AD-A051 536 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 9/2
MINOS SYSTEM MANUAL.(U)
DEC 77 M A SAUNDERS N00014-75-C-0865
UNCLASSIFIED SOL-77-31 ARO-12215.22-M NL

1 OF 2
AD
A051536

MINOS

SYSTEM MANUAL

BY

MICHAEL A. SAUNDERS

TECHNICAL REPORT SOL 77-31

DECEMBER 1977

Systems Optimization Laboratory

Department of Operations Research

Stanford University

Stanford California 94305

(6) MINOS

SYSTEM MANUAL

(1)

by

(10) Michael A. Saunders

(9) TECHNICAL REPORT, (14) SOL-77-31

(11) December 1977

(12) 140 p.

*** This manual documents MINOS Version 3.3, dated October 1977 ***

(15) N00014-75-C-0865, DAAG29-74-C-0034

(18) ARO (19) 12215.22-M

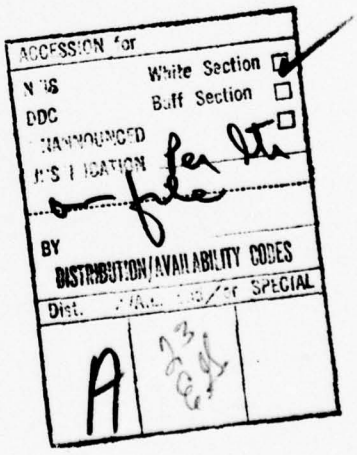
Research and reproduction of this report were partially supported by the Energy Research and Development Contract EY-76-S-03-0326 PA #18; the Office of Naval Research Contract NOOO14-75-C-0865; the National Science Foundation Grants MCS76-20019 and ENG77-06761; the Army Research Office Contract DAAG-29-74-C-0034; and by the New Zealand Department of Scientific and Industrial Research.

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH

Stanford University
Stanford, California

Abstract

MINOS is a Fortran system for solving large-scale linearly constrained optimization problems. The System Manual gives an overview of the system, the programming conventions used, data structures, tolerances, and error conditions. Details are given of a practical implementation of the method of Bartels and Golub for maintaining a sparse LU factorization. The reduced-gradient approach for handling a nonlinear objective function has been described elsewhere by Murtagh and Saunders; further implementation details are included here. The System Manual should facilitate interfacing of MINOS with other optimization software.



MINOS

SYSTEM MANUAL

C O N T E N T S

		PAGE
1.	INTRODUCTION	3
2.	OVERVIEW	5
	2.1 Storage Allocation	5
	2.2 Subroutine Hierarchy	6
3.	PROGRAMMING CONVENTIONS	8
	3.1 Conversion to Other Machines	10
4.	VARIABLES	12
	4.1 Variable-dimension Information	12
	4.2 COMMON Declarations	13
	4.3 Definition of COMMON Variables	14
	4.4 Arrays	19
5.	TOLERANCES	21
6.	DATA STRUCTURES	24
	6.1 Constraint Matrix [A b I]	24
	6.2 LU Factors of the Basis	26
	6.2.1 Storage for L	27
	6.2.2 Storage for U	32
	6.2.3 Storage for F	33
	6.3 Cholesky Factor of the Reduced Hessian	34
7.	INVERT STATISTICS	35
8.	INVERT REQUESTS	38
9.	ERROR CONDITIONS	39

