is appropriate for all applications. This is unfortunate, since the derivative values can significantly affect the results obtained from any interpolating procedure.

The subroutine TRMESH is available for obtaining a triangular grid, and the subroutines GRADG and GRADL are available for derivative estimation (performed in two different manners). After the grid has been obtained and the derivatives estimated, then the subroutines SFVAL and SFVAL2 are available for computing a continuously differentiable mapping $z = F(x, y)$ for which $z_i = F(x_i, y_i)$ for $i = 1, \ldots, n$. Extrapolation is permitted.

CALL TRMESH(n, X, Y,IADJ,IEND,IERR)

X is an array containing $x_1, \ldots, x_n$ and Y an array containing $y_1, \ldots, y_n$. IADJ is an array of dimension $6n$ or larger, IEND an array of dimension n, and IERR a variable. When TRMESH is called, if no input errors are detected then IERR is set to 0 and a Thiessen triangulation of the convex hull is obtained. The triangulation is stored in IADJ and IEND.

Error Return. IERR = 1 if $n < 3$ and IERR = 2 if the data points (x_i, y_i) are collinear.

Remark. For efficiency, it is recommended that the data points (x_i, y_i) be ordered so that $x_1 \leq \cdots \leq x_n$.

Programming. TRMESH employs the routines ADNODE, BDYADD, INTADD, SHIFTD, SWAPD, TRFIND and functions SWPTST and TINDX. These routines were written by Robert J. Renka (University of North Texas, Denton, Texas).

References.

(1) Cline, A. K. and Renka, R. J., "A Storage-Efficient Method for Construction of a Thiessen Triangulation," ***Rocky Mountain J. Math*** **14** (1984), pp. 119–139.

(2) Renka, R. J., "Algorithm 624. Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane," ***ACM Trans. Math Software*** **10** (1984), pp. 440–442.

CALL GRADG(n, X, Y, Z,IADJ,IEND,DZ,IERR)
CALL GRADL(n, X, Y, Z,IADJ,IEND,DZ,IERR)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and Z an array containing $z_1, \ldots, z_n$. IADJ is an array of dimension $6n$ or larger, and IEND an array of dimension n. IADJ and IEND contain the grid information given by the routine TRMESH.

DZ is a 2-dimensional array of dimension $2 \times n$. When GRADG or GRADL is called, if the partial derivatives of $F(x, y)$ are computed at the data points then the derivatives $D_1F(x_1, y_1), \ldots, D_1F(x_n, y_n)$ are stored in the first row of DZ and the derivatives $D_2F(x_1, y_1), \ldots, D_2F(x_n, y_n)$ in the second row of DZ.

Error Return. IERR is a variable that reports the status of the results. When the subroutine terminates IERR has one of the following values:

IERR = 0 The partial derivatives of F were evaluated and stored in DZ.
IERR = 1 (Input Error) $n < 3$.
IERR = 2 The partial derivatives could not be estimated by GRADL. Either the data points are almost collinear, or the abscissas and ordinates of the data points are too poorly scaled for the derivatives to be computed (see remark (2) below).

Algorithm. GRADG employs the global derivative estimation procedure and GRADL the local derivative estimation procedure given in reference (1).

Remarks.

(1) GRADG is faster than GRADL, but it frequently is less accurate.

(2) The local derivative estimation procedure can be quite sensitive to the scaling of the abscissas x_i and ordinates y_i of the data points. Frequently, rescaling the abscissas and ordinates will have no effect on the accuracy of the results obtained by GRADG, but it may increase or decrease the accuracy of the results obtained by GRADL by several decimal digits (or make it impossible to estimate the derivatives locally).

Programming. GRADG and GRADL are interface routines for the subroutines GRADG1 and GRADL1, written by Robert J. Renka (University of North Texas, Denton, Texas). GRADL1 calls the subroutines GETNP, SETRM, SROT, and SROTG.

References.

(1) Renka, R. J., and Cline, A. K., "A Triangle Based C^1 Interpolation Method," ***Rocky Mountain J. Math*** **14** (1984), pp. 223–237.

(2) Renka, R. J., "Algorithm 624. Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane," ***ACM Trans. Math Software*** **10** (1984), pp. 440–442.

CALL SFVAL$(n, X, Y, Z, m, XI, YI, ZI$,IADJ,IEND,DZ,IERR)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and Z an array containing $z_1, \ldots, z_n$. IADJ is an array of dimension $6n$ or larger, IEND an array of dimension n, and DZ an array of dimension $2 \times n$. It is assumed that IADJ and IEND contain the triangulation given by the subroutine TRMESH, and that DZ contains the partial derivatives computed by GRALG or GRADL.

It is assumed that $F(x, y)$ is to be evaluated at $(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_m, \bar{y}_m)$. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array containing $\bar{y}_1, \ldots, \bar{y}_m$, and ZI an array of dimension m or larger. IERR is a variable. When SFVAL is called, if no errors are detected then IERR is set to 0. Also $F(\bar{x}_i, \bar{y}_i)$ is computed and stored in $ZI(i)$ for $i = 1, \ldots, m$.

Error Return. IERR $= 1$ if $n < 3$ or $m < 1$, and IERR $= 2$ if the data points (x_i, y_i) $(i = 1, \ldots, n)$ are collinear.

Programming. SFVAL is a driver for the subroutine INTRC1. INTRC1 employs the subroutines GRADL1, GETNP, SETRM, SROT, SROTG, TRFIND, and TVAL. INTRC1 was written by Robert J. Renka (University of North Texas, Denton, Texas).

References.

(1) Renka, R. J., and Cline, A. K., "A Triangle Based C^1 Interpolation Method," ***Rocky Mountain J. Math*** **14** (1984), pp. 223–237.

(2) Renka, R. J., "Algorithm 624. Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane," ***ACM Trans. Math Software*** **10** (1984), pp. 440–442.

CALL SFVAL2$(n, X, Y, Z, \ell, m, kz, XI, YI, ZI$,IADJ,IEND,DZ,IERR)

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and Z an array containing $z_1, \ldots, z_n$. IADJ is an array of dimension $6n$ or larger, IEND an array of dimension n, and DZ an array of dimension $2 \times n$. It is assumed that IADJ and IEND contain the triangulation given by the subroutine TRMESH, and that DZ contains the partial derivatives computed by GRADG or GRADL.

It is assumed that $F(x, y)$ is to be evaluated at (x_i, y_j) for $i = 1, \ldots, \ell$ and $j = 1, \ldots, m$. XI is an array containing $x_1, \ldots, x_\ell$, YI an array containing $y_1, \ldots, y_m$, and ZI a two-dimensional array of dimension $kz \times m$ where $kz \geq \ell$. IERR is a variable. When SFVAL2 is called, if no errors are detected then IERR is set to 0. Also $F(x_i, y_j)$ is computed and stored in $ZI(i, j)$ for $i = 1, \ldots, \ell$ and $j = 1, \ldots, m$.

Error Return. IERR $= 1$ if $n < 3$, $\ell < 1$, $m < 1$, or $kz < \ell$, and IERR $= 2$ if the data points (x_i, y_i) $(i = 1, \ldots, n)$ are collinear.

Programming. SFVAL2 is a driver for the subroutine INTRC1. INTRC1 employs the subroutines GRADL1, GETNP, SETRM, SROT, SROTG, TRFIND, and TVAL. INTRC1 was written by Robert J. Renka (University of North Texas, Denton, Texas)

WEIGHTED LEAST SQUARES FITTING WITH POLYNOMIALS OF N VARIABLES

Let $\{(x_1^{(i)}, \ldots, x_n^{(i)}) : i = 1, \ldots, \ell\}$ be a set of ℓ distinct points, $z_1, \ldots, z_\ell$ be the corresponding function values to be approximated, and $w_1, \ldots, w_\ell$ be positive weights. Then for any nonnegative integer IDEG where $\binom{n+\mathrm{IDEG}}{n} \le \ell$,[1] the subroutines MFIT and DMFIT are available for obtaining the coefficients of the unique polynomial $F(x_1, \ldots, x_n)$ of degree IDEG which minimizes $\sum_{i=1}^{\ell} w_i[F(x_1^{(i)}, \ldots, x_n^{(i)}) - z_i]^2$. Also, the subroutines MEVAL and DMEVAL are available for computing this polynomial. MFIT and MEVAL yield single precision results, and DMFIT and DMEVAL yield double precision results.

CALL MFIT(n, IDEG, m, ℓ, X, kx, Z, W, R, IER, IWK, WK, LIWK, LWK, MIWK, MWK)
CALL DMFIT(n, IDEG, m, ℓ, X, kx, Z, W, R, IER, IWK, WK, LIWK, LWK, MIWK, MWK)

It is assumed that $n \ge 1$ and $\ell \ge 1$. X is an $\ell \times n$ matrix whose i^{th} row contains the point $(x_1^{(i)}, \ldots, x_n^{(i)})$; i.e., $X(i,j) = x_j^{(i)}$ for $i = 1, \ldots, \ell$ and $j = 1, \ldots, n$. The argument kx is the number of rows in the dimension statement for X in the calling program. Z is an array containing $z_1, \ldots, z_\ell$ and W an array containing $w_1, \ldots, w_\ell$. X and Z are modified by the routine.

Remark. For IDEG > 0, $\binom{n+\mathrm{IDEG}}{n}$ polynomials $1, x_1, \ldots, x_n, x_1^2, x_1x_2, \ldots$ are needed for a basis of the space of polynomials of degree $\le$ IDEG. The basis polynomials are ordered. For $k \ge 1$, the degree $k-1$ basis polynomials precede the degree k polynomials. The degree k basis polynomials are $x_{i_1} \cdots x_{i_k}$ where $1 \le i_1 \le \cdots \le i_k \le n$. For any two such polynomials $x_{i_1} \cdots x_{i_k}$ and $x_{j_1} \cdots x_{j_k}$, let r be the smallest integer such that $i_r \ne j_r$. Then $x_{i_1} \cdots x_{i_k}$ precedes $x_{j_1} \cdots x_{j_k}$ when $i_r < j_r$.

IDEG and m are variables. If IDEG > 0 then the routine attempts to obtain the polynomial $F(x_1, \ldots, x_n)$ of degree IDEG which is the best least squares fit. Otherwise, if IDEG ≤ 0 then it is assumed that $m \ge 1$ and that the first m basis polynomials are to be used to obtain the least squares fit. When the routine terminates, IDEG = the degree of the polynomial $F(x_1, \ldots, x_n)$ obtained and m = the number of basis polynomials that are actually used.

R is an array of dimension ℓ or larger. $R(i) = z_i - F(x_1^{(i)}, \ldots, x_n^{(i)})$ for $i = 1, \ldots, \ell$ when the routine terminates.

IWK is an array of dimension LIWK and WK an array of dimension LWK. When the routine terminates, IWK and WK contain the information needed for computing the polynomial $F(x_1, \ldots, x_n)$. Sufficient storage for IWK and WK can be assured by setting LIWK and LWK as follows. If IDEG $= 0$ then let $N = \min\{\ell, \binom{n+\mathrm{IDEG}}{n}\}$ and $\delta = \mathrm{IDEG}$.

[1] $\binom{k}{0} = 1$ and $\binom{k}{r} = \frac{k(k-1)\cdots(k-r+1)}{r!}$ for $r = 1, 2, \ldots$.

Otherwise, if IDEG < 0 then let $N = m$ and δ be the smallest nonnegative integer such that $\binom{n+\delta}{n} \geq m$. Then set

$$\text{LIWK} \geq 4N + \delta n$$

$$\text{LWK} \geq 2N + n + 1 + \ell N + \frac{1}{2}(N-1)(N-2).$$

N is the maximum number of basis polynomials that will be used, and δ is the degree of the polynomial $F(x_1, \ldots, x_n)$ if N basis polynomials are used.

MIWK and MWK are variables. MIWK is set by the routine to the dimension needed for IWK, and MWK is set to the dimension needed for WK. MIWK and MWK depend only on n, ℓ,IDEG, and m.

If MFIT is called then X, Z, W, R, and WK are single precision real arrays. Otherwise, if DMFIT is called then X, Z, W, R, and WK are double precision arrays.

IER is a variable that reports the status of the results. When the routine terminates, IER has one of the following values:

IER = 0 The desired polynomial was obtained.

IER = −1 Not all the basis polynomials could be used. IDEG is the degree of the polynomial obtained. This setting occurs when the problem is not solvable or is too ill-conditioned for the requested degree.

IER = 1 Only ℓ basis polynomials were used. A polynomial $F(x_1, \ldots, x_n)$ was obtained which solves the equations $F(x_1^{(i)}, \ldots, x_n^{(i)}) = z_i$ for $i = 1, \ldots, \ell$.

IER = 2 (Input error) IDEG < 0 and $m \leq 0$.

IER = 3 (Input error) $n < 1$ or $\ell < 1$.

IER = 4 (Input error) LIWK or LWK is too small. Set LIWK > MIWK and LWK > MWK.

When an input error is detected, the routine immediately terminates.

Remark. When IER ≥ -1 then MEVAL or DMEVAL may be used to compute the polynomial obtained.

Algorithm. The revised Gram-Schmidt orthogonalization procedure is used.

Programming. MFIT employs the routines ALLOT, BASIZ, MTABLE, GNRTP, INCDG, SCALPM,SCALDN,and DMFIT employs the routines ALLOT,BASIZ,MTABLE,DGNRTP, DINCDG, DSCALP, DSCALD. MFIT and DMFIT are modifications by A. H. Morris of CONSTR, written by Richard H. Bartels (University of Waterloo) and John J. Jezioranski (Ontario Cancer Institute).

References.

(1) Bartels, R. H. and Jezioranski, J. J., "Least Squares Fitting using Orthogonal Multinomials," *ACM Trans. Math Software* 11 (1985), pp. 201–217.

(2) ———, "Algorithm 634, CONSTR and EVAL: Routines for Fitting Multinomials in a Least Squares Sense," *ACM Trans. Math Software* 11 (1985), pp. 218–228.

CALL MEVAL$(n, \text{KDEG}, \overline{m}, \bar{l}, \text{XI}, kxi, \text{ZI}, \text{IND}, \text{IWK}, \text{WK}, \text{LIWK}, \text{LWK}, \text{T})$
CALL DMEVAL$(n, \text{KDEG}, \overline{m}, \bar{l}, \text{XI}, kxi, \text{ZI}, \text{IND}, \text{IWK}, \text{WK}, \text{LIWK}, \text{LWK}, \text{T})$

MEVAL (DMEVAL) computes the polynomial obtained by MFIT (DMFIT), or a portion thereof. Let IDEG and m be the output values given by MFIT (DMFIT).

The argument $\overline{m}$ is a variable. If KDEG < 0 then it is assumed that $1 \leq \overline{m} \leq m$ and that the polynomial using the first $\overline{m}$ basis polynomials is to be computed. In this case, the polynomial computed is the best least squares fit for the basis polynomials involved.

If KDEG ≥ 0 then it is assumed that KDEG $\leq$ IDEG. In this case, when the routine terminates, $\overline{m}$ = the number of basis polynomials used. If $\overline{m} \leq m$ (which will be the case when KDEG $<$ IDEG), then the polynomial computed is the polynomial of degree KDEG which is the best least squares fit.

Usage. If IER $= \pm 1$ when MFIT(DMFIT) terminates, then the setting KDEG = IDEG normally causes an error to occur since $\overline{m} > m$. Hence, if it is desired that the full polynomial obtained by MFIT(DMFIT) be computed, no matter whether the value for IER is 0 or ± 1, then KDEG should be assigned a negative value and $\overline{m} = m$.

It is assumed that the polynomial is to be computed at the points $(\overline{x}_1^{(i)}, \ldots, \overline{x}_n^{(i)})$ for $i = 1, \ldots, \bar{\ell}$. XI is an $\bar{\ell} \times n$ matrix whose i^{th} row contains the point $(\overline{x}_1^{(i)}, \ldots, \overline{x}_n^{(i)})$. The argument kxi is the number of rows in the dimension statement for XI in the calling program. ZI is an array of dimension $\bar{\ell}$ or larger. When the routine terminates, ZI(i) contains the value of the polynomial at the point $(\overline{x}_1^{(i)}, \ldots, \overline{x}_n^{(i)})$ for $i = 1, \ldots, \bar{\ell}$.

IWK and WK are the arrays obtained from MFIT or DMFIT. LIWK is the dimension of IWK and LWK the dimension of WK. T is an array of dimension n or larger that is a work space for the routine.

If MEVAL is called then XI, ZI, WK and T are single precision arrays. Otherwise, if DMEVAL is called then XI, ZI, WK and T are double precision arrays.

IND is a variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0 The desired computation was performed.
IND = −1 (Input error) $\overline{m} < 1$ or $\overline{m} > m$.
IND = −2 (Input error) $n < 1$ or $\bar{\ell} < 1$.

Programming. MEVAL calls the subroutine MEVAL1 and DMEVAL calls the subroutine DMEVL1. MEVAL and DMEVAL are modifications by A. H. Morris of EVAL, written by Richard H. Bartels (University of Waterloo) and John J. Jezioranski (Ontario Cancer Institute).

EVALUATION OF INTEGRALS OVER FINITE INTERVALS

QAGS, QXGS, QSUBA, DQAGS, and DQXGS are available for computing integrals $\int_a^b F(x)\,dx$ over finite intervals. The subroutines QAGS and QXGS and the function QSUBA yield single precision results. These procedures are adaptive. In such procedures, the selection of the points at which the integrand is evaluated depends on the nature of the integrand. DQAGS and DQXGS are double precision versions of the subroutines QAGS and QXGS.

QSUBA is applicable only when F and its derivatives have no singularities in the closed interval $[a, b]$. Otherwise, QAGS or QXGS should be used. These subroutines are appropriate when $F(x)$ is continuous except possibly for a finite number of singularities. QAGS and QXGS generally yield accurate results when any singularities which exist are located at the endpoints of the interval $[a, b]$. However, if F has an exceedingly narrow spike which contributes significantly to the value of the integral, then the subroutines may overlook the spike and produce incorrect results. QAGS and QXGS normally have approximately the same accuracy. QAGS uses less storage than QXGS, but it frequently requires considerably more function evaluations than QXGS. Consequently, QXGS is recommended when the integrand $F(x)$ is expensive to compute.

CALL QAGS(F, a, b,AERR,RERR,z,ERROR,NUM,IERR,ℓ, m, n,IWK,WK)

$F(x)$ is a user defined function whose arguments and values are assumed to be real numbers. The purpose of QAGS is to compute the integral $\int_a^b F(x)\,dx$. F need not be defined at a and b, and it is not required that $a \leq b$. F must be declared in the calling program to be of type EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used, and z is a variable. When QAGS is called, z is assigned the value obtained for $I = \int_a^b F(x)\,dx$. The subroutine attempts to obtain a value z which satisfies $|I - z| \leq \max\{\text{AERR}, \text{RERR} \cdot |I|\}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR and NUM are variables. When QAGS terminates, ERROR is the estimated absolute error of z and NUM the number of points at which F was evaluated.

IWK is an array of dimension ℓ and WK an array of dimension m. IWK and WK are work spaces for the routine. The input argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$ and $m \geq 4\ell$. The argument n is a variable. When QAGS terminated, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0 The routine is satisfied that the integral has been computed to the desired accuracy.

IERR = 1 The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.

IERR = 2 The integral has been computed, but because of roundoff error QAGS is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.

IERR = 3 Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.

IERR = 4 The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.

IERR = 5 The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.

IERR = 6 (Input Error) Either $\ell < 1$, $m < 4\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, NUM, and n are set to 0.

Remark. NUM $\leq 42(\ell - 1) + 21$

Algorithm. The 21 point Kronrod rule and ϵ-algorithm of P. Wynn are used.

Programming. QAGS employs the subroutines QAGSE, QK21F, QPSRT, and QELG. These routines were developed by Robert Piessens and Elise de Doncker-Kapenga (Katholiek Universiteit Leuven, Belgium), and modified by A. H. Morris and Los Alamos National Laboratory. The function SPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK*: *A Subroutine Package for Automatic Integration***, Springer-Verlag, 1983.

CALL QXGS(F, a, b,AERR,RERR,z,ERROR,IERR,ℓ,μ,m,n,IWK,WK)

F is a user defined function whose arguments and values are assumed to be real numbers. The purpose of QXGS is to compute the integral $\int_a^b F(x)\,dx$. F must be defined at a and b, but it is not required that $a \leq b$. F must be declared in the calling program to be of type EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used, and z is a variable. When QXGS is called, z is assigned the value obtained for $I = \int_a^b F(x)\,dx$. The subroutine attempts to obtain a value z which satisfies $|I - z| \leq \max\{\text{AERR}, \text{RERR} \cdot |I|\}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR is a variable. When QXGS terminates, ERROR is the estimated absolute error of z.

IWK is an array of dimension μ and WK an array of dimension m. IWK and WK are work spaces for the routine. The input argument ℓ is the maximum number of subintervals

in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$, $\mu \geq 3\ell$, and $m \geq 46\ell$. The argument n is a variable. When QXGS terminates, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values;

IERR = 0 The routine has satisfied that the integral has been computed to the desired accuracy.

IERR = 1 The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.

IERR = 2 The integral has been computed, but because of roundoff error QXGS is not certain of the accuracy of the result. The error may be greater than is reported by ERROR.

IERR = 3 Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.

IERR = 4 The algorithm did not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.

IERR = 5 The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.

IERR = 6 (Input Error). Either $\ell < 1$, $\mu < 3\ell$, $m < 46\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, and n are set to 0.

Remarks.

(1) If F is unbounded at a or b, then set $F(a)$ or $F(b)$ to a dummy value, say 0.

(2) The first 4ℓ locations of WK serve the same purpose for QXGS as for QAGS. Functional values are stored in the remaining 42ℓ locations of WK.

Algorithm. The recursive monotone quadrature formulas of P. Favati, G. Lotti, and F. Romani, and the ϵ-algorithm of P. Wynn are used.

Programming. QXGS employs the subroutines QXGSE, QXCPY, QXLQM, QXRUL, QXRRD, QPSRT, and QELG. QXGS is an adaptation of QAGS where the Gauss-Kronrod formulae have been replaced with the recursive monotone formulae. QXGS was designed by Paola Favati (Instituto di Elaborazione dell'Informazione, CNR, Pisa), Grazia Lotti, and Francesco Romani (Universita di Pisa, Italy), and modified by A. H. Morris. The function SPMPAR is also used.

Reference. Favati, P., Lotti, G., and Romani, F., "Algorithm 691. Improving QUADPACK Automatic Integration Routines," *ACM Trans. Math Software* **17** (1991), pp. 218-232

QSUBA$(F, a, b$,RERR,MCOUNT,ERROR,IERR)

$F(x)$ is a user defined function whose arguments and values are assumed to be real

numbers. The purpose of QSUBA is to compute the integral $\int_a^b F(x)\,dx$. F need not be defined at the points a and b. However, it is assumed that $a \leq b$. F must be declared in the calling program to be of type EXTERNAL.

RERR is the relative error tolerance to be satisfied. It is assumed that RERR > 0. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$.

The input argument MCOUNT is the maximum number of points at which F may be evaluated. It is recommended that MCOUNT ≥ 1000.

ERROR is a variable that is set by QSUBA. If the value of QSUBA is not 0 then ERROR is an estimate of the relative error of the computed result. Otherwise, if the value of QSUBA is 0 then ERROR is an estimate of the absolute error.

IERR is a variable that reports the status of the results. When QSUBA terminates, IERR has one of the following values:

IERR = 0 The function QSUBA is satisfied that the integral has been computed to the desired accuracy.

IERR = 1 The integral has been computed, but QSUBA is not certain of the accuracy of the result.

IERR = 2 $F(x)$ was evaluated at MCOUNT points. More evaluations are needed to complete the computation of the integral.

IERR = 3 The function QSUBA cannot compute the integral to the desired accuracy.

If IERR = 0 or 1 then the function QSUBA is assigned the value obtained for the integral. If IERR = 2 then QSUBA has for its value the most recent acceptable partial estimate made of the integral. Otherwise, if IERR = 3, then QSUBA has for its value the best estimate of the value of the integral that it can make.

Remark. QSUBA assumes that F and its derivatives have no singularities in the closed interval $[a, b]$. If this is not the case then QAGS or QXGS should be used.

Algorithm. Gaussian quadrature is employed.

Programming. QSUBA calls the subroutine QUAD. QSUBA and QUAD were written by T. N. L. Patterson (Queen's University, Belfast, Northern Ireland), and QSUBA was modified by A. H. Morris. The function SPMPAR is used.

Reference. Patterson, T. N. L.,"Algorithm for Automatic Numerical Integration Over a Finite Interval," *Comm. ACM* **16** (1973), pp. 694-699.

CALL DQAGS(F, a, b,AERR,RERR,z,ERROR,NUM,IERR,ℓ, m, n,IWK,WK)

$F(x)$ is a user defined function whose arguments and values are assumed to be double precision numbers. The purpose of DQAGS is to compute the integral $\int_a^b F(x)\,dx$. The arguments a and b are double precision numbers. F need not be defined at a and b, and

it is not required that $a \leq b$. F must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL.

AERR and RERR are double precision numbers and z a double precision variable. AERR and RERR are the absolute and relative error tolerances to be used. When DQAGS is called, z is assigned the value obtained for $I = \int_a^b F(\tau)\,dx$. The subroutine attempts to obtain a value z which satisfies $|I - z| \leq \max\{\text{AERR}, \text{RERR} \cdot |I|\}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR is a double precision variable and NUM an integer variable. When DQAGS terminates, ERROR is the estimated absolute error of z and NUM the number of points at which F was evaluated.

IWK is an integer array of dimension ℓ and WK a double precision array of dimension m. IWK and WK are work spaces for the routine. The argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$ and $m \geq 4\ell$. the argument n is a variable. When DQAGS terminates, $n =$ the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR $= 0$ The routine is satisfied that the integral has been computed to the desired accuracy.

IERR $= 1$ The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.

IERR $= 2$ The integral has been computed, but because of roundoff error DQAGS is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.

IERR $= 3$ Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.

IERR $= 4$ The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.

IERR $= 5$ The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.

IERR $= 6$ (Input Error) Either $\ell < 1$, $m < 4\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, NUM, and n are set to 0.

Remark. NUM $\leq 42(\ell - 1) + 21$.

Algorithm. The 21 point Kronrod rule and ϵ-algorithm of P. Wynn are used.

Programming. DQAGS employs the routines DQAGSE, DQK21, DQPSRT, and DQELG. These subroutines are double precision versions by A. H. Morris of the routines QAGSE, QK21F, QPSRT, and QELG. The function DPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK*: *A Subroutine Package for Automatic Integration***, Springer-Verlag, 1983.

CALL DQXGS(F, a, b,AERR,RERR,z,ERROR,IERR,ℓ, μ, m, n,IWK,WK)

F is a user defined function whose arguments and values are assumed to be double precision numbers. The purpose of DQXGS is to compute the integral $\int_a^b F(x)\,dx$. The arguments a and b are double precision numbers. F must be defined at a and b, but it is not required that $a \leq b$. F must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL.

AERR and RERR are double precision numbers and z a double precision variable. AERR and RERR are the absolute and relative error tolerances to be used. When DQXGS is called, z is assigned the value obtained for $I = \int_a^b F(x)\,dx$. The subroutine attempts to obtain a value z which satisfies $|I - z| \leq \max\{\text{AERR}, \text{RERR} \cdot |I|\}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR is a double precision variable. When DXQGS terminates, ERROR is the estimated absolute error of z.

IWK is an integer array of dimension μ and WK a double precision array of dimension m. IWK and WK are work spaces for the routine. The argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$, $\mu \geq 3\ell$, and $m \geq 46\ell$. The argument n is a variable. When DQXGS terminates, $n =$ the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

- IERR $= 0$ The routine is satisfied that the integral has been computed to the desired accuracy.
- IERR $= 1$ The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
- IERR $= 2$ The integral has been computed, but because of roundoff error DQXGS is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
- IERR $= 3$ Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.
- IERR $= 4$ The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.
- IERR $= 5$ The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.
- IERR $= 6$ (Input Error). Either $\ell < 1$, $\mu < 3\ell$, $m < 46\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, and n are set to 0.

Remarks.

(1) If F is unbounded at a or b, then set $F(a)$ or $F(b)$ to a dummy value, say 0.
(2) The first 4ℓ locations of WK serve the same purpose for DQXGS as for DQAGS. Functional values are stored in the remaining 42ℓ locations of WK.

Algorithm. The recursive monotone quadrature formulas of P. Favati, G. Lotti, and F. Romani, and the ϵ-algorithm of P. Wynn are used.

Programming. DQXGS employs the routines DQXGSE, DQXCPY, DQXLQM, DQXRUL, DQXRRD, DQPSRT, and DQELG. DQXGS is an adaptation of DQAGS where the Gauss–Kronrod formulae have been replaced with the recursive monotone formulae. DQXGS was designed by Paola Favati (Instituto di Elaborazione dell'Informazione, CNR, Pisa), Grazia Lotti, and Francesco Romani (Universita di Pisa, Italy) and modified by A. H. Morris. The function DPMPAR is also used.

Reference. Favati, P., Lotti, G., and Romani, F., "Algorithm 691. Improving QUADPACK Automatic Integration Routines," ***ACM Trans. Math Software*** **17** (1991), pp. 218–232.

EVALUATION OF INTEGRALS OVER INFINITE INTERVALS

The subroutines QAGI and DQAGI are available for computing integrals over infinite intervals. QAGI yields single precision results and DQAGI yields double precision results. QAGI and DQAGI are adaptive routines.

CALL QAGI(F,a,MO,AERR,RERR,z,ERROR,NUM,IERR,ℓ,m,n,IWK,WK)

$F(x)$ is a user defined function whose arguments and values are assumed to be real numbers. The argument a is a real number, z is a variable, and MO may be 1, -1, or 2. When QAGI is called, z is assigned the value $\int_a^\infty F(x)\,dx$ if MO $= 1$ and the value $\int_{-\infty}^a F(x)\,dx$ if MO $= -1$. Otherwise, if MO $= 2$ then z is assigned the value $\int_{-\infty}^\infty F(x)\,dx$. If MO $= \pm 1$ then F need not be defined at a. Otherwise, if MO $= 2$ then a is not used. F must be declared in the calling program to be of type EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used. The subroutine attempts to obtain a value z which satisfies $|\int F(x)\,dx - z| \leq max\{$AERR,RERR$\cdot |\int F(x)\,dx|\}$. It is assumed that AERR ≥ 0 and RERR ≥ 0. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR and NUM are variables. When QAGI terminates, ERROR is the estimated absolute error of z and NUM the number of points at which F was evaluated.

IWK is an array of dimension ℓ and WK is an array of dimension m. IWK and WK are work spaces for the routine. The input argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$ and $m \geq 4\ell$. The argument n is a variable. When QAGI terminates, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

- IERR $= 0$ The routine is satisfied that the integral has been computed to the desired accuracy.
- IERR $= 1$ The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
- IERR $= 2$ The integral has been computed, but because of roundoff error QAGI is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
- IERR $= 3$ Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.
- IERR $= 4$ The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.
- IERR $= 5$ The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.

IERR = 6 (Input Error) Either $\ell < 1$, $m < 4\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, NUM, and n are set to 0.

Note. F may have a singularity at a when MO $= \pm 1$. However, it is recommended that no singularities appear in the interior of the interval of integration.

Algorithm. The integrals are transformed as follows:

$$\int_a^\infty F(x)\,dx = \int_0^1 F(a-1+1/t)\frac{dt}{t^2}$$
$$\int_{-\infty}^a F(x)\,dx = \int_0^1 F(a+1-1/t)\frac{dt}{t^2}$$
$$\int_{-\infty}^\infty F(x)\,dx = \int_0^1 [F(-1+1/t)+F(1-1/t)]\frac{dt}{t^2}$$

The transformed integrals are computed by the 15 point Kronrod rule and the ϵ-algorithm of P. Wynn.

Programming. QAGI employs the subroutines QAGIE, QK15I, QPSRT, and QELG. These routines were developed by Robert Piessens and Elise de Doncker-Kapenga (Katholieke Universiteit Leuven, Heverlee, Belgium), and modified by A. H. Morris and Los Alamos National Laboratory. The function SPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK*: *A Subroutine Package for Automatic Integration***, Springer-Verlag, 1983.

CALL DQAGI(F,a,MO,AERR,RERR,z,ERROR,NUM,IERR,ℓ,m,n,IWK,WK)

$F(x)$ is a user defined function whose arguments and values are assumed to be double precision numbers. The argument a is a double precision number, z is a double precision variable, and MO may be 1, -1, or 2. When DQAGI is called, z is assigned the value $\int_a^\infty F(x)\,dx$ if MO $= 1$ and the value $\int_{-\infty}^a F(x)\,dx$ if MO $= -1$. Otherwise, if MO $= 2$ then z is assigned the value $\int_{-\infty}^\infty F(x)\,dx$. If MO $= \pm 1$ then F need not be defined at a. Otherwise, if MO $= 2$ then a is not used. F must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used. The subroutine attempts to obtain a value z which satisfies $|\int F(x)\,dx - z| \leq max\{$AERR,RERR$\cdot |\int F(x)\,dx|\}$. It is assumed that AERR and RERR are nonnegative double precision numbers. If one wants accuracy to k significant digits then set RERR $= 10^{-k}$. If RERR $= 0$ then it is assumed that machine accuracy is desired.

ERROR is a double precision variable and NUM an integer variable. When DQAGI terminates, ERROR is the estimated absolute error of z and NUM the number of points at which F was evaluated.

IWK is an integer array of dimension ℓ and WK a double precision array of dimension m. IWK and WK are work spaces for the routine. The argument ℓ is the maximum number

of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geq 1$ and $m \geq 4\ell$. The argument n is a variable. When DQAGI terminates, n= the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has of the following values:

IERR = 0 The routine is satisfied that the integral has been computed to the desired accuracy.

IERR = 1 The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.

IERR = 2 The integral has been computed, but because of roundoff error DQAGI is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.

IERR = 3 Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.

IERR = 4 The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.

IERR = 5 The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.

IERR = 6 (Input Error) Either $\ell < 1$, $m < 4\ell$, AERR < 0, or RERR < 0. In this case, the variables z, ERROR, NUM, and n are set to 0.

Remarks. F may have a singularity at a when MO $= \pm 1$. However, it is recommended that no singularities appear in the interior of the interval of integration. DQAGI is a double precision version of the routine QAGI.

Programming. DQAGI employs the routines DQAGIE, DQK15I, DQPSRT, and DQELG. These subroutines are double precision versions by A. H. Morris of the subroutines QAGIE, QK15I, QPSRT, and QELG. The function DPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK*: *A Subroutine Package for Automatic Integration***, Springer-Verlag, 1983.