10. SUBROUTINE SPECIFICATIONS 41

ADDCOL 42
ALIGN 43
BTRANL 44
BTRANU 45
BUMPS 46
CALCFG 47
CALCG 48
CHKDIR 49
CHKGRD 50
CG 51
CHUZQ 52
CHUZR 53
COMDFP 56
CRASH 57
DELCOL 59
DOT 60
DOT1 60
DRIVER 61
DUMPN 64
FACTOR 65
FORMC 68
FTRANL 70
FTRANU 71
FUNGRD 72
GETGRD 73
GO 74
HASH 76
INITLZ 77
INSERT 79
INVERT 80
ITEROP 82
LOADB 83
LOADN 85
LPITN 87
MINOS 89
MKLIST 91
MODLU 92
MOVE 94
MPS 95
MPSIN 98
NEWPTC 100
NMSRCH 101
PACKLU 102
PRICE 103
PUNCH 105
P3 106
P4 107
RESETR 109
RGITN 110
RTRSOL 113
R1ADD 114
R1MOD 115
R1PROD 116
R1SUB 117
SAVEB 118
SEARCH 119
SETPI 120
SETX 121
SOLN 122
SOLPRT 124
SPECS 125
SPECS2 127
SQUEEZ 128
STATE 129
TRNSVL 130
UNPACK 131

11. PERFORMANCE 132

12. ADDENDA TO MINOS USER'S GUIDE 134

REFERENCES 135

1. INTRODUCTION

MINOS is an in-core, Fortran-based optimization system for the solution of large-scale linear and nonlinear programming problems involving sparse linear constraints. Full user information is given in the MINOS User's Guide (Murtagh and Saunders, 1977). Theoretical aspects of the algorithm, and experience with an earlier version of the system, are described in Murtagh and Saunders (1978).

This manual provides details of the implementation methods used. Much of the complexity of the system is concerned with a reliable and efficient implementation of the primal simplex method, as described in Saunders (1976). The nonlinear programming features are an integral part of the design, and this required that most of the standard simplex routines be moderately generalized. However, the bulk of the code for nonlinear problems is contained in 24 additional subroutines.

Occasional reference is made to MINOS/GRG, an extension of the present system designed to handle problems with nonlinear constraints and a sparse Jacobian matrix (Jain, Lasdon and Saunders, 1976). This system will be documented elsewhere.

The primary goals throughout the coding of MINOS have been as follows (in alphabetical order).

Efficiency - Use of efficient algorithms; conservation of core requirements through dynamic storage allocation; avoidance of multi-dimensional arrays; sequential access to data where possible to avoid frequent page-swapping in a virtual memory environment.

Flexibility - Free-format input for user-defined parameters; ability to solve problems of arbitrary size without recompilation of the source program; various restart facilities; provision for interface with existing commercial systems.

Modularity - Well-defined functions for each subroutine; use of parameters to pass all array storage and dimension information.

Portability - Use of only those Fortran constructs that are commonly available on (large) machines; separation and internal documentation of routines that are necessarily machine-dependent; calculation of precision tolerances from one machine-dependent constant.

Readability - Indentation of loops as a bare minimum; meaningful names for variables; a reasonable amount of in-line documentation.

Reliability - Use of numerically stable algorithms; consistency checks on input data, restarting files and gradient computation.

There is always room for improvement in putting such goals into practice, but certainly some such aims are necessary if a system is to develop while the algorithms being implemented are themselves continually evolving.