EVALUATION OF DOUBLE INTEGRALS OVER TRIANGLES

Let $f(x,y)$ be a real-valued function defined on a triangle T. Then the subroutine CUBTRI is available for computing the integral $\iint_T f(x,y)\,dx\,dy$. CUBTRI is an adaptive routine.

CALL CUBTRI(F,T,ϵ,MAX,A,ERR,n,W,ℓ,IDATA,RDATA,IERR)

T is a 2-dimensional real array of dimension 2×3 where $T(1,j)$ and $T(2,j)$ are the x and y coordinates of the j^{th} vertex of the given triangle ($j = 1,2,3$).

IDATA and RDATA are arrays provided by the user containing any integer or real data needed for computing the integrand $f(x,y)$. The arrays may be of any size. F is a user defined real-valued function having the arguments x,y,IDATA,RDATA. It is assumed that $F(x,y$,IDATA,RDATA$) = f(x,y)$ for any point (x,y) in the triangle of integration T. F must be declared in the calling program to be of type EXTERNAL.

The input argument ϵ is the error tolerance to be satisfied, and A is a variable. When CUBTRI is called, A is assigned the value obtained for $\iint_T f(x,y)\,dx\,dy$. The routine attempts to obtain a value A which satisfies $|\iint f(x,y)\,dx\,dy - A| < max\{\epsilon, \epsilon|A|\}$. Thus if $|A| \leq 1$ then ϵ is an absolute tolerance, whereas if $|A| > 1$ then ϵ is a relative tolerance. If one wants k digit accuracy then set $\epsilon = 10^{-k}$. ERR is a variable. When CUBTRI terminates, ERR is the estimated error $|\iint f(x,y)\,dx\,dy - A|$ of the result.

The input argument MAX is the maximum number of points (x,y) at which F may be evaluated, and n is a variable. On an initial call to CUBTRI, the user must set $n = 0$. When the routine terminates, n will have for its value the number of points at which F was evaluated. (For subsequent calls concerning the same integral, see below.)

W is an array of dimension ℓ for internal use by the routine. The input argument ℓ specifies the maximum number of subtriangles in which the triangle of integration T may be partitioned. The subtriangles are stored in W, each subtriangle requiring 6 storage locations. Thus $\ell/6$ is an estimate of the maximum number of subtriangles that might have to be stored ($\ell \quad max\{1, 3m + 1\}$ where $m \quad$ (MAX/19 $\quad$ 1)/4).

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR 0 The integral was computed to the desired accuracy.

IERR 1 MAX is too small. F must be evaluated at more points.

IERR 2 The storage space W is full. Its dimension ℓ must be increased.

IERR 3 Further subdivision of the subtriangles impossible. This normally occurs when $f(x,y)$ has a singularity in the region. The situation can frequently be eliminated by placing the singularity at a vertex of the triangle of integration T.

IERR 4 No further improvement in accuracy is possible because of roundoff error in the computation of F or the irregular behavior of F.

IERR = 5 No further improvement in accuracy is possible because subdivision does not change the estimated integral value A. Machine accuracy has probably been reached.

After an initial call to CUBTRI, the routine may be recalled to continue the computation of $\iint_T f(x,y)\,dx\,dy$. When the routine is recalled, the value of n obtained on the previous call to CUBTRI is used for the next call. This value for n tells the routine where computation should be resumed (using the information previously stored in W). At least one of the values ϵ, MAX, or ℓ must be modified before CUBTRI is recalled. F, T, n, W, IDATA, and RDATA may not be changed when the routine is recalled.

Remark. F may have a singularity at one of the vertices of T (such as in the case when we are computing $\int_0^1 \int_0^x (x^2+3y^2)^{-1/2}\,dy\,dx$). However, it is recommended that no singularities appear in the interior of the triangle of integration.

Algorithm. The 7-point degree 5 rule of Radon and a new 19-point degree 8 rule are used.

Programming. CUBTRI calls the function RNDERR and subroutine CUBRUL. Information is saved in labeled common blocks. The block names are CUBSTA and CUBATB. The routines were written by D. P. Laurie (National Research Institute for the Mathematical Sciences, Pretoria, South Africa).

Reference. Laurie, D. P., "Algorithm 584, CUBTRI: Automatic Cubature over a Triangle," *ACM Trans. Math Software* 8 (1982), pp. 210–218.

SOLUTION OF FREDHOLM INTEGRAL EQUATIONS OF THE SECOND KIND

If $k(s,t)$ and $f(s)$ are continuous real-valued functions for $a \le s \le b$ and $a \le t \le b$, then the equation to be solved is

$$x(s) - \int_a^b k(s,t)x(t)\,dt = f(s)$$

for $a \le s \le b$. Let K be the operator defined by $(Kx)(s) = \int_a^b k(s,t)x(t)\,dt$ for any real-valued function x continuous on $[a,b]$. Then $(Kx)(s)$ is continuous for $a \le s \le b$, and k is called the ***kernel*** of K. Also the above integral equation can be written in the form $(I - K)x = f$ where I is the identity operator. This equation has a unique solution if and only if $I - K$ is $1 - 1$, in which case $x = (I - K)^{-1}f$. The subroutine IESLV is available for computing this solution.

Remark. If $C[a,b]$ is the normed space of real-valued functions x continuous on $[a,b]$ and having the norm $\|x\| = max\{\,|x(t)| : a \le t \le b\,\}$, then K is a compact mapping $C[a,b] \to C[a,b]$ having the norm $\|K\| = \max\limits_{a \le s \le b} \int_a^b |k(s,t)|\,dt$.

CALL IESLV(k, f, a, b,EPS,IFLAG,S, X, ℓ, N, M,NF,MF,NORMK,WK,IERR)

It is assumed that $a < b$, and that $k(s,t)$ and $f(s)$ are user defined real-valued functions for $a < s,t < b$. It is recommended that k and f be several times continuously differentiable. The functions k and f must be declared in the calling program to be of type EXTERNAL.

EPS is a variable and IFLAG an input argument whose values are 0 and 1. On input EPS is the error tolerance that the solution must satisfy. If IFLAG = 0 then EPS is an absolute tolerance. Otherwise, if IFLAG = 1 then EPS is a relative tolerance. If IESLV solves the equation, then on output EPS is the estimated error of the result.

Before the remaining arguments $s, x, \ell, \ldots$ can be described, it is necessary to give a brief outline of the algorithm used. When IESLV is called, the integral equation is approximated by

$$(*) \qquad x_n(s) - \sum_{j=1}^{n} w_{jn}k(s,t_{jn})x_n(t_{jn}) = f(s)$$

for $a < s < b$. Here w_{jn} and t_{jn} are the weights and nodes of Gauss-Legendre quadrature. This equation is treated as an interpolation for $x(s)$ in terms of the values $x_n(t_{jn})$. These values are obtained by solving the equations

$$(**) \qquad x_n(t_{in}) - \sum_{j=1}^{n} w_{jn}k(t_{in},t_{jn})x_n(t_{jn}) = f(t_{in})$$

for $i = 1, \ldots, n$. This system of equations can be solved directly or iteratively. The following algorithm is used:

(1) Set $n = 2$ and go to (2).

(2) The n equations are solved directly. Then set $m = 2n$ and solve the m equations (**) iteratively. If the rate of convergence is sufficiently rapid or n cannot be increased, then go to (3). Otherwise, set $n = m$, and go to (2).

(3) Here n remains fixed. Repeatedly double the value of m and solve the m equations (**) iteratively until convergence occurs, m cannot be increased, or the iterations diverge.

When the algorithm terminates, values $x_m(t_{im})$ will have been computed for the nodes $t_{im} (i = 1, \ldots, m)$. Then from (*), $x(s) \approx x_m(s)$ can be interpolated for $a \leq s \leq b$.

N and M are input arguments, and WK is an array that is a work space for the routine. N and M are upper limits for n and m in the algorithm, and WK is of dimension $5N^2 + 9(N + M)$ or larger. It is assumed that $M \geq N \geq 2$. Since n and m are always powers of 2, N and M need only be set to powers of 2. However, this is not required.

S and X are arrays, and ℓ is a variable. On input it is assumed that $\ell \geq 0$. If $\ell > 0$ then S is assumed to contain ℓ points $s_1, \ldots, s_\ell$ at which the solution $x(s)$ is to be evaluated. Also X is assumed to be an array of dimension ℓ or larger. When IESLV terminates, X contains the values obtained for $x(s_1), \ldots, x(s_\ell)$. (This is true irregardless of whether or not the desired accuracy has been achieved.) Otherwise, if $\ell = 0$ then S and X are assumed to be arrays of dimension M or larger. When IESLV terminates $\ell =$ the final value obtained for m, S contains the Gaussian nodes $t_{i\ell} (i = 1, \ldots, \ell)$, and X contains the values obtained for $x(t_{i\ell})$.

NF and MF are variables. When the routine terminates, NF is the final value for n and MF the final value for m.

NORMK is a real variable. If $\ell > 0$ on input, then when IESLV terminates, NORMK is an approximation for $\|K\|$. Otherwise, if $\ell = 0$ then NORMK = 0.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0 The solution was obtained to the desired accuracy. EPS is the estimated error of the result.

IERR = 1 The solution was not obtained to the desired accuracy. EPS is the estimated error of the result.

IERR = 2 The solution was not obtained to the desired accuracy. It is not clear what accuracy (if any) has been achieved. EPS has been set to 0.

IERR = 3 The input value for EPS was too small. This may be due to ill-conditioning of the integral equation. The value of EPS was reset to a more realistic tolerance, which the solution satisfied.

IERR = 4 The solution $x(s)$ was obtained at the Gaussian nodes to the desired precision. However, the interpolation process may not preserve this accuracy for the evaluation of $x(s)$ for other points s. EPS is the estimated error of the solution at the Gaussian nodes.

IERR = 5 The solution $x(s)$ was not obtained to the desired accuracy at the Gaussian nodes. EPS is the estimated error at these nodes. The interpolation process may not preserve this accuracy for the evaluation of $x(s)$ for other points s.

IERR = 6 The input value for EPS was too small. This may be due to ill-conditioning of the integral equation. The value of EPS was reset to a more realistic tolerance, which the solution $x(s)$ satisfied at the Gaussian nodes. The interpolation process may not preserve this accuracy for the evaluation of $x(s)$ for other points s.

Difficulties can arise, causing IERR ≥ 1, when the integral equation is ill-conditioned or the kernel $k(s,t)$ is not appropriate for standard Gaussian quadrature. Ill-conditioning can occur whe the operator $I - K$ is near singular or the norm $\|K\|$ is exceedingly large. Inappropriate kernels $k(s,t)$ include those which are highly oscillatory or not continuously differentiable for s and t in the open interval (a,b).

Programming. IESLV employs the subroutines IEGS, NSTERP, WANDT,LEAVE, ITERT, LNSYS and functions RMIN,RNRM,CONEW. The routines save and exchange information in labled common blocks. The block names are XXINFO and XXLIN. The routines were written by Kendall E. Atkinson (University of Iowa), and modified by A H. Morris. The function SPMPAR is also used.

Reference. Atkinson, K. E.,"An Automatic Program for Linear Fredholm Integral Equations of the Second Kind," ***ACM Trans. Math Software*** **2** (1976), pp. 154–171.

THE INITIAL VALUE SOLVERS — INTRODUCTORY COMMENTS

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t, y) = (f_1(t, y), \ldots, f_n(t, y))$. Assume that $y(a)$ is known. Then for $b \neq a$ the subroutines ODE, BRKF45, RKF45, GERK, SFODE, and SFODE1 are available for computing $y(b)$. These routines are adaptive variable step, multistep differential equations solvers. The remaining subroutines (RK and RK8) are one step procedures. Given the values $y(t)$ and h, the one step solvers compute a value for $y(t + h)$. The problem of selecting a suitable value for h is left to the user. Given $y(a)$ and b, the one step solvers must be repeatedly called to step along the interval from a to b. In contrast, given $y(a)$, b, and error tolerances that are to be satisfied, the adaptive solvers continually adjust their step size (and possibly order) as they automatically step along the interval from a to b.

The adaptive subroutines differ in their capabilities. ODE, BRKF45, RKF45, and GERK are recommended for nonstiff equations, and SFODE and SFODE1 for stiff equations. If one does not know whether the equations are stiff, then ODE should be tried. ODE maintains greater accuracy than the other subroutines, and it notifies the user if the equations appear to be stiff. ODE, BRKF45, RKF45, and GERK are able to handle mildly stiff problems satisfactorily, but SFODE and SFODE1 are the only subroutines that can be used for solving extremely stiff problems.

If the equations are nonstiff, then ODE and BRKF45 should first be considered for solving the equations. If accuracy to 10 or more digits is needed then ODE should be used. Otherwise, if accuracy to 8 or fewer digits is desired and the derivative evaluations are inexpensive, then BRKF45 may be the most efficient subroutine for solving the problem. BRKF45 normally requires more derivative evaluations than ODE, but its overhead is less than that for ODE. The nonstiff solvers RKF45 and GERK may also be used if accuracy to 8 or fewer digits is desired. However, these routines are not recommended since interpolation is not employed (see the output comments below).

When the user specifies error tolerances to be satisfied, normally he is interested only in the global error (the error of $y(b)$). However, the adaptive subroutines employ the tolerances for controlling local error (the error generated at each step in the interval from a to b). No attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. GERK is the only subroutine that estimates the global error. This subroutine employs the same Runge-Kutta-Fehlberg formulae used by RKF45. GERK is 2-3 times slower than RKF45, but it is more accurate. RKF45 and BRKF45 take roughly the same amount of time, and have approximately the same accuracy.

Output Considerations. Generally, when the equations $y'(t) = f(t, y(t))$ are to be solved and $y(a_0)$ is known, then solutions will be needed at a sequence of points $a_1, \ldots, a_m$. If ODE, BRKF45, SFODE, or SFODE1 is used, then the closeness of the output points a_i should be of no concern. The subroutines partially ignore a_{i+1} in the selection of the step size when going from a_i to a_{i+1}. Instead, they step along the interval using the largest steps that are appropriate (accuracy and efficiency are the prime concerns). Normally a_{i+1} will be passed in the process. If a_{i+1} is passed then a quick interpolation yields the desired

result for $y(a_{i+1})$. Thus, the process of solving the equations at a_{i+1} when $y(a_i)$ is known may require that no steps be taken (a_{i+1} may have been passed on a previous call to the routine), or it may require that one or more steps be taken.

The situation is considerably different when RKF45 or GERK is used. Since these subroutines do not have an interpolation procedure, they must select a step size so as not to bypass a_{i+1} when going from a_i to a_{i+1}. Thus, the output points a_i may be so close to one another that inordinately small step sizes are required. If this occurs then the performance of RKF45 and GERK may deteriorate dramatically. The subroutines notify the user when this occurs.

ADAPTIVE ADAMS SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y), \ldots, f_n(t,y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(a)$ is known. Then for $b \neq a$ the subroutine ODE is available for computing $y(b)$. ODE is recommended for nonstiff equations. The algorithm used is a variable order, variable step Adams predictor-corrector procedure.

CALL ODE(F, n, Y, T, TOUT,RERR,AERR,IND,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL $F(t, Y,$ DY)

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $100 + 21n$ or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to ODE.

It is assumed that $a \neq b$. The argument Y of ODE is an array of dimension n, and the arguments T,RERR,AERR,IND are variables. (TOUT need not be a variable.) When ODE is initially called, it is assumed that:

$T = a$
TOUT $= b$
$Y(1), \ldots, Y(n)$ contain the values $y_1(a), \ldots, y_n(a)$
RERR = the relative error tolerance to be satisfied
AERR = the absolute error tolerance to be satisfied
IND $= \pm 1$

It is preferable, both for efficiency and accuracy, that ODE be permitted to step along the axis from a to b using the largest steps that are appropriate. This is what is done when IND is set to 1. If IND $= 1$ then ODE will step along the axis, possibly passing b and going as far as the point $a + 10(b - a)$. If b is passed, then the solution for the equations at b is obtained by interpolation. However, IND $= 1$ cannot be used if the equations are not defined at all points between b and $a + 10(b - a)$. In a situation such as this, when integration cannot be permitted to step internally past TOUT, IND must be set to -1. If IND $= -1$ then it is required that the subroutine F be defined at TOUT. However, F need not be defined at points t past TOUT. If the equations $y'(t) = f(t, y(t))$ are not defined at $t =$ TOUT, then it should suffice to let F set each DY(i) $= 0$ when $t =$ TOUT. A solution (if one exists) will be obtained by extrapolation.

If IND is positive (negative), then when ODE terminates IND will have been reset by ODE to one of the values $2,3,4,5,6,7 (2, -3, -4, -5, -6, 7)$. These values have the following meanings:

IND $= 2$ The equations have been solved at TOUT. T now has the value TOUT and Y contains the solution at TOUT.

IND $= \pm 3$ The error tolerances RERR and AERR are too small. T is set to the point closest to TOUT for which the equations were solved

and Y contains the solution at the point. RERR and AERR have been reset to larger acceptable values.

IND $= \pm 4$ MAXNUM steps were performed.[1] More steps are needed to reach TOUT. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point.

IND $= \pm 5$ MAXNUM steps were performed. More steps are needed to reach TOUT. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point. The equations appear to be stiff.

IND $= \pm 6$ ODE did not reach TOUT because AERR $= 0$. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point.

IND $= 7$ No computation was performed. An input error was detected. The user must correct the error and call ODE again.

If IND $= \pm 3, \pm 4, \pm 5$ then to continue the integration just call ODE again. Similarly, if IND $= \pm 6$ then reset AERR to be positive and call ODE again. In these cases do not modify T, Y,IND. The output values for these parameters are the appropriate input values for the next call to ODE. However, AERR and RERR may always be modifed when continuing an integration.

If the equations appear to be stiff (i.e., if IND $= \pm 5$) then ODE may not be suitable for solving the equations. In this case it is recommended that a routine designed specifically for stiff equations be used.

Whenever IND $= 2$ occurs, then the equations have been solved at TOUT $= b$. WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration, normally one need only reset TOUT to the new value c and call ODE again. Do not modify T, Y,IND. The output values for these parameters are normally the appropriate input values for the next call to ODE. The one exception is when the equations are not defined at points past c. If this occurs, then one should also reset the output value IND $= 2$ (from the last call to ODE) to the input value IND $= -2$ for the next call to ODE. If IND is reset to -2, then integration will not proceed internally past the new TOUT when ODE is recalled. In this case, the subroutine F need not be defined for points past TOUT. However, it is required that F be defined at TOUT.

If after going from a to b, ODE is recalled to continue the integration and solve the equations at a new point c, then it is important that IND be set to ± 2 for the next call to ODE. Setting IND to ± 1 causes the integration procedure to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting not only can take more time, but also can lead to less accurate results. If IND is set to ± 2, then the integration procedure restarts itself only if the direction of integration is being reversed or IND was negative when ODE was last recalled. The direction of integration is reversed when b does not lie between a and c.

[1] Each step normally requires two calls to the subroutine F. Currently the internal parameter MAXNUM is set at 500.

If one has a choice between setting IND to be positive or negative, then always set IND to be positive. Extrapolation is normally involved when IND is negative. The extrapolation can require more time and be less accurate than the procedures employed when IND is positive.

Input Errors. IND = 7 when one of the following conditions is violated:

(1) $n \geq 1$
(2) $T \neq$ TOUT
(3) RERR ≥ 0 and AERR ≥ 0
(4) RERR and AERR are not both 0
(5) $1 \leq |\text{IND}| \leq 5$, or IND $= \pm 6$ and AERR > 0
(6) When continuing an integration, the input value for T is the output value of TOUT from the previous call to ODE.

The last condition is automatically satisfied if the user has not inadvertently modified T.

Error Control. Assuming that ODE has obtained the correct value for $y(t)$, let e_i denote the error generated in the computation $Y(i)$ of $y_i(t+h)$ for $i = 1, \ldots, n$ when ODE steps from t to $t+h$. The routine attempts at each step to maintain the accuracy $\Sigma_i(e_i/w_i)^2 \leq 1$ where $w_i =$RERR $|Y(i)|+$AERR. When this criterion is satisfied, then $|e_i| \leq w_i$ for $i = 1, \ldots, n$. This criterion includes as special cases relative error (AERR = 0) and absolute error (RERR = 0). However, if AERR = 0 and $Y(i) = 0$ for some i, then $w_i = 0$ and IND $= \pm 6$.

When going from T to TOUT, ODE continually adjusts its order and step size so as to maintain accuracy at each step. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results by rerunning the problem. If this is done, then in the second run ask for greater accuracy.

Programming. ODE employs the subroutines DE1, STEP1, and INTRP. These routines were written by L. F. Shampine and M. K. Gordon (Sandia Laboratories). The function SPMPAR is also used.

Reference. Shampine, L. F., and Gordon, M. K., ***Computer Solution of Ordinary Differential Equations***, W. H. Freeman and Company, San Francisco, 1975.

ADAPTIVE BLOCK RKF SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t, y) = (f_1(t, y), \ldots, f_n(t, y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(a)$ is known, and that f is defined and continuous on the interval from a to TEND. Then for any b in this interval the subroutine BRKF45 is available for computing $y(b)$ and its derivative $y'(b)$. BRKF45 was designed for solving nonstiff differential equations when derivative evaluations are inexpensive and the accuracy requirements are low (8 or fewer significant digits). The subroutine employs the Cash block Runge–Kutta–Fehlberg procedure.

CALL BRKF45(F, n, Y, T,TOUT,TEND,DY,RERR,AERR,IND,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL $F(t, Y$,DY)

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y'_1(t), \ldots, y'_n(t)$ using $y'(t) = f(t, y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $6 + 9n$ or larger, and IWK an array of dimension 5 or larger. WK and IWK contain information needed by subsequent calls to BRKF45.

The arguments Y and DY of BRKF45 are arrays of dimension n, and the arguments T, RERR, and IND are variables. (TOUT, TEND, and AERR need not be variables.) When BRKF45 is initially called, it is assumed that:

$T = a$
TOUT $= b$
TEND = end of the interval of integration (TEND $\neq a$)
$Y(1), \ldots, Y(n)$ contain the values $y_1(a), \ldots, y_n(a)$
RERR = the relative error tolerance to be satisfied (RERR ≥ 0)
AERR = the absolute error tolerance to be satisfied (AERR ≥ 0)
IND $= \pm 1$

For efficiency, BRKF45 will step along the interval (in 2–step blocks) from a to b, using the largest steps that are appropriate. IF IND $= -1$ then only a single 2-step block will be taken. Otherwise, if IND $= 1$ then BRKF45 will step along the interval, possibly passing b (but never passing TEND). If b is passed then the solution for the equations at b will be obtained by interpolation.

On output T is set to the point closest to TOUT for which the equations were solved, Y contains the solution $y(T)$ at T, and DY contains the derivative $y'(T)$ at T. Also, IND reports the status of the results. BRKF45 sets IND to one of the following values:

IND $= 2$ Either the equations were solved at TOUT (in which case, T now has the value TOUT), or TOUT was beyond TEND and the equations were solved at TEND (in which case, T now has the value TEND).

IND $= -2$ A single 2 step block in the direction of TOUT was taken.

IND = 3 The error tolerance RERR was too small. RERR has been reset to a larger acceptable value.

IND = 4 18000 derivative evaluations (involving approximately 2000 2-step blocks) were performed. More derivative evaluations are needed to reach TOUT.

IND = 5 BRKF45 did not reach TOUT because AERR = 0. AERR must be made positive.

IND = 6 Too much accuracy was requested. RERR or AERR must be increased in value.

IND = 7 An input error was detected (see below).

If IND = 2 and T now has the value TOUT, then the equations have been solved at TOUT = b. The arrays WK and IWK contain information that can be reused in continuing along the interval from b to TEND and solving the equation at a new point c. To continue the integration the user need only reset TOUT to the new point c and call BRKF45 again.

If IND = -2 then to continue the integration another 2-step block just call BRKF45 again. In the single 2-step block mode (IND = -1, -2) the user must keep in mind that each step taken is in the direction of the current TOUT. Upon reaching TOUT (which is indicated by IND = 2 and T = TOUT), the user may define a new TOUT and set IND = ± 2 for further integration.

If IND = 3 or 4 then to continue the integration just call BRKF45 again. However if IND = 5 then the user must first reset AERR to be positive before BRKF45 can be recalled. If this is not done then the run will be terminated by a STOP instruction! If IND = 6 then it is required that IND be reset to ± 2 and that AERR or RERR be increased in value. If this is not done then the run will be terminated by a STOP instruction.

If after going from a to b, BRKF45 is recalled to continue the integration and solve the equations at a new point c, then it is important that IND be set to ± 2 instead of ± 1. Setting IND = ± 1 causes the integration process to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting wastes time and is normally not needed. The only exceptions are when the direction of integration is reversed or TEND is modified. Then the integration must be restarted. (TEND must be modified when the direction of integration is reversed.)

The purpose for the argument TEND is to ensure that no integration is performed past this point. This is important since F may not be continuous at TEND, or F may not be defined beyond TEND. When the equations are solved at TEND (which is indicated by IND = 2 and T = TEND), then no further integration can be performed. It is always assumed that $T \neq$ TEND on input.

Notes.

(1) T, n, WK, IWK must never be modified on a continuation call to BRKF45. However, AERR and RERR may be modified at any time.

(2) When continuing an integration, one may switch from the standard multistep mode (IND = 2) to the single 2-step block mode (IND = -2) when it is convenient to do so.

Input Errors. IND = 7 when one of the following errors is detected:

(1) $n < 0$
(2) T = TEND
(3) Either AERR or RERR is negative.
(4) TOUT and TEND are in opposite directions from T.
(5) IND = 0 or |IND| > 7.

If (2) occurs then integration cannot be continued. Otherwise, the error must be corrected and IND reset to ± 1 (or ± 2 when the previous call was a continuation call). If this is not done then the run will be terminated by a STOP instruction when BRKF45 is recalled.

Error Control. When going from T to TOUT, BRKF45 continually adjusts its step size so as to maintain accuracy at each 2-step block. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results by rerunning the problem with BRKF45 or ODE. If this is done then in the second run ask for greater accuracy.

Programming. BRKF45 employs the subroutines RKFC and EXTRA. These routines were written by J. R. Cash (Imperial College, London), and modified by Desmond J. Higham and A. H. Morris. The function SPMPAR is also used.

References.

(1) Cash, J.R.,"A Block 6(4) Runge-Kutta Formula for Nonstiff Initial Value Problems," ***ACM Trans. Math Software*** **15** (1989), pp. 15–28.
(2) Higham, D.J., "Remark on Algorithm 669," ***ACM Trans. Math Software*** **17** (1991), pp. 424–426.

ADAPTIVE RKF SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t, y) = (f_1(t, y), \ldots, f_n(t, y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(a)$ is known. Then for $b \neq a$ the subroutine RKF45 is available for computing $y(b)$. RKF45 was designed for solving nonstiff differential equations when derivative evaluations are inexpensive and the accuracy requirements are low (less than 8 significant digits). The routine employs the fourth-fifth order Runge-Kutta-Fehlberg formulae.

CALL RKF45(F, n, Y, T, TOUT,RERR,AERR,IND,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL $F(t,Y,$DY)

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $3 + 6n$ or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to RKF45.

The argument Y of RKF45 is an array of dimension n, and the arguments T, RERR, IND are variables. (TOUT and AERR need not be variables.) When RKF45 is initially called, it is assumed that:

$T = a$

TOUT $= b$

$Y(1), \ldots, Y(n)$ contain the values $y_1(a), \ldots, y_n(a)$

RERR = the relative error tolerance to be satisfied (RERR > 0)

AERR = the absolute error tolerance to be satisfied (AERR ≥ 0)

IND $= \pm 1$

Normally IND $= 1$. However, if only a single step in the direction of TOUT is to be taken, then set IND $= -1$.

On output T is set to the point closest to TOUT for which the equations were solved, and Y contains the solution at T. Also IND reports the status of the results. RKF45 sets IND to one of the following values:

IND $= 2$ The equations were successfully solved at TOUT. T now has the value TOUT.

IND $= -2$ A single step in the direction of TOUT was taken.

IND $= 3$ The error tolerance RERR was too small. RERR has been reset to a larger acceptable value.

IND $= 4$ 3000 derivative evaluations were performed. More derivative evaluations are needed to reach TOUT.

IND $= 5$ RKF45 did not reach TOUT because AERR $= 0$. AERR must be made positive.

IND $= 6$ Too much accuracy has been requested. AERR or RERR must be increased in value.

IND = 7 The closeness of the output points is restricting the natural step size choice.

IND = 8 An input error was detected (see below).

If IND = 2 then the equations have neen solved at TOUT = b. The arrays WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration the user need only reset TOUT to the new point c and call RKF45 again.

If IND = -2 then to continue the integration another single step just call RKF45 again. In the single step mode (IND = $-1, -2$) the user must keep in mind that each step taken is in the direction of the current TOUT. Upon reaching TOUT (which is indicated by IND being set to 2), the user may then define a new TOUT and set IND to ± 2 for further integration.

If IND = 3 or 4 then to continue the integration just call RKF45 again. However, if IND = 5 then the user must first reset AERR to be positive before RKF45 can be recalled. If this is not done then the run will be terminated by a STOP instruction! If IND = 6 then it is required that IND be reset to ± 2 and that AERR or RERR be increased in value. If this is not done then the run will be terminated by a STOP instruction.

If IND = 7 then the user should switch to a routine (such as ODE or BRKF45) which performs interpolation. If the user insists on continuing the integration with RKF45 then it is required that IND be reset to ± 2 before RKF45 is recalled. If this is not done then the run will be terminated by a STOP instruction.

If after going from a to b, RKF45 is recalled to continue the integration and solve the equations at a new point c, then it is important that IND be set to ± 2 instead of ± 1. Setting IND = ± 1 causes the integration process to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting wastes time and is normally not needed. The one exception is when the direction of integration is to be reversed. Then the integration must be restarted.

Notes.

(1) AERR and RERR can be modified each time that RKF45 is called.

(2) When continuing an integration, one may switch from the standard multistep mode (IND = 2) to the one step mode (IND = 2) whenever it is convenient to do so.

Input Errors. IND = 8 occurs when one of the following conditions is violated:

(1) $n < 1$

(2) $T \neq$ TOUT when IND $\neq \pm 1$

(3) RERR < 0 and AERR < 0

(4) IND $= \pm 1, \pm 2, 3, 4, \ldots, 8$

If IND = 8 then the error must be corrected and IND reset to ± 1 (or ± 2 when the previous call was a continuation call). If this is not done then the run will be terminated by a STOP instruction when RKF45 is recalled.

Error Control. When going from T to TOUT, RKF45 continually adjusts its step size so as to maintain accuracy at each step. Assuming that RKF45 has obtained the correct value for $y(t)$, let e_i denote the error generated in the computation of $y_i(t+h)$ for $i = 1, \ldots, n$ when RKF45 steps from t to $t+h$. Then at each step the error is controlled so that

$$|e_i| \leq \frac{|y_i(t)| + |y_i(t+h)|}{2} \text{ RERR} + \text{AERR}$$

for $i = 1, \ldots, n$. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results. This can best be done by comparing the results obtained by RKF45 with those obtained by ODE or GERK. If ODE is used then ask for greater accuracy. However, if GERK is used then the current error tolerances can be used. GERK is more accurate than RKF45, and it estimates the global error generated.

Programming. RKF45 employs the subroutines RKFS and FEHL. These routines were written by H. A. Watts and L. F. Shampine (Sandia Laboratories). The function SPMPAR is also used.

References. Shampine, L. F.,and Allen, R. C., ***Numerical Computing: An Introduction***, W. B. Sanders, Philadelphia, 1973.

ADAPTIVE RKF SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS WITH GLOBAL ERROR ESTIMATION

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y), \ldots, f_n(t,y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(a)$ is known. Then for $b \neq a$ the subroutine GERK is available for computing $y(b)$. GERK was designed for solving nonstiff differential equations when derivative evaluations are inexpensive and the accuracy requirements are low (less than 8 significant digits). The routine employs the fourth-fifth order Runge-Kutta-Fehlberg formulae. GERK estimates the accuracy of the solution $y(b)$.

CALL GERK(F, n, Y, T, TOUT,RERR,AERR,IND,GERROR,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL $F(t, Y, \mathrm{DY})$

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $3 + 8n$ or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to GERK.

The argument Y of GERK is an array of dimension n or larger, and the arguments T and IND are variables. (TOUT, RERR, AERR need not be variables.) When GERK is initially called, it is assumed that:

T = a

TOUT = b

$Y(1), \ldots, Y(n)$ contain the values $y_1(a), \ldots, y_n(a)$

RERR = the relative error tolerance to be satisfied (RERR > 0)

AERR = the absolute error tolerance to be satisfied (AERR > 0)

IND = +1

Normally IND = 1. However, if only a single step in the direction of TOUT is to be taken then set IND = −1.

GERROR is an array of dimension n or larger. On output T is set to the point closest to TOUT for which the equations were solved, Y contains the solution at T, and GERROR(i) is an estimate of the error of $Y(i)$ for $i = 1, \ldots, n$. Also IND reports the status of the results. GERK sets IND to one of the following values:

IND = 2 The equations were successfully solved at TOUT. T now has the value TOUT.

IND = 2 A single step in the direction of TOUT was taken.

IND = 3 9000 derivative evaluations were performed. More derivative evaluations are needed to reach TOUT.

IND = 4 GERK did not reach TOUT because AERR = 0. AERR must be made positive.

IND = 5 Too much accuracy has been requested. AERR or RERR must be increased in value.

IND = 6 The closeness of the output points is restricting the natural step size choice.

IND = 7 An input error was detected (see below)

If IND = 2 then the equations have been solved at TOUT = b. The arrays WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration the user need only reset TOUT to the new point c and call GERK again.

If IND = -2 then to continue the integration another single step just call GERK again. In the single step mode (IND = $-1, -2$) the user must keep in mind that each step taken is in the direction of the current TOUT. Upon reaching TOUT (which is indicated by IND being set to 2), the user may then define a new TOUT and set IND to ± 2 for further integration.

If IND = 3 then to continue the integration just call GERK again. However, if IND = 4 then the user must first reset AERR to be positive before GERK can be recalled. If this is not done then the run will be terminated by a STOP instruction! If IND = 5 then it is required that IND be reset to ± 2 and that AERR or RERR be increased in value. If this is not done then the run will be terminated by a STOP instruction.

If IND = 6 then the user should switch to a routine (such as ODE or BRKF45) which performs interpolation. If the user insists on continuing the integration with GERK then it is required that IND be reset to ± 2 before GERK is recalled. If this is not done then the run will be terminated by a STOP instruction.