2. OVERVIEW

2.1 Storage Allocation

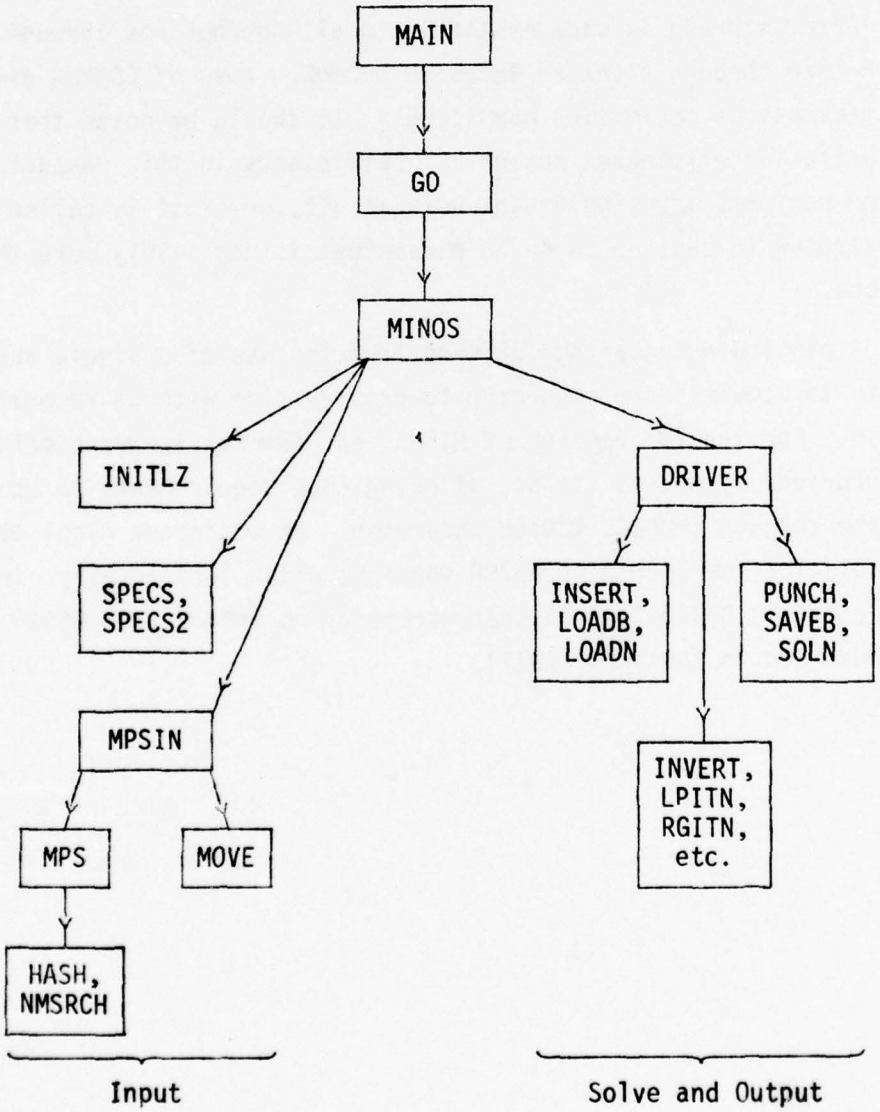
To ensure storage economy, a single array Z(*) is used to provide all array workspace. The type of Z must match the largest word size used (e.g. REAL*8 for IBM, REAL for CDC). Arrays of other types are allocated compactly within Z, using three constants to define the word sizes available on any particular machine.

Array storage is made available to all subroutines through parameters rather than through blank or labelled COMMON. (Use of COMMON diminishes the effectiveness of optimizing compilers.) It should be noted that sensible modularization eliminates potential inefficiency in this respect. If a subroutine performs any significant work at all, or of it is called infrequently, the overhead in passing 20 or 30 parameters is negligibly more than in passing just one.

A particular advantage arising from the use of a single array Z is the ability to acquire core according to problem size without recompiling the program. For the IBM version of MINOS, an assembly language main program has been provided by Hedges (1975), allowing core requirements to be defined at run-time through the JCL REGION parameter. An analogous Algol procedure is available for the Burroughs B6700 version, which incidentally circumvents the restriction in B6700 Fortran that arrays be no longer than 65535 words. This is documented in Saunders (1977).

2.2 Subroutine Hierarchy

The diagram below illustrates the broad grouping of subroutines. Once a problem has been input, the dimensions of most arrays are determined. Hence subroutine MINOS is able to pass many segments of the array Z to DRIVER, which thereafter knows them as separate arrays with meaningful names. A few other arrays for the LU file remain to be allocated by INVERT within the remaining part of Z.



A more complete classification of subroutines is given on the next page.