If after going from a to b, GERK is recalled to continue the integration and solve the equations at a new point c, then it is important that IND be set to ± 2 instead of ± 1. Setting IND = ± 1 causes the integration process to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting wastes time and is normally not needed. The one exception is when the direction of integration is to be reversed. Then the integration must be restarted.

Notes.

(1) AERR and RERR can be modified each time that GERK is called.

(2) When continuing an integration, one may switch from the standard multistep mode (IND = 2) to the one step mode (IND = -2) whenever it is convenient to do so.

Input Errors. IND = 7 occurs when one of the following conditions is violated:

(1) $n \geq 1$

(2) $T \neq$ TOUT when IND $\neq \pm 1$

(3) RERR ≥ 0 and AERR ≥ 0

(4) IND $= \pm 1, \pm 2, 3, 4, \ldots, 7$

If IND = 7 then the error must be corrected and IND reset to ± 1 (or ± 2 when the previous call was a continuation call). If this is not done then the run will be terminated by a STOP

instruction when GERK is recalled.

Accuracy Considerations. Error control in GERK is almost identical to that in RKF45. One minor difference is that GERK never employs relative error tolerances less than $3 \cdot 10^{-11}$, whereas RKF45 never employs relative error tolerances less than 10^{-12}.

The only significant difference between GERK and RKF45 is that GERK generates two solutions for the differential equations, whereas RKF45 generates only one. Let $y(t)$ and $\tilde{y}(t)$ denote the solutions generated by GERK at point t. One of these solutions, say $y(t)$, will frequently be identical to the solution computed by RKF45. When going from t to $t+h$, the step size h is selected so that $y(t+h)$ satisfies the local error criterion. After a suitable h is found then GERK takes two steps, each of length $h/2$, to generate $\tilde{y}(t+h)$ from $\tilde{y}(t)$. When GERK terminates, say at point T, then the $\tilde{y}(T)$ solution is stored in the Y array and GERK estimates the error of $\tilde{y}_i(T)$ to be $(y_i(T) - \tilde{y}_i(T))/31$ for $i = 1, \ldots, n$.

Programming. GERK employs the subroutines GERKS and FEHL. These routines were written by H. A. Watts and L. F. Shampine (Sandia Laboratories). The function SPMPAR is also used.

Reference. Shampine, L. F., and Allen, R. C., ***Numerical Computing: An Introduction***, W. B. Saunders, Philadelphia, 1973.

ADAPTIVE SOLUTION OF STIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y), \ldots, f_n(t,y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(a)$ is known. Then for $b \neq a$ the following subroutines are available for computing $y(b)$. These routines are designed for stiff differential equations. The algorithm used is a variable order, variable step backward differentiation procedure.

CALL SFODE(F, n, Y, T,TOUT,INFO,RERR,AERR,IER,
WK,ℓ,IWK,m,RD,ID)

RD and ID are arrays defined by the user containing any real and integer data that is needed for computing f. These arrays may contain any information that the user desires. The argument F is the name of a user defined subroutine that has the format:

CALL $F(t, Y$,DY,RD,ID)

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

INFO is an array of dimension 4, WK an array of dimension ℓ, and IWK an array of dimension m. WK and IWK are work spaces for the routine, and INFO is an array defined by the user containing information on how the equations are to be treated.

INFO(1): Set INFO(1) = 0 on an initial call to the routine. On a continuation call INFO(1) = 1.

INFO(2): Normally INFO(2) = 0. However, INFO(2) = 1 when the intermediate output mode is desired (see below).

INFO(3): When INFO(3) = 0, SFODE proceeds from a to b using the largest steps that are appropriate. If b is passed then $y(b)$ is obtained by interpolation. However, for some problems the routine cannot be permitted to step past a point TSTOP because $y'(t) = f(t, y(t))$ is discontinuous or not defined beyond TSTOP. When this is the case set INFO(3) = 1 and WK(1) = TSTOP.

INFO(4): When proceeding from a to b, the $n \times n$ Jacobian matrix $Jf(t) = (\partial f_i / \partial y_i)$ is computed and stored in WK. Normally it is assumed that

INFO(4) = 0
$\ell \geq 250 + 10n + n^2$
$m \geq 55 + n$.

However, if $Jf(t)$ is banded for all t, having the lower and upper band widths m_ℓ and m_u where $2m_\ell + m_u < n$, then the following setup can be used:

INFO(4) = 1
IWK(1) = m_ℓ
IWK(2) = m_u
$\ell \geq 250 + 10n + (2m_\ell + m_u + 1)n$
$m \geq 55 + n$

T, TOUT, RERR, and AERR are variables, and the argument Y of SFODE is an array of dimension n. On an initial call to the routine it is assumed that

INFO(1) = 0
T = a
TOUT = b
$Y(1), \ldots, Y(n)$ contain the values $y_1(a), \ldots, y_n(a)$
RERR = the relative error tolerance to be satisfied (RERR ≥ 0)
AERR = the absolute error tolerance to be satisfied (AERR ≥ 0).

IER is a variable. When SFODE terminates T is the final point where the equations were solved, Y contains the solution at T, and IER reports the status of the results. IER is assigned one of the following values:

IER = 1 A step was taken in the intermediate output mode. TOUT was not reached. To continue, call the routine again.

IER = 2 The solution at TOUT was obtained by stepping exactly to TOUT.

IER = 3 The solution at TOUT was obtained by stepping past TOUT and then interpolating. On output T = TOUT.

IER = −1 500 steps have been taken. TOUT has not been reached. To continue, call the routine again.

IER = −2 The tolerances RERR and AERR were too stringent. RERR and AERR have been modified by the routine. The tolerances may be further modified by the user if he desires. To continue, call the routine again.

IER = −3 In this case AERR = 0. SFODE stopped when y_i became 0. INFO(1) was set to $-i$. To continue set AERR to be positive, INFO(1) = 1, and call the routine again.

IER = −6 Convergence failed on the last attempted step. An inaccurate Jacobian matrix may be the problem. To continue, restart the routine by setting INFO(1) = 0 and call the routine again.

IER = −7 Repeated error test failures occurred on the last attempted step. The problem should be reexamined. A singularity may be present in the solution. To continue, restart by setting INFO(1) = 0 and call the routine again.

IER $\leq$ −33 An input error was detected (see below).

When IER $\geq$ −2, then INFO(1) = 1 on output.

When the equations are solved at TOUT (IER = 2 or 3), integration can be continued along the axis to solve the equations at a new point c beyond TOUT. To continue, one need only set TOUT to the new value c and call the routine again. When continuing an integration where INFO(1) = 1, never modify T, Y, WK, IWK, INFO(3), and INFO(4). However, INFO(2), RERR, AERR, RD, and ID may be modified at any time.

Intermediate Output Mode. If one wishes to study the behavior of the solution $y(t)$ as the routine steps from T to TOUT, then set INFO(2) = 1. Then SFODE will stop after each successful step (yielding IER = 1) until TOUT is reached. One may switch from the standard mode of operation (INFO(2) = 0) to the intermediate output mode (INFO(2) = 1) or visa versa at any time.

Remark. The diagnostic IER = −1 does not state that 500 steps have been taken on the current call to SFODE. On an initial call to the routine the step counter is set to 0. On continuation calls, the counter continues to increase until 500 steps have accumulated. When IER = −1 is reported, the counter is reset to 0, and only then does the step counting begin again.

Input Errors. IER is set to one of the following values when an input error is detected.

IER = −33 $n < 1$
IER = −34 RERR < 0
IER = −35 AERR < 0
IER = −36 The routine has been called with TOUT, but it has also been told not to step past the point TSTOP.
IER = −37 T = TOUT. This is not permitted on continuation calls.
IER = −38 The user has modified T.
IER = −39 TOUT is not beyond T. An attempt is being made to change the direction of integration without restarting.
IER = −40 The Jacobian matrix is banded. However, m_ℓ and m_u do not satisfy $0 \le m_\ell < n$ and $0 \le m_u < n$.
IER = −41 $\ell < 250 + 10n + n^2$
IER = −42 $\ell < 250 + 10n + (2m_\ell + m_u + 1)n$
IER = −43 $m < 55 + n$
IER = −44 INFO(1) is incorrect.

After the error is corrected, set INFO(1) = 0 and call the routine again.

Error Control. Assuming that SFODE has the correct value for $y(t)$, let e_i denote the error generated in computing $y_i(t+h)$ for $i = 1, \ldots, n$ when SFODE steps from t to $t+h$. The routine attempts at each step to maintain the accuracy $\frac{1}{n}\Sigma_i(e_i/w_i)^2 \le 1$ where $w_i =$ RERR $|y_i(t)|$ + AERR. When this criterion is satisfied, $|e_i| \le \sqrt{n}w_i$ for $i = 1, \ldots, n$. This criterion includes as special cases relative error (AERR = 0) and absolute error (RERR = 0). However, if AERR = 0 and $y_i(t) = 0$ for some i, then this criterion cannot be applied and IER = −3 occurs.

When proceeding from T to TOUT, the routine continually readjusts its order and step size so as to maintain accuracy at each step. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results. If the problem is nonstiff or mildly stiff for an interval, then the best procedure is to compare the results obtained by SFODE with those obtained by ODE for the interval. ODE normally maintains greater accuracy than SFODE. However, if the problem is extremely stiff then rerun the problem with SFODE. On the second run, request greater accuracy.

Programming. SFODE calls the subroutines STFODE and ZZZJAC. STFODE employs the subroutines LSOD1, HSTART, INTYD, STOD, CFOD, PJAC, SLVS, SGBFA, SGBSL, SGEFA, SGESL, SAXPY, and SSCAL, and the functions VNORM, VNWRMS, ISAMAX, SDOT, and SPMPAR. The routines save and exchange information in a labeled common block having the block name DEBDF1. STFODE is a modification by A. H. Morris of the

subroutine DEBDF, designed by L. F. Shampine and H. A. Watts (Sandia Laboratories). DEBDF appears in the SLATEC library. STFODE is a driver for a modification of the code LSODE, written by A. C. Hindmarsh (Lawrence Livermore Laboratory).

CALL SFODE1(F, n, Y, T,TOUT,INFO,RERR,AERR,IER,
WK,ℓ,IWK,m,RD,ID)

SFODE1 differs from SFODE only in the treatment of RERR and AERR. In SFODE1, RERR and AERR are arrays of dimension n. RERR(i) and AERR(i) are relative and absolute error tolerances to control the accuracy of the i^{th} solution component $y_i(t)$ for $i = 1, \ldots, n$. Let e_i denote the error generated in the computation of $y_i(t+h)$ from $y_i(t)$ when SFODE1 steps from t to $t+h$. Then SFODE1 attempts at each step to maintain the accuracy $\frac{1}{n}\Sigma_i(e_i/w_i)^2 \leq 1$ where $w_i = \text{RERR}(i)|y_i(t)| + \text{AERR}(i)$. When this criterion is satisfied $|e_i| \leq \sqrt{n}w_i$ for $i = 1, \ldots, n$. However, if AERR(i) = 0 and $y_i(t) = 0$ for some i, then the criterion cannot be applied and IER = −3 occurs.

When IER references RERR and AERR, the settings for IER provide the following information:

IER = −2 The accuracy required by RERR and AERR is too stringent. RERR and AERR have been modified by the routine. RERR and AERR may be further modified by the user if he desires. To continue, call SFODE1 again.

IER = −3 SFODE1 stopped when y_i became 0 and AERR(i) = 0. INFO(1) was set to $-i$. To continue set AERR(i) to be positive, INFO(1) = 1, and call the routine again.

IER = −34 (Input error) RERR(i) < 0 for some i.

IER = −35 (Input error) AERR(i) < 0 for some i.

RERR and AERR may be modified on any continuation call to SFODE1.

Programming. SFODE1 calls the subroutines STFODE and ZZZJAC.

FOURTH-ORDER RUNGE-KUTTA

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order nonstiff differential equations where $f(t, y) = (f_1(t, y), \ldots, f_n(t, y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(t_0)$ is known. Then for a small real number h, the subroutine RK is available for computing $y(t_0 + h)$. RK employs the standard fourth-order Runge-Kutta procedure.

CALL RK(n, T, h, A, F)

The argument F is the name of a user defined subroutine that has the format:
CALL $F(t, Z)$
Z is an array of dimension n containing the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in Z, thereby destroying the original data in Z. F must be declared in the calling program to be of type EXTERNAL.

T is a variable having the value t_0 and A an array of dimension $3n$ or larger. It is assumed that $A(1), \ldots, A(n)$ contain the values $y_1(t_0), \ldots, y_n(t_0)$. If $h = 0$ then RK computes the derivatives $y_1'(t_0), \ldots, y_n'(t_0)$ and stores them in $A(n + 1), \ldots, A(2n)$. If $h \neq 0$ then it is assumed that the derivatives $y_1'(t_0), \ldots, y_n'(t_0)$ have already been computed and stored in $A(n + 1), \ldots, A(2n)$. In this case, when RK is called, the values $y_1(t_0 + h), \ldots, y_n(t_0 + h)$ and derivatives $y_1'(t_0 + h), \ldots, y_n'(t_0 + h)$ are computed and stored in $A(1), \ldots, A(2n)$. Also T is reset to the value $t_0 + h$.

Note. The area $A(2n + 1), \ldots, A(3n)$ serves as work space for the routine.

Example. Consider the equations
$x'(t) = y(t)$
$y'(t) = -x(t)$
where $x(0) = 0$ and $y(0) = 1$. The following code may be used for solving these equations at the points $.01, .02, \ldots, 1.00$, and storing the results in the arrays X and Y.

```
      DIMENSION A(6), X(100), Y(100)
      EXTERNAL FUN
      T = 0.0
      H = .01
      A(1) = 0.0
      A(2) = 1.0
      A(3) = 1.0
      A(4) = 0.0
      DO 10 I = 1, 100
      CALL RK(2,T,H,A,FUN)
      X(I) = A(1)
10    Y(I) = A(2)
```

Here FUN may be defined by:

```
SUBROUTINE FUN(T,Z)
DIMENSION Z(2)
X = Z(1)
Y = Z(2)
Z(1) = Y
Z(2) = -X
RETURN
END
```

Note that the statements A(3) = 1.0 and A(4) = 0.0, which store the derivatives $x'(0)$ and $y'(0)$ in A(3) and A(4), can be replaced with CALL RK(2,T,0.0,A,FUN).

Programmer. A.H. Morris.

EIGHTH-ORDER RUNGE-KUTTA

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order nonstiff differential equations where $f(t, y) = (f_1(t, y), \ldots, f_n(t, y))$ and $y(t) = (y_1(t), \ldots, y_n(t))$. Assume that $y(t_0)$ is known. Then for a small real number h, the subroutine RK8 is available for computing $y(t_0 + h)$.

CALL RK8$(n, T, h, Y, \mathrm{DY}, \mathrm{WK}, F)$

The argument F is the name of a user defined subroutine that has the format:

CALL $F(t, Z)$

Z is an array of dimension n containing the values $y_1(t), \ldots, y_n(t)$ for the argument t. F computes the derivatives $y_1'(t), \ldots, y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in Z, thereby destroying the original data in Z. F must be declared in the calling program to be of type EXTERNAL.

T is a variable having the value t_0, and Y and DY are arrays of dimension n. It is assumed that Y contains the values $y_1(t_0), \ldots, y_n(t_0)$. If $h = 0$ then RK8 computes the derivatives $y_1'(t_0), \ldots, y_n'(t_0)$ and stores them in DY. If $h \neq 0$ then it is assumed that the derivatives $y_1'(t_0), \ldots, y_n'(t_0)$ have already been computed and stored in DY. In this case, when RK8 is called, the values $y_1(t_0 + h), \ldots, y_n(t_0 + h)$ and derivatives $y_1'(t_0 + h), \ldots, y_n'(t_0 + h)$ are computed and stored in Y and DY, thereby destroying the original data in Y and DY. Also T is reset to the value $t_0 + h$.

WK is an array of dimension $8n$ or larger that is used for a work space by the routine.

Algorithm. The routine employs formulae (8–12) given on p. 34 of the reference.

Remarks. RK8 is used in the same manner as the routine RK. RK8 takes more time and storage than RK, but may be more accurate.

Reference. Shanks, E. B., "Solutions of Differential Equations by Evaluations of Functions," *Math. Comp.* 20 (1966), pp. 21–38.

Programmer. A. H. Morris.

SEPARABLE SECOND-ORDER ELLIPTIC EQUATIONS ON RECTANGULAR DOMAINS

Given a separable elliptic equation

$$a(x)u_{xx} + b(x)u_x + c(x)u + d(y)u_{yy} + e(y)u_y + f(y)u = g(x,y)$$

on the rectangle $a_1 \le x \le a_2, b_1 \le y \le b_2$, where u is periodic in x or y, or u or its normal derivative $\partial u/\partial n$ is given on each of the edges. For $m, n > 1$ let $x_i = a_1 + (i-1)h$ and $y_j = b_1 + (j-1)k$ where $h = (a_2 - a_1)/(m-1), k = (b_2 - b_1)/(n-1)$, $i = 1, \ldots, m$,and $j = 1, \ldots, n$. Then the subroutine SEPDE is available for computing u at the points (x_i, y_j).