Input routines (for processing the SPECS and MPS files)

HASH	MOVE	MPS	MPSIN	SPECS	SPECS2

Basis loading routines

CRASH	INSERT	LOADB	LOADN	NMSRCH

Basis and solution saving routines

DUMP	PUNCH	SAVEB	SOLN	SOLPRT	STATE

Algorithmic routines required for purely linear programs

BTRANL	BTRANU	BUMPS	CHUZR	DRIVER	FACTOR
FORMC	FTRANL	FTRANU	GO	INITLZ	INVERT
ITEROP	LPITN	MINOS	MKLIST	MODLU	PACKLU
PRICE	P3	P4	SETPI	SETX	TRNSVL
UNPACK					

Additional algorithmic routines for nonlinear problems

ADDCOL	ALIGN	CALCFG	CALCG	CG	CHKDIR
CHKGRD	CHUZQ	COMDFP	DELCOL	DOT	DOT1
FUNGRD	GETGRD	NEWPIC	RESETR	RGITN	RTFSOL
R1ADD	R1MOD	R1PROD	R1SUB	SEARCH	SQUEEZ

3. PROGRAMMING CONVENTIONS

The originating source code is for the IBM Systems 360 and 370. The notes here will facilitate conversion to other machines.

Types

In most subroutines the type of a variable is defined by the first character of its name, as follows:

A-B	REAL
C-G, O-Z	REAL*8
H	INTEGER*2
I-N	INTEGER

The only variables beginning with H are half-word arrays. These and other arrays are always typed and dimensioned explicitly, e.g.

```
INTEGER*2  HA(NE)
REAL       A(NE)
REAL*8     X(MN)
```

A few variables are explicitly typed LOGICAL.

IMPLICIT

Non-standard implicit typing is used only for double precision simple variables, and is accomplished where necessary by the statement

```
IMPLICIT   REAL*8(C-G,O-Z)
```

(which immediately follows a SUBROUTINE statement).

COMMON

Blank COMMON is not used. There are 24 labelled COMMON blocks, containing only simple variables and a few very short arrays. None of the COMMON variables is used for variable dimensions.

EQUIVALENCE

No EQUIVALENCE statements are used, except in a straightforward manner in subroutine MPS:

```
EQUIVALENCE  (KEY, ID(1), ID2(1), ID3(1), ID5(1))
```

DATA

Some subroutines use data statements to initialize integer variables to strings of at most four characters, e.g.

```
DATA   LRHS /3HRHS/
```

Such items immediately precede the first executable statement in a subroutine. For CDC compatibility, apostrophes are never used.

Variable dimensions

Subroutine parameters are used to define the size of all variable-length arrays. Arrays are strictly one-dimensional. Declarations such as REAL A(1) are never used.

Subscripts

Subscript expressions of the form X(M+J) are often used, but more complex forms such as X(HB(J)) are not.

Avoidance of trouble-spots

The following Fortran syntactical items are not used:

arithmetic IF
assigned GO TO
BACKSPACE
blank COMMON
BLOCK DATA
COMPLEX
ENDFILE
ENTRY
EQUIVALENCE (except as noted above)
EXTERNAL
FIND
FUNCTION
NAMELIST
nonstandard RETURN
PAUSE

PRINT
PUNCH
TIME
unformatted READ or WRITE
$ in identifiers
/ around subroutine parameters

3.1 Conversion to Other Machines

(a) INTEGER*2 may be replaced by INTEGERbb throughout (where b means "space"), although most compilers do the right thing anyway.

(b) If double precision is not required, the following modifications should be made:

change	to	in
REAL*8	REALbb	all
DABS	ABS	all
DMAX1	AMAX1	subroutine INITLZ
DMIN1	AMIN1	subroutine SEARCH
DSQRT	SQRT	all
D-	E-	subroutine INITLZ
DLOG	ALOG	subroutine CALCFG

The IMPLICIT statements may then be deleted.

(c) On the Burroughs B6700, a simpler alternative is to use the compiler control card $SET DBLTOSNGL.

(d) To speed editing it may help to note that the specifications
IMPLICIT
INTEGER*2
REAL*8
all begin in column 7.