CALL SEPDE(COFX,COFY,g,ITYPE,BVAL,IORD,a_1, a_2, m, b_1, b_2, n, U,ku,W, ℓ,IND)

It is assumed that $m \ge 7$ and $n \ge 6$. U is an $m \times n$ matrix. The argument ku is the number of rows in the dimension statement for U in the calling program. When SEPDE is called, if the elliptic equation is solved then $U(i,j) = u(x_i, y_j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

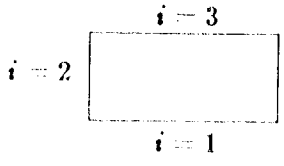
The input argument IORD is the order of the approximation procedure to be used. IORD may have the values 2 or 4.

The argument COFX is the name of a user defined subroutine that has the format:
CALL COFX(x, A, B, C)
A, B, and C are variables. COFX sets $A = a(x)$, $B = b(x)$, and $C = c(x)$ for the argument x. COFX must be declared in the calling program to be of type EXTERNAL.

The argument COFY is the name of a user defined subroutine that has the format:
CALL COFY(y, D, E, F)
D, E, and F are variables. COFY sets $D = d(y)$, $E = e(y)$, and $F = f(y)$ for the argument y. COFY must be declared in the calling program to be of type EXTERNAL.

The argument g is the name of a user defined function, where $g(x,y)$ gives the right hand side of the elliptic equation for all $a_1 \le x \le a_2, b_1 \le y \le b_2$. The argument g must be declared in the calling program to be of type EXTERNAL.

Boundary Conditions. The edges of the rectangular domain are labeled in a clockwise manner as follows:



edge 1 = $\{ (x, b_1) \mid a_1 \le x \le a_2 \}$
edge 2 = $\{ (a_1, y) \mid b_1 \le y \le b_2 \}$
edge 3 = $\{ (x, b_2) \mid a_1 \le x \le a_2 \}$
edge 4 = $\{ (a_2, y) \mid b_1 \le y \le b_2 \}$

ITYPE is an array of dimension 4. For edge i ($i = 1, \ldots, 4$), ITYPE(i) specifies the type of boundary condition on the edge. ITYPE(i) must be set by the user to one of the following values:

ITYPE(i) = 0 It is assumed that u is given on the edge.

ITYPE(i) = 1 If $i = 1$ or $i = 3$ then u_y is given on the edge. Otherwise, if $i = 2$ or $i = 4$ then u_x is given on the edge.

ITYPE(i) = −1 If $i = 1$ or $i = 3$ then it is assumed that u is periodic in y; i.e., $u(x, y + b_2 - b_1) = u(x, y)$ for all x, y. In this case ITYPE(i) must be −1 for both $i = 1$ and $i = 3$. If $i = 2$ or $i = 4$ then it is assumed that u is periodic in x; i.e., $u(x + a_2 - a_1, y) = u(x, y)$ for all x, y. In this case ITYPE(i) must be −1 for both $i = 2$ and $i = 4$.

The argument BVAL is the name of a user defined function. BVAL(i, x, y) is defined for any point (x, y) on edge i when ITYPE(i) = 0 or 1, where

$$\text{BVAL}(i, x, y) = \begin{cases} u(x, y) & \text{if ITYPE}(i) = 0 \\ u_y(x, y) & \text{if ITYPE}(i) = 1 \ (i = 1 \text{ or } i = 3) \\ u_x(x, y) & \text{if ITYPE}(i) = 1 \ (i = 2 \text{ or } i = 4) \end{cases}$$

The function BVAL(i, x, y) is ignored when ITYPE(i) = −1. BVAL must be declared in the calling program to be of type EXTERNAL.

W is an array of dimension ℓ that is a work space. The argument ℓ is a variable whose value depends on IORD, m, n, and the types of boundary conditions used. Let ν be the largest integer $\leq \log_2 n$ and $\ell_1 = (\nu - 1)2^{\nu+2} + \nu + 14m + 12n + 6$. Then

$\ell \geq \ell_1$ if IORD = 2.

$\ell \geq \ell_1 + mn$ if IORD = 4.

When the routine terminates, ℓ will have been reset to the actual amount of storage needed.

IND is a variable that reports the status of the results. When SEPDE terminates, IND has one of the following values:

IND = 0 The solution U was obtained.

IND = −1 A constant (which is stored in $W(1)$) was subtracted from the right hand side of the equation in order to obtain a solution U. The solution is a weighted minimal least squares solution of the original problem.

IND = 1 (Input error) $a_1 \geq a_2$ or $b_1 \geq b_2$.

IND = 2 (Input error) ITYPE(i) $\neq 0, \pm 1$ for some edge i.

IND = 4 The approximating linear system of equations is not diagonally dominant. This cannot occur when m and n are sufficiently large. Increase m and n, and reset ℓ.

IND = 5 (Input error) ku $< m$.

IND = 6 (Input error) $m < 7$.

IND = 7 (Input error) $n < 6$.

IND = 8 (Input error) IORD $\neq 2, 4$.

IND = 10 (Input error) $a(x)d(y) \leq 0$ for some interior point (x, y) of the rectangle. This violates the assumption that the equation is elliptic.

IND = 11 (Input error) ℓ was too small. ℓ has been reset to the minimum amount of storage needed for W.

IND = 12 (Input error) ITYPE(i) = -1 for edge 1 or 3, but not for both edges.
IND = 13 (Input error) ITYPE(i) = -1 for edge 2 or 4, but not for both edges.

Precision. If IORD = 2 then the elliptic equation is approximated by a set of linear equations using finite differences. Otherwise, if IORD = 4 then the approximating equations are obtained by deferred corrections. The most accuracy is achieved when ITYPE(i) = 1 boundary conditions are not involved. For $m, n \geq 100$, 3–4 digit accuracy may be attained when IORD = 2 and 7–8 digit accuracy when IORD = 4. When ITYPE(i) = 1 boundary conditions are used, then for $m, n \geq 100$, 2–3 digits may be attained when IORD = 2 and 5–6 digits when IORD = 4.

Programming. SEPDE is an interface by A. H. Morris for SEPELL, a modification of the routine SEPELI described in the reference. SEPELI was developed by John C. Adams, being supported (in part) by codes written by Paul Swarztrauber and Roland Sweet (National Center for Atmospheric Research, Boulder, Colorado). SEPDE employs the subroutines PDEDGE, SEPELL, SEPEL1, CHKPRM, CHKSNG, ORTHG, MINSOL, TRISP, DEFER, DXFN, DYFN, BLKTRI, BLKTR1, COMPB, PROD0, PRODP, CPROD0, CPRODP, INDXA, INDXB, INDXC, PPADD, TQLRT0 and functions PSGF, BSRH, PPSGF, PPSPF, SPMPAR. The routines exchange information in the labeled common blocks having block names CBLKT and SPLP.

Example. Consider $(1+x)^2 u_{xx} - 2(1+x)u_x + u_{yy} = 3(1+x)^4 \sin y$ for $0 \leq x \leq 1$ and $|y| \leq \pi$

where $u_x(0, y) = 4 \sin y \quad |y| \leq \pi$
$u(1, y) = 16 \sin y$

and u is periodic in y. This problem has the solution $u = (1 + x)^4 \sin y$. Let

```
ITYPE(1) = -1
ITYPE(2) =  1
ITYPE(3) = -1
ITYPE(4) =  0.
```

Then the following routines and functions may be used for describing the problem. (Here g = GVAL.)

```
      SUBROUTINE COFX (X,A,B,C)
      T = 1.0 + X
      A = T*T
      B = -2.0*T
      C = 0.0
      RETURN
      END

      SUBROUTINE COFY (Y,D,E,F)
```

```
      D = 1.0
      E = 0.0
      F = 0.0
      RETURN
      END

      REAL FUNCTION GVAL (X,Y)
      GVAL = 3.0*(1.0 + X)**4*SIN(Y)
      RETURN
      END

      REAL FUNCTION BVAL (I,X,Y)
      BVAL = 4.0*SIN(Y)
      IF (I .EQ. 4)BVAL = 4.0*BVAL
      RETURN
      END
```

COFX, COFY, GVAL, and BVAL must be declared in the calling program to be of type EXTERNAL.

Reference. Adams, J., Swarztrauber, P., and Sweet, R., ***FISHPAK: Efficient FORTRAN Subprograms for the Solution of Separable Elliptic Partial Differential Equations, Version*** 3. National Center for Atmospheric Research, Boulder, Colorado, 1978.

UNIFORM RANDOM SELECTION OF VALUES FROM A FINITE SET OF INTEGERS

Given the integers i_a and i_b where $i_a < i_b$. Then the following subroutine is available for selecting integers i where $i_a \le i \le i_b$. The selection is performed so that any integer i is equally likely to occur with probability $1/m$ where $m = i_b - i_a + 1$. Any integer i may be selected more than once. When the number of selected values become large, then the mean and variance of the values approximate the mean and variance of the discrete uniform distribution where

$$\text{mean} = i_a + (i_b - i_a)/2$$
$$\text{variance} = (m^2 - 1)/12.$$

CALL URGET(n, i_a, i_b, ix, L,IERR)

The argument n is the number of values to be selected, L is an array of dimension n or larger, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of values. It is assumed that $1 \le ix < 2^{31} - 1$. When URGET is called, if no input errors are detected then IERR is set to 0 and n values are stored in L. On output, ix is a new seed for selecting more values.

Error Return. IERR = 1 if $n \le 0$, IERR = 2 if $i_a \ge i_b$ or $m \ge 2^{31} - 1$, and IERR = 3 if ix is not a proper seed.

Usage. A given seed always initiates the same sequence of values. Thus, the following two sets of instructions

```
(1) IX = 7
    CALL URGET(30,0,1,IX,L,IERR)

(2) IX = 7
    CALL URGET(20,0,1,IX,L,IERR)
    CALL URGET(10,0,1,IX,L(21),IERR)
```

generate the same 30 values (0 and 1).

Remarks.

(1) The n values selected need be distinct only when $m = 2^{31} - 2$ and $n = m$.
(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \le 2^{31} - 1$.

Programming. Originally formulated by John R. Crigler. Written by A. H. Morris.

UNIFORM RANDOM NUMBER GENERATOR

The following subroutines are available for generating a sequence of uniform variates in the open interval $(0,1)$.

CALL URNG(ix,A,n,IERR)
CALL DURNG(ix,A,n,IERR)

A is an array of dimensiion n or larger where $n \geq 1$. URNG is used if A is a real array and DURNG is used if A is a double precision array.

The argument n is the number of variates to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When URNG or DURNG is called, if no input errors are detected then IERR is set to 0 and n uniform variates are stored in A. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$ and IERR = 2 if ix is not a proper seed.

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 103
    CALL URNG(IX,A,30,IERR)

(2) IX = 103
    CALL URNG(IX,A,20,IERR)
    CALL URNG(IX,A(21),10,IERR)
```

generate the same 30 variates.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. Written by Linus Schrage (University of Chicago). Adapted by A.H. Morris.

Reference. Schrage, Linus,"A More Portable Fortran Random Number Generator," ***ACM Trans. Math Software*** **5** (1979), pp. 132-138.

GENERATING POINTS UNIFORMLY IN A SQUARE

The following subroutines are available for generating points $(x_1, y_1), \ldots, (x_n, y_n)$ uniformly where $0 < x_i < 1$ and $0 < y_i < 1$ for each i.

CALL URNG2(ix,X,Y,n,IERR)
CALL DURNG2(ix,X,Y,n,IERR)

X and Y are arrays of dimension n or larger where $n \geq 1$. URNG2 is used if X and Y are real arrays, and DURNG2 is used if X and Y are double precision arrays.

The argument n is the number of points to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of points. It is assumed that $1 \leq ix < 2^{31} - 1$. When URNG2 or DURNG2 is called, if no input errors are detected then IERR is set to 0 and n points are generated. The abscissas $x_1, \ldots, x_n$ of the points are stored in X and the ordinates $y_1, \ldots, y_n$ are stored in Y. On output, ix is a new seed for generating more points.

Error Return. IERR = 1 if $n \leq 0$ and IERR = 2 if ix is not a proper seed.

Usage. A given seed always initiates the same set of points. Thus, the following two sets of instructions

```
(1) IX = 4
    CALL URNG2(IX,X,Y,30,IERR)

(2) IX = 4
    CALL URNG2(IX,X,Y,20,IERR)
    CALL URNG2(IX,X(21),Y(21),10,IERR)
```

generate the same 30 points.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. Written by Linus Schrage (University of Chicago). Adapted by A.H. Morris.

Reference. Schrage, Linus,"A More Portable Fortran Random Number Generator," ***ACM Trans. Math Software*** **5** (1979), pp. 132–138.

GENERATING POINTS UNIFORMLY IN A CIRCLE

The following subroutines are available for generating points $(x_1, y_1), \ldots, (x_n, y_n)$ uniformly where $x_i^2 + y_i^2 < 1$ for each i.

CALL RCIR(n,ix,X,Y,IERR)
CALL DRCIR(n,ix,X,Y,IERR)

X and Y are arrays of dimension n or larger where $n \geq 1$. RCIR is used if X and Y are real arrays, and DRCIR is used if X and Y are double precision arrays.

The argument n is the number of points to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of points. It is assumed that $1 \leq ix < 2^{31} - 1$. When RCIR or DRCIR is called, if no input errors are detected then IERR is set to 0 and n points are generated. The abscissas $x_1, \ldots, x_n$ of the points are stored in X and the ordinates $y_1, \ldots, y_n$ are stored in Y. On output, ix is a new seed for generating more points.

Error Return. IERR = 1 if $n \leq 0$ and IERR = 2 if ix is not a proper seed.

Usage. A given seed always initiates the same set of points. Thus, the following two sets of instructions

```
(1) IX = 7
    CALL RCIR(30,IX,X,Y,IERR)

(2) IX = 7
    CALL RCIR(20,IX,X,Y,IERR)
    CALL RCIR(10,IX,X(21),Y(21),IERR)
```

generate the same 30 points.

Remarks.

(1) RCIR and DRCIR never generate the point (0,0)
(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Algorithm. Points are generated uniformly in the square $\{(x, y) : |x| < 1 \text{ and } |y| < 1\}$. The procedure terminates when n points have been obtained which lie in the unit circle.

Programming. RCIR calls the subroutine RCIR1, and DRCIR calls the subroutine DRCIR1. These routines were written by A. H. Morris.

NORMAL RANDOM NUMBER GENERATOR

Consider the normal distribution having the distribution function

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$$

for all real x. This distribution has mean 0 and standard deviation 1. The following subroutines are available for generating normal variates from this distribution.

CALL RNOR(ix,A,n,IERR)
CALL DRNOR(ix,A,n,IERR)

A is an array of dimension n or larger where $n \geq 1$. A is a real array if RNOR is used, and A is a double precision array if DRNOR is used.

The argument n is the number of variates to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When RNOR or DRNOR is called, if no input errors are detected then IERR is set to 0 and n variates are stored in A. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$ and IERR = 2 if ix is not a proper seed.

Algorithm. The polar algorithm is used.

Usage. Normal variates are generated in pairs. If n is odd then the last variate a_{n+1} generated is not stored. Also, a given seed always initiates the same sequence of variates. Thus, if we consider the following three sets of instructions

(1) IX = 5
CALL RNOR(IX,A,30,IERR)

(2) IX = 5
CALL RNOR(IX,A,10,IERR)
CALL RNOR(IX,A(11),20,IERR)

(3) IX = 5
CALL RNOR(IX,A,9,IERR)
CALL RNOR(IX,A(10),20,IERR)

then we note that (1) and (2) generate the same 30 normal variates. Also (3) generates 29 of these 30 variates, skipping the 10^{th} variate. The reason for this is that the request in (3) for 9 variates requires 10 variates to be generated, only 9 of which are used.

Remarks.

(1) If $a_1, \ldots, a_n$ are variates from RNOR and $v_i = x_0 + \sigma a_i$ for $i = 1, \ldots, n$ and $\sigma > 0$, then $v_1, \ldots, v_n$ are variates from a normal distribution with mean x_0 and standard deviation σ.

(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. RNOR calls the subroutine RCIR1, and DRNOR calls the subroutine DRCIR1. These routines are written by A. H. Morris.

Reference. Marsaglia, G. and Bray, T.A.,"A Convenient Method for Generating Normal Variables," ***SIAM Review*** **6** (1964), pp. 260–264.

CALL NRNG(ix,A,n,IERR)
CALL DNRNG(ix,A,n,IERR)

A is an array of dimension n or larger where $n \geq 1$. A is a real array if NRNG is used and A is a double precision array if DNRNG is used.

The argument n is the number of variates to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When NRNG or DNRNG is called, if no input errors are detected then IERR is set to 0 and n variates are stored in A. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$ and IERR = 2 if ix is not a proper seed.

Algorithm. The Box–Muller algorithm is used.

Usage. Normal variates are generated in pairs. The usage of NRNG and DNRNG is identical to the usage of RNOR and DRNOR.

Remarks.

(1) NRNG and DNRNG require 15–25% more time than RNOR and DRNOR.

(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. NRNG calls the subroutine URNG, and DNRNG calls the subroutine DURNG. NRNG and DNRNG were written by A. H. Morris.

Reference. Box, G.E.P. and Muller, M.E.,"A Note on the Generation of Random Normal Deviates," ***Annals of Math Statistics*** **29** (1950), pp. 610–611.

MULTIVARIATE NORMAL RANDOM VECTOR GENERATOR

Let A be a real symmetric positive definite matrix of order m, $\det(A)$ be the determinant of A (which must be positive), and $\mu = (\mu_1, \ldots, \mu_m)$. Consider the multivariate normal distribution having the density function

$$f(x) = \frac{1}{\sqrt{(2\pi)^m \det(A)}} e^{-1/2(x-\mu)A^{-1}(x-\mu)^t}$$

for $x = (x_1, \ldots, x_m)$ where all x_i are real. This distribution has the covariance matrix A and mean vector μ. The subroutines NRVG and DNRVG are available for generating variates x from this distribution when A is an arbitrary positive definite matrix. If A is a diagonal matrix then the subroutines NRVG1 and DNRVG1 are also available for generating variates from the distribution.

CALL NRVG(MO,ix,n,m,A,μ,X,kx,IERR)
CALL DNRVG(MO,ix,n,m,A,μ,X,kx,IERR)

It is assumed that $m \geq 2$. A is an array of dimension $m(m+1)/2$ or larger containing the positive definite matrix A in packed form,[1] and μ is an array of length m containing the mean vector of the distribution.

The argument n is the number of vectors to be generated, and X is a 2-dimensional array of dimension $kx \times n$. It is assumed that $kx \geq m$.

NRVG is used if A, μ, and X are real arrays, and DNRVG is used if A, μ, and X are double precision arrays.

MO is an argument which specifies if NRVG or DNRVG is being called for the first time. On an initial call, MO = 0 and we have the following setup:

The arguments ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of vectors. It is assumed that $1 \leq ix < 2^{31} - 1$. When NRVG or DNRVG is called, if no input errors are detected then IERR is set to 0 and n vectors x are stored in X. The j^{th} vector is stored in the j^{th} column of X. On output, ix is a new seed for generating more vectors.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $m \leq 1$ or $m > kx$, IERR = 3 if A is not a positive definite matrix, and IERR = 4 if ix is not a proper seed.

On an initial call to NRVG or DNRVG, the lower triangular matrix in the Cholesky decomposition of A replaces the original data in A. If IERR = 0 then the routine may be recalled with MO $\neq$ 0 to generate more vectors. If MO $\neq$ 0 then it is assumed that A and m have not been modified by the user. However, n, μ, X, and kx may be different.

[1] See the section on packed symmetric matrices and the subroutines MCVFS, MCVSF, DMCVFS, and DMCVSF.

The routine employs the lower triangular matrix obtained on the initial call to NRVG or DNRVG to generate n new vectors. As before, if no input errors are detected then IERR is set to 0 and the new vectors are stored in X.

Usage. A given seed always initiates the same set of vectors. Thus, the following two sets of instructions

```
(1) IX = 3
    CALL NRVG (0,IX,30,M,A,X0,X,KX,IERR)
    IF (IERR .NE. 0) STOP

(2) IX = 3
    CALL NRVG (0,IX,20,M,A,X0,X,KX,IERR)
    IF (IERR .NE. 0) STOP
    CALL NRVG (1,IX,10,M,A,X0,X(1,21),KX,IERR)
```

generate the same 30 vectors.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Algorithm. The lower triangular matrix L in the Cholesky decomposition of A is computed, and n vectors $z = (z_1, \ldots, z_m)$ are generated where the values z_i are independent variates from the normal distribution with mean 0 and variance 1. The n variates $x = Lz + \mu$ of the multivariate normal distribution are then computed.

Programming. NRVG employs the subroutines RNOR, RCIR1, SPPFA, and function SDOT, and DNRVG employs the subroutines DRNOR, DRCIR1, DPPFA, and function DDOT. NRVG and DNRVG were written by A. H. Morris.

Reference. Anderson, T.W., ***An Introduction to Multivariate Statistical Analysis***, John Wiley and Sons, New York, 1958, pp. 5–27.

CALL NRVG1(MO,ix,n,m,A,μ,X,kx,IERR)
CALL DNRVG1(MO,ix,n,m,A,μ,X,kx,IERR)

It is assumed that $m \geq 2$, and that A is a diagonal matrix whose diagonal elements are positive numbers. A is an array of dimension m or larger containing the diagonal elements. and μ is an array of dimension m containing the mean vector of the distribution.

The argument n is the number of vectors to be generated, and X is a 2–dimensional array of dimension $kx \times n$. It is assumed that $kx \geq m$.

NRVG1 is used if A, μ, and X are real arrays, and DNRVG1 is used if A, μ, and X are double precision arrays.

MO is an argument which specifies if NRVG1 or DNRVG1 is being called for the first

time. On an initial call, MO = 0 and we have the following setup:

The argument ix and IERR are variables. On input, ix is an integer (called a *seed*) for initializing the sequence of vectors. It is assumed that $1 \leq ix < 2^{31} - 1$. When NRVG1 or DNRVG1 is called, if no input errors are detected then IERR is set to 0 and n vectors x are stored in X. The j^{th} vector is stored in the j^{th} column of X. On output, ix is a new seed for generating more vectors.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $m \leq 1$ or $m > kx$, IERR = 3 if $A(i) \leq 0$ for some i, and IERR = 4 if ix is not a proper seed.

On an initial call to NRVG1 or DNRVG1, the square roots of the diagonal elements of the matrix are stored in A. If IERR = 0 then the routine may be recalled with MO $\neq 0$ to generate more vectors. If MO $\neq 0$ then it is assumed that A and m have not been modified by the user. However, n, μ, X, and kx may be different. The routine employs the roots obtained on the initial call to NRVG1 or DNRVG1 to generate n new vectors. As before, if no input errors are detected then IERR is set to 0 and the new vectors are stored in X.

Usage. The usage of NRVG1 and DNRVG1 is identical to the usage of NRVG and DNRVG.

Remarks.

(1) NRVG and NRVG1 generate the same vectors when A is a diagonal matrix.
(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. NRVG1 employs the subroutines RNOR and RCIR1, and DNRVG1 employs the subroutines DRNOR and DRCIR1. NRVG1 and DNRVG1 were written by A. H. Morris.

EXPONENTIAL RANDOM NUMBER GENERATOR

For $a > 0$ consider the exponential distribution having the distribution function

$$F(x) = \frac{1}{a}\int_0^x e^{-t/a}\,dt = 1 - e^{-x/a} \quad (x \geq 0).$$

This distribution has the mean a and variance a^2. The following subroutines are available for generating variates from this distribution.

CALL RANEXP(n, a, ix, X,IERR)
CALL DRNEXP(n, a, ix, X,IERR)

X is an array of dimension n or larger where $n \geq 1$. The argument a is a real number and X a real array if RANEXP is used. Otherwise, a is a double precision number and X a double precision array if DRNEXP is used.

The argument n is the number of variates to be generated, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When RANEXP or DRNEXP is called, if no input errors are detected then IERR is set to 0 and n exponential variates are stored in X. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $a \leq 0$, and IERR = 3 if ix is not a proper seed.

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 3
    CALL RANEXP(30,A,IX,X,IERR)

(2) IX = 3
    CALL RANEXP(20,A,IX,X,IERR)
    CALL RANEXP(10,A,IX,X(21),IERR)
```

generate the same 30 variates.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. RANEXP calls the subroutine URNG, and DRNEXP calls the subroutine DURNG. RANEXP and DRNEXP were written by John R. Criggler. RANEXP has been obtained from the STATLIB library.

Reference. Thomas, M.A., Gemmill,G.W., and Crigler,J.R., ***STATLIB: NSWC Library of Statistical Programs and Subroutines***, Report NSWC TR 89-97, Naval Surface Warfare Center, Dahlgren, Va., 1989.

GAMMA RANDOM NUMBER GENERATOR AND THE CHI-SQUARE DISTRIBUTION

For $a > 0$ consider the gamma distribution having the distribution function

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt \qquad (x \geq 0),$$

and for $\nu > 0$ consider the chi-square distribution having the distribution function

$$F_\nu(x) = \frac{2^{-\nu/2}}{\Gamma(\nu/2)} \int_0^x e^{-t/2} t^{\nu/2-1}\, dt \qquad (x \geq 0).$$

Then $F_\nu(x) = P(\nu/2, x/2)$, so that only the gamma distribution need be computed. The following subroutines are available for generating variates from the gamma distribution when $a \geq 10^{-1}$.

CALL RGAM$(ix, a, n, X, \text{IERR})$
CALL DRGAM$(ix, a, n, X, \text{IERR})$

X is an array of dimension n or larger where $n \geq 1$. The argument a is a real number and X a real array if RGAM is used. Otherwise, a is a double precision number and X a double precision array if DRGAM is used.

It is assumed that $a \geq 0.1$ and that n is the number of variates to be generated. IERR and ix are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When RGAM or DRGAM is called, if no input errors are detected then IERR is set to 0 and n gamma variates are stored in X. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $a < 0.1$, and IERR = 3 if ix is not a proper seed.

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 3
    CALL RGAM(IX,A,30,X,IERR)

(2) IX = 3
    CALL RGAM(IX,A,20,X,IERR)
    CALL RGAM(IX,A,10,X(21),IERR)
```

generate the same 30 variates.

Warning. The double precision subroutine DRGAM can be quite slow. It is recommended that its speed be checked if thousands of variates are to be generated.

Remarks.

(1) If $a = \nu/2$ and $x_1, \ldots, x_n$ are variates for the gamma distribution given by $P(a, x)$ then $2x_1, \ldots, 2x_n$ are variates for the chi-square distribution given by $F_\nu(x)$.

(2) It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. RGAM calls the subroutines URNG0 and GAMINV, and DRGAM calls the subroutines DURNG0 and DGINV. RGAM and DRGAM were written by A. H. Morris.

BETA RANDOM NUMBER GENERATOR

For $a, b > 0$ consider the beta distribution having the distribution function

$$I_x(a,b) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}\,dt \qquad (0 \le x \le 1),$$

where $B(a,b)$ is the beta function. The following subroutines are available for generating variates from this distribution when $a, b \ge 1/4$

CALL RBETA$(n, a, b, ix, X, \text{IERR})$
CALL DRBETA$(n, a, b, ix, X, \text{IERR})$

X is an array of dimension n or larger where $n \ge 1$. The arguments a and b are real numbers and X a real array if RBETA is used. Otherwise, a and b are double precision numbers and X a double precision array if DRBETA is used.

It is assumed that $a, b \ge 1/4$ and that n is the number of variates to be generated. IERR and ix are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \le ix < 2^{31} - 1$. When RBETA or DRBETA is called, if no input errors are detected then IERR is set to 0 and n beta variates are stored in X. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \le 0$, IERR = 2 if $a < 1/4$ or $b < 1/4$, and IERR = 3 if ix is not a proper seed.

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 7
    CALL RBETA (30, A, B, IX, X, IERR)

(2) IX = 7
    CALL RBETA (20, A, B, IX, X, IERR)
    CALL RBETA (10, A, B, IX, X(21), IERR)
```

generate the same 30 variates.

Warning. The double precision subroutine DRBETA can be quite slow. It is recommended that its speed be checked if thousands of variates are to be generated.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| < 2^{31} - 1$.

Programming. RBETA calls the subroutines URNG0 and GAMINV, and DRBETA calls the subroutines DURNG0 and DGINV. RBETA and DRBETA were written by A. H. Morris.

F-DISTRIBUTION RANDOM NUMBER GENERATOR

For $a, b > 0$ consider the F-distribution having the distribution function

$$P_{ab}(F) = \frac{a^{a/2}b^{b/2}}{B(a/2, b/2)} \int_0^F t^{(a/2)-1}(b + at)^{-(a+b)/2}\, dt \qquad (F \geq 0)$$

where $B(a/2, b/2)$ is the beta function. The values a and b are called the ***numerator and denominator degrees of freedom*** of this distribution. The value $F = (bs)/(at)$ is a variate from this distribution when s and t are independent variates from the gamma distributions with parameters $a/2$ and $b/2$ respectively. The following subroutines are available for generating variates from the F-distribution when $a, b \geq 1/2$.

CALL FRAN$(n, a, b, ix, X, \text{IERR})$
CALL DFRAN$(n, a, b, ix, X, \text{IERR})$

X is an array of dimension n or larger where $n \geq 1$. The arguments a and b are real numbers and X a real array if FRAN is used. Otherwise, a and b are double precision numbers and X a double precision array if DFRAN is used.

It is assumed that $a, b \geq 1/2$ and that n is the number of variates to be generated. IERR and ix are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When FRAN or DFRAN is called, if no input errors are detected then IERR is set to 0 and n variates are stored in X. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $a < 1/2$ or $b < 1/2$, IERR = 3 if b is too small for the floating arithmetic being used, and IERR = 4 if ix is not a proper seed. IERR = 3 can occur only if the floating arithmetic has a exceedingly limited range (say, its largest value is less than 10^{37}).

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 3
    CALL FRAN (30, A, B, IX, X, IERR)

(2) IX = 3
    CALL FRAN (20, A, B, IX, X, IERR)
    CALL FRAN (10, A, B, IX, X(21), IERR)
```

generate the same 30 variates

Warning. The double precision subroutine DFRAN can be quite slow. It is recommended that its speed be checked if thousands of variates are to be generated.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. FRAN calls the subroutines URNG0 and GAMINV, and DFRAN calls the subroutines DURNG0 and DGINV. FRAN and DFRAN were written by A. H. Morris.

STUDENT t-DISTRIBUTION RANDOM NUMBER GENERATOR

For $a > 0$ consider the student t-distribution having the distribution function

$$P_a(t) = [\sqrt{a}\, B(1/2, a/2)]^{-1} \int_{-\infty}^{t} (1 + x^2/a)^{-(a+1)/2}\, dx$$

where $B(1/2, a/2)$ is the beta function. The value $t = z/\sqrt{w/a}$ is a variate from this distribution when z is a variate from the normal distribution with mean 0 and variance 1, and w is an independent variate from the chi-square distribution with a degrees of freedom. The following subroutines are available for generating variates from the student t-distribution when $a \geq 1/2$.

CALL TRAN$(n, a, ix, X, \text{IERR})$
CALL DTRAN$(n, a, ix, X, \text{IERR})$

X is an array of dimension n or larger where $n \geq 1$. The argument a is a real number and X a real array if TRAN is used. Otherwise, a is a double precision number and X a double precision array if DTRAN is used.

It is assumed that $a \geq 1/2$ and that n is the number of variates to be generated. IERR and ix are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leq ix < 2^{31} - 1$. When TRAN or DTRAN is called, if no input errors are detected then IERR is set to 0 and n variates are stored in X. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leq 0$, IERR = 2 if $a < 1/2$, IERR = 3 if a is too small for the floating arithmetic being used, and IERR = 4 if ix is not a proper seed. IERR = 3 can occur only if the floating arithmetic has a exceedingly limited range (say, its smallest positive value is greater than 10^{-37}).

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1) IX = 3
    CALL TRAN (30, A, IX, X, IERR)

(2) IX = 3
    CALL TRAN (20, A, IX, X, IERR)
    CALL TRAN (10, A, IX, X(21), IERR)
```

generate the same 30 variates.

Warning. The double precision subroutine DTRAN can be quite slow. It is recommended that its speed be checked if thousands of variates are to be generated.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. TRAN calls the subroutines URNG0, URNG1, PNI, and GAMINV, and DTRAN calls the routines DURNG0, DURNG1, DPNI, and DGINV. TRAN and DTRAN were written by A. H. Morris.

FIRST ORDER MARKOV RANDOM NUMBER GENERATOR

Given a sequence of independent variates $z_1, \ldots, z_n$ $(n \geq 2)$ from a normal distribution with mean 0 and standard deviation $\sigma \geq 0$. For any real x_0 and $|\alpha| < 1$ let

$$(*) \qquad \begin{aligned} a_1 &= x_0 + z_1/\sqrt{1-\alpha^2} \\ a_j &= x_0 + \alpha(a_{j-1} - x_0) + z_j \qquad (j = 2, \ldots, n). \end{aligned}$$

Then $a_1, \ldots, a_n$ are variates from a normal distribution with mean x_0 and variance $\sigma^2/(1-\alpha^2)$. Since $z_1, \ldots, z_n$ are independent variates, a_j $(j \geq 2)$ is considered to depend only on a_{j-1} for information from the past. Hence, $a_1, \ldots, a_n$ forms a first order Markov (i.e, first order autoregressive) process. Since the process is covariance stationary, the correlation between terms a_j and a_k of the sequence becomes progressively weaker when $|k-j|$ increases.

For any x_0, σ, α, and n the following subroutines are available for obtaining sequences $a_1, \ldots, a_n$ of variates from $(*)$.

CALL RMK1$(n, x_0, \sigma, \alpha, ix, A, \text{IERR})$
CALL DRMK1$(n, x_0, \sigma, \alpha, ix, A, \text{IERR})$

The arguments x_0, σ, and α are real numbers and A a real array if RMK1 is used. Otherwise, x_0, σ, and α are double precision numbers and A a double precision array when DRMK1 is used.

A is an array of dimension n or larger when $n \geq 2$, and ix and IERR are variables. It is assumed that $\sigma \geq 0$ and $|\alpha| < 1$. On input, ix is an integer (called a *seed*) for initializing the sequence of variates $z_1, \ldots, z_n$. It is assumed that $1 \leq ix < 2^{31} - 1$. When RMK1 or DRMK1 is called, if no input errors are detected then IERR is set to 0 and a sequence $a_1, \ldots, a_n$ is obtained and stored in A. On output, ix is a new seed for generating other sequences $a_1, \ldots, a_n$.

Error Return. IERR is set to one of the following values if an input error is detected.

IERR = 1 if $n < 2$
IERR = 2 if $\sigma < 0$
IERR = 3 if $|\alpha| \geq 1$
IERR = 4 if ix is not a proper seed.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leq 2^{31} - 1$.