(e) The quantities EPS, NWORDR, NWORDI, NWORDH must be set correctly in subroutine INITLZ.

(f) Floating-point constants are coded in single precision, e.g. PLINFY = 1.0E+30, and therefore do not require conversion. (An exception is the arguments to DMAX1 in subroutine INITLZ, requiring the change of D- to E- as noted above.)

(g) If double precision is required but the IMPLICIT statement is not allowed, all relevant simple variables beginning with C-G, O-Z must be typed explicitly, e.g. DOUBLE PRECISION EPS,EPSO,EPS1. This includes local variables, COMMON variables and subroutine parameters. Such a conversion will require many additional lines of otherwise unnecessary code. It will also be highly prone to error, since Fortran compilers normally do not issue warnings or error messages for undeclared variables.

(h) For CDC-RUN, several READ statements of the form

```
READ(u,f,END=m) l
```

must have the ",END=m" eliminated, and additional cards inserted of the form

```
IF (EOF,u) m,n
n CONTINUE
```

(i) In FORMAT statements, apostrophes are used to delimit strings. However, strings never contain the character *. Hence for CDC machines it is acceptable to convert every apostrophe to * (except perhaps in comment cards).

4. VARIABLES

4.1 Variable-dimension information

The following INTEGER variables are used to define the current length of various variable-dimension arrays. They usually occur at the front of a subroutine's parameter list.

LFN	Length of hash table used to store row names during input.
M	No. of rows in constraint matrix A.
MAXL	Maximum no. of elements currently allowed for in L file.
MAXS	Maximum no. of superbasic variables allowed.
MAXU	Maximum no. of elements currently allowed for in U file.
MAXZ	Size of Z, the single array of core in which all else resides.
MCOLS	No. of columns allowed for in A during input (must be >= N).
MELMS	No. of nonzeros allowed for in A during input (must be >= NE).
MN	M + MAXS. Maximum dimension of basics + superbasics.
MNN	max(MN, NN). Used only for dimension of work vectors Y and Y1.
MROWS	No. of rows allowed for in A during input (must be >= M).
MS	M + NS. Current dimension of basics + superbasics.
M1	M + 1.
M2	M + KINV. Up to this many transformations allowed for in L and U.
N	No. of variables in constraint matrix (structurals + RHS + slacks).
NE	No. of elements in constraint matrix.
NEN	No. of entries in hash table during input.
NF	Size of array F = MSPK*(MSPK+1)/2 where MSPK = max no. of spikes.
NN	No. of nonlinear variables (those involved in nonlinear objective).
NN0	max(NN, 1)
NP1	N+1.
NR	Size of array R = MAXS*(MAXS+1)/2.
NS	Current no. of superbasic variables.

4.2 COMMON declarations

The following labelled COMMON blocks are used to segment parameters into various logical sets. This is to avoid repetition of one large COMMON block in every subroutine.

```
      LOGICAL    CONV,RESTRT
      COMMON    /BGCOM / TOLSWP,NBUMP,NSPIKE,MSPK
      COMMON    /CGCOM / CGBETA,ITNCG,MSGCG,MODCG,RESTRT
      COMMON    /CONVCM/ BTASH,ETABG,CONV(4),LVLTOL,NXTPHS,NFAIL
      COMMON    /CORE  / KZ1,KZ2,KZ3
      COMMON    /DJCOM / TOLDJ1,TOLDJ2,TOLDJ3,TOLDJ
      COMMON    /EPSCOM/ EPS,EPS0,EPS1,EPS2,EPS3,EPS4,EPS5,PLINFY
      COMMON    /FILES / ISCR,INPUT,IOLDB,INEWB,INSRT,IPNCH,ILOAD,IDUMP
      COMMON    /FREQS / KCHK,KINV,KSAV,KLOG,I1FREQ,I2FREQ
      COMMON    /FXCOM / FX,SINF,WTOBJ,MINIMZ,NFX,NINF,IOBJ,NPROB
      COMMON    /INTCOM/ ITN,ITNLIM,NPHS,KMODLU,KMODPI
      COMMON    /INVCOM/ INVFQ,INVITN,INVMOD,NBELEM,KHL,KDL,KHU,KDU,KFF
      COMMON    /ITNLOG/ DJQ,THETA,PIVOT,COND,NONOPT,JP,JQ,KADD,KSUB
      COMMON    /ITNLG2/ IHEAD,JQ1,JQ2,JR1,JR2
      COMMON    /LPCOM / KRHS,NS1,MAXR,IERR,IDEBUG
      COMMON    /LUFILE/ IMBED,NETAL,NETAP,NETAU,KL,KU,LPGN,JLBGN
      COMMON    /MPSCOM/ AIJTOL,ESTRUC(2),MINMAX,MLST,MER,
     1                   NAME(2),MOBJ(2),MRHS(2),MRNG(2),MBND(2)
      COMMON    /NLCCOM/ NNLCN,LDERIV,LPRINT
      COMMON    /PARMCM/ DPARM(9),IPARM(9)
      COMMON    /PRCCOM/ NEARER,NPRC,JPRC,KPRC,MREJ,NREJ,JREJ(20)
      COMMON    /PRCCM2/ NMULPR,NEWSB
      COMMON    /RGTOLS/ XTOL(3),FTOL(3),GTOL(3),PINORM,PGNORM,TOLRG
      COMMON    /SCLNCM/ ISCIN,KSCLN,MSCIN,NSTATE,LCHKGR
      COMMON    /TOLS  / TOLX,TOLPIV,TRPIV1,TRPIV2,TOLROW,XNORM
      COMMON    /WORDSZ/ NWORDR,NWORDI,NWCRDH
```

4.3 Definition of COMMON variables

AIJTOL Tolerance for ignoring matrix coefficients during MPS input.

BSTRUC(2) Default lower and upper bound values for structural variables.

CGBETA Conjugate gradient scalar. (new D) = -gradient + CGBETA*(old D).

COND Square of condition number of diagonal of P. This provides a lower bound on the condition of the Hessian approximation.

CONV(4) Logical variables showing result of convergence tests in RGITN.

DJQ Best reduced cost found by PRICE.

DPARM(9) Array of floating-point parameters.

EPS Machine precision. 16**(-13) for IBM 370.

EPS0 EPS**(4/5). Used, among other things, for packing LU tranformations.

EPS1 EPS**(2/3). Used for finite differencing in ADDCOL.

EPS2 SQRT(EPS).

EPS3 EPS**(1/3).

EPS4 EPS**(1/4).

EPS5 EPS**(1/5).

ETARG Minimization-within-subspace accuracy tolerance, used in RGITN.

ETASR Linesearch accuracy tolerance, used in NEWPTC.

FTOL(3) Current, loose and tight tols on change in objective.

FX Current objective value (linear + nonlinear terms).

GTOL(3) Current, loose and tight tols on norm of reduced gradient.

IDEBUG Debug level.

IDUMP Unit no. for DUMP file.

IERR Error indicator for numerous routines.

IHEAD Determines if another heading is to be printed in itn log.

ILOAD Unit no. for LOAD file.

IMBED Determines if part of L should be imbedded in A.

INEWB	Unit no. for NEW BIT-MAP file.
INPUT	Unit no. for MPS file.
INSRT	Unit no. for INSERT file.
INVITN	No. of iterations performed since last invert.
INVMOD	No. of modifications made to LU factors since last invert.
INVRQ	Invert request. Set positive for numerous reasons.
IOBJ	Row no. of linear objective. Zero if none.
IOLDB	Unit no. for OLD BIT-MAP file.
IPARM(9)	Array of integer parameters.
IPNCH	Unit no. for PUNCH file.
ISCR	Unit no. for SCRATCH