Programming. RMK1 employs the subroutines RNOR and RCIR1, and DRMK1 employs the subroutines DRNOR and DRCIR1. RMK1 was written by John R. Crigler and modified by A. H. Morris. RMK1 is a modification of the subroutine RANMK1, which resides in the STATLIB library. DRMK1 was adapted from RMK1 by A. H. Morris.

Reference. Thomas, M.A., Gemmill, G.W., and Crigler, J.R., ***STATLIB: NSWC Library of Statistical Programs and Subroutines***, Report NSWC TR 89-37, Naval Surface Warfare Center, Dahlgren, Virginia, 1989.

APPENDIX

Installation Of The NSWC Library And Conversion Of Codes From Single To Double Precision Form

Code for the library can be obtained on 9-track tape and on $5\frac{1}{4}$ inch disks that can be read by the IBM PC. Two copies of the code are given on a tape.

The first function in the library, namely IPMPAR, must be modified for the particular Fortran being used. IPMPAR provides the integer constants which characterize the integer, single precision, and double precision arithmetics being used (see pp. 3–4).

> Instructions are given in the in-line documentation of IPMPAR for defining the constants that are needed. If constants are not provided for the compiler being used, then the Fortran manual for the compiler normally gives the constants for the integer arithmetic. However, help may be needed in obtaining the constants for the single and double precision arithmetics. The subroutines MACH and RADIX are provided for this purpose.
>
> MACH and RADIX, and their subroutines MACH1, STORE2, MACH2, DSTOR2 are the next subprograms after IPMPAR. Instructions for the use of MACH and RADIX are given in MACH. These subroutines are experimental. They are provided only as an aid for obtaining the constants for the single and double precision arithmetics. They are not used by any of the functions and subroutines in the NSWC library, and they are not considered to be part of the library. MACH1 and MACH2 perform some writing. None of the functions and subroutines in the NSWC library perform I/O.

After IPMPAR has been defined, the remainder of the library can be installed. None of the remaining code needs any modification. Codes for the functions and subroutines appear approximately in the order that they appear in the manual.

Single/Double Precision Conversion. The following modifications must be made when a subroutine is converted from single to double precision form.

(1) Replace the functions SPMPAR, EPSLN, and EXPARG with DPMPAR, DEPSLN, and DXPARG. Do not modify the arguments of these functions. DPMPAR, DEPSLN, and DXPARG must be declared to be of type DOUBLE PRECISION when they are employed.
(2) If the function IPMPAR appears in the subroutine then replace IPMPAR(5) with IPMPAR(8), IPMPAR(6) with IPMPAR(9), and IPMPAR(7) with IPMPAR(10).

INDEX

ABSLV ... 245
ADAPT ... 435
AERF ... 53
AI ... 100
AIE ... 100
ALI ... 74
ALNREL ... 31
ANG ... 23
AORD ... 7
ARCECO ... 223
ARCESL ... 224
ARTNQ ... 11
ASSGN ... 417
BADD ... 269
BCVDR ... 261
BCVRC ... 265
BCVRD ... 261
BESI ... 95
BESJ ... 92
BETALN ... 87
BFIT ... 503
BI ... 101
BIE ... 101
BIMAG ... 263
BLKORD ... 333
BLSQ ... 403
BNRM ... 283
BPOSE ... 267
BPROD ... 273
BRATIO ... 89
BRCOMP ... 90
BREAL ... 263
BRKF45 ... 555
BSLSQ ... 501
BSLSQ2 ... 519
BSLV ... 285
BSLV1 ... 285
BSPP ... 495
BSSLI ... 95
BSSLJ ... 91
BSSLK ... 98
BSSLY ... 93
BSTRP ... 499
BSTRP2 ... 517
BSUBT ... 271
BTPRD ... 275
BTPRD1 ... 276
BTSLV ... 225
BVAL ... 491
BVALI ... 493
BVAL2 ... 521
BVPRD ... 275
BVPRD1 ... 275
B1CND ... 287
B1NRM ... 281
CAI ... 99
CAPO ... 15
CAXPY ... 175
CBADD ... 269
CBCRT ... 155
CBESK ... 97
CBI ... 99
CBND ... 159
CBPOSE ... 267
CBPROD ... 273
CBRT ... 11
CBSLV ... 293
CBSLV1 ... 293
CBSPL ... 455
CBSSLE ... 95
CBSSLJ ... 91
CBSSLK ... 98
CBSUBT ... 271
CBTPD ... 279
CBTPD1 ... 280
CBVPD ... 279
CBVPD1 ... 279
CCOPY ... 161
CDET ... 243
CDIV ... 13
CDIVID ... 13
CDOTC ... 171
CDOTU ... 171
CEIG ... 367
CEIGV ... 369
CERF ... 45
CERFC ... 46
CEXEXI ... 71
CEXPLI ... 65
CFRMLT ... 61
CGAMMA ... 75
CHPDV ... 433
CIN ... 69
CIRCV ... 125
CK ... 103
CKF ... 104
CKPROD ... 211
CLE ... 253
CLI ... 73
CL1 ... 375
CMADD ... 200
CMADJ ... 197
CMCONJ ... 193
CMCOPY ... 191
CMCVBS ... 259
CMCVSB ... 259
CMIMAG ... 189
CMPROD ... 203
CMREAL ... 189
CMSLV ... 235
CMSLV1 ... 236
CMSUBT ... 202
CMTMS ... 203
CONSTR ... 21
COSQB ... 429
COSQF ... 430
COSQI ... 429
COS1 ... 25
CPABS ... 11
CPOSE ... 313
CPOSE1 ... 313
CPSC ... 439
CPSI ... 79
CREC ... 13
CROUT ... 215
CSADD ... 315
CSCAL ... 173
CSCOPY ... 305
CSEVL ... 145
CSIMAG ... 301
CSINT ... 465
CSINT1 ... 465
CSINT2 ... 465
CSLOOP ... 473
CSLV ... 348
CSLVMP ... 239
CSPFIT ... 459
CSPROD ... 319
CSPSLV ... 347
CSRCAL ... 301
CSRF ... 510
CSRF2 ... 510
CSROT ... 165
CSSCAL ... 173
CSSUBT ... 317
CSURF ... 509
CSURF1 ... 509
CSVDC ... 241
CSWAP ... 163
CTIP ... 195
CTPOSE ... 195
CTPRD ... 325
CTPRD1 ... 325
CTRANS ... 197
CTSLV ... 348
CUBTRI ... 543
CURVD ... 479
CURVI ... 479
CURV1 ... 475
CURV2 ... 477
CVBC ... 257
CVBD ... 257
CVBR ... 257
CVCB ... 257
CVCB1 ... 258
CVCS ... 297
CVDB ... 257
CVDB1 ... 258
CVDS ... 297
CVPRD ... 325
CVPRD1 ... 325
CVRB ... 257
CVRB1 ... 258
CVRS ... 297
CVSC ... 297
CVSD ... 297
CVSR ... 297
DABSLV ... 248
DAERF ... 51
DAORD ... 7
DARTNQ ... 11
DASUM ... 177
DAW ... 59
DAXPY ... 175
DBADD ... 269
DBETLN ... 87
DBNRM ... 283
DBPOSE ... 267
DBPROD ... 273
DBSLV ... 289
DBSLV1 ... 289
DBSUBT ... 271
DBTPD ... 277
DBTPD1 ... 278
DBVPD ... 277
DBVPD1 ... 277
DB1CND ... 291
DB1NRM ... 281
DCBCRT ... 155
DCBRT ... 11
DCEIG ... 371
DCEIGV ... 373
DCERF ... 47
DCERFC ... 48
DCGAMA ... 76
DCMSLV ... 237
DCOPY ... 161
DCOS1 ... 25
DCPABS ... 11
DCPOLY ... 157
DCPSC ... 440
DCPSI ... 80
DCREC ... 13
DCSEVL ... 145
DCSQRT ... 13
DDOT ... 171
DDSORT ... 8
DEI ... 66
DEIG ... 355
DEIGV ... 357
DEI1 ... 67
DELLPI ... 109
DEPI ... 112
DEPSLN ... 5
DERF ... 48
DERFC ... 48
DERFC1 ... 49
DERFI ... 51
DET ... 243
DFRAN ... 599
DGAMLN ... 77
DGAMMA ... 77
DGINV ... 86
DGRAT ... 84
DHFTI ... 387
DHFTI2 ... 387
DISORT ... 8
DKPROD ... 211
DLLSQ ... 385
DLNREL ... 32

DLSQR ... 386
DMADD ... 199
DMCOPY ... 191
DMCVBS ... 259
DMCVFS ... 183
DMCVSB ... 259
DMCVSF ... 183
DMEVAL ... 529
DMEXP ... 251
DMFIT ... 527
DMPROD ... 203
DMSLV ... 217
DMSLV1 ... 218
DMSUBT ... 201
DMTMS ... 203
DNRM2 ... 179
DNRNG ... 588
DNRVG ... 589
DNRVG1 ... 590
DORDER ... 8
DPADD ... 131
DPCHOL ... 231
DPCOPY ... 129
DPDAW ... 59
DPDET ... 243
DPDIV ... 137
DPINV ... 141
DPLE ... 253
DPLPWR ... 139
DPMPAR ... 4
DPMULT ... 135
DPNI ... 57
DPOSE ... 311
DPOSE1 ... 311
DPSI ... 80
DPSUBT ... 133
DQAGI ... 540
DQAGS ... 534
DQDCRT ... 155
DQICRT ... 155
DQXGS ... 536
DRBETA ... 597
DRCIR ... 585
DRCOMP ... 84
DRDVAL ... 110
DREXP ... 29
DRFVAL ... 109
DRGAM ... 595
DRJVAL ... 112
DRLOG ... 32
DRLOG1 ... 32
DRMK... ... 603
DRNE... ... 553
DRNK ... 213
DRNOR ... 587
DROT ... 165
DROTG ... 17
DROTM ... 169
DROTM... ... 168
DRPOL... ... 257
DSADD ... 315
DSCAL ... 173
DSCOPY ... 305
DSEIG ... 363
DSEIGV ... 365
DSIN1 ... 25
DSLV ... 342
DSMSLV ... 227
DSNRM ... 329
DSORT ... 7
DSPROD ... 319
DSPSLV ... 341
DSSUBT ... 317
DSVDC ... 241
DSVPRD ... 205
DSWAP ... 163
DS1CND ... 345
DS1NRM ... 327
DTASLV ... 247
DTIP ... 195
DTOPLX ... 233
DTPOSE ... 195
DTPRD ... 323
DTPRD1 ... 323
DTRAN ... 601
DTSLV ... 342
DURNG ... 581
DURNG2 ... 583
DVPRD ... 323
DVPRD1 ... 323
DXPARG ... 5
DZERO ... 151
EIG ... 351
EIGV ... 353
EIGV1 ... 353
EIG1 ... 351
ELLPF ... 115
ELLPI ... 107
ELPFC1 ... 116
EPI ... 111
EPSLN ... 5
ERF ... 46
ERFC ... 46
ERFC1 ... 46
ERFI ... 51
EXPARG ... 5
EXPLI ... 66
FFT ... 425
FFT1 ... 425
FMIN ... 407
FRAN ... 599
FRNL ... 63
GAMINV ... 85
GAMLN ... 76
GAMMA ... 75
GERK ... 563
GRADG ... 524
GRADL ... 524
GRATIO ... 83
HBRD ... 153
HC ... 41
HFTI ... 379
HFTI2 ... 379
HTRP ... 447
HULL ... 37
ICAMAX ... 181
IDAMAX ... 181
IESLV ... 545
INTRVL ... 489
IORDER ... 8
IPMPAR ... 4
ISAMAX ... 181
ISHELL ... 7
ISUBX ... 90
KPROD ... 211
KROUT ... 216
KURVP1 ... 483
KURVP2 ... 484
KURV1 ... 481
KURV2 ... 482
LAINV ... 421
LE ... 253
LGRNGN ... 147
LGRNGV ... 147
LGRNGX ... 147
LLSQ ... 375
LLSQMP ... 383
LMDIFF ... 411
LOCPT ... 33
LOPCMP ... 474
LOPDF ... 474
LSEI ... 391
LSQR ... 378
LTRP ... 445
L2SLV ... 333
MADD ... 199
MCOPY ... 191
MCVBS ... 259
MCVDR ... 185
MCVFS ... 183
MCVRC ... 187
MCVRD ... 185
MCVSB ... 259
MCVSF ... 183
MEVAL ... 529
MEXP ... 251
MFFT ... 427
MFFT1 ... 427
MFIT ... 527
MKF ... 419
MPLNMV ... 143
MPROD ... 203
MSLV ... 217
MSLV1 ... 218
MSUBT ... 201
MTMS ... 203
MTPRD ... 321
MTPRD1 ... 321
MVPRD ... 321
MVPRD1 ... 421
NPIVOT ... 216
NRNG ... 588
NRVG ... 589
NRVG1 ... 590
NSURF2 ... 515
ODE ... 551
OPTF ... 409
ORTHOS ... 149
ORTHOV ... 149
ORTHOX ... 149
PADD ... 131
PAREA ... 39
PCHOL ... 231
PCOEFF ... 449
PDFIT ... 469
PDIV ... 137
PDSPL ... 467
PEQ ... 119
PEQ1 ... 120
PFIND ... 35
PFIT ... 451
PINV ... 141
PKILL ... 127
PLCOPY ... 129
PLEM ... 120
PLEM1 ... 121
PLPWR ... 139
PMULT ... 135
PNDF ... 55
PNI ... 57
POCA ... 15
PPVAL ... 497
PSCMP ... 471
PSEVL ... 471
PSI ... 79
PSIDF ... 81
PSUBT ... 133
QAGI ... 539
QAGS ... 531
QDCRT ... 155
QSORTD ... 8
QSORTI ... 8
QSORTR ... 8
QSUBA ... 533
QTCRT ... 155
QURV1 ... 485
QURV2 ... 486
QXGS ... 532
RANEXP ... 593
RBETA ... 597
REND ... 159
RCIR ... 585
RCOMP ... 83
RDVAL ... 108
REXP ... 29
RFVAL ... 108
RGAM ... 595
RJ...ORT ... 7
RJVAL ... 111
RK ... 571

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 the 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 Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1993	

4. TITLE AND SUBTITLE

NSWC Library of Mathematics Subroutines

5. FUNDING NUMBERS

6. AUTHOR(S)

Alfred H. Morris, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Surface Warfare Center (Code B10)
Dahlgren Division
Dahlgren, Virginia 22448-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NSWCDD/TR-92/425

9. SPONSORING/MONITORING AGENCY NAME(S) AND

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The NSWC library is a library of general purpose Fortran subroutines that provide a basic computational capability for a variety of mathematical activities. Emphasis has been placed on the transportability of the codes. Subroutines are available in the following areas: elementary operations, geometry, special functions, polynomials, vectors, matrices, large dense systems of linear equations, banded matrices, sparse matrices, eigenvalues and eigenvectors, ℓ_1 solution of linear equations, least squares solution of linear equations, optimization, transforms, approximation of functions, curve fitting, surface fitting, manifold fitting, numerical integration, integral equations, ordinary differential equations, partial differential equations, and random number generation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	465
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

BLOCK 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If Known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with ...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2
NTIS	- Leave blank

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief ***(Maximum 200 words)*** factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code ***(NTIS only)***

Block 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of this page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

RKF45 ... 559
RK8 ... 573
RLOG ... 31
RLOG1 ... 31
RMK1 ... 603
RNK ... 213
RNOR ... 587
RORDER ... 8
ROTA ... 15
ROT3 ... 19
RPOSE ... 309
RPOSE1 ... 309
RSCOPY ... 305
RSLV ... 336
SADD ... 315
SASUM ... 177
SAXPY ... 175
SCASUM ... 177
SCNRM2 ... 179
SCOMP ... 461
SCOMP1 ... 462
SCOMP2 ... 462
SCONJ ... 307
SCOPY ... 161
SCVDR ... 299
SCVRC ... 303
SCVRD ... 299
SDOT ... 171
SEIG ... 359
SEIGV ... 361
SEIGV1 ... 361
SEIG1 ... 359
SEPDE ... 575
SEVAL ... 463
SEVAL1 ... 464
SEVAL2 ... 464
SFODE ... 567
SFODE1 ... 570
SFVAL ... 525
SFVAL2 ... 525
SHELL ... 7
SHELL2 ... 7
SI ... 69
SINQB ... 430
SINQF ... 430
SIN1 ... 25
SLVMP ... 221
SMADD ... 199
SMCOPY ... 191
SMPLX ... 413
SMPROD ... 209
SMSLV ... 227
SMSUBT ... 201
SNHCSH ... 27
SNRM ... 329
SNRM2 ... 179
SPFIT ... 457
SPFIT2 ... 507
SPLIFT ... 456
SPLSQ ... 405
SPMPAR ... 3
SPORD ... 331
SPROD ... 319
SPSLV ... 335
SQUINT ... 249
SROT ... 165
SROTG ... 17
SROTM ... 169
SROTMG ... 168
SSCAL ... 173
SSPLX ... 413
SSUBT ... 317
SSVDC ... 241
SSWAP ... 163
STLSQ ... 405
SURF ... 513
SURF2 ... 515
SVPRD ... 205
S1CND ... 339
S1NRM ... 327
TASLV ... 247
TIP ... 195
TMPROD ... 207
TOPLX ... 233
TPOSE ... 195
TRAN ... 601
TRMESH ... 523
TRP ... 443
TSLV ... 336
URGET ... 579
URNG ... 581
URNG2 ... 583
VALR2 ... 123
WNNLS ... 395
WPFIT ... 453
ZEROIN ... 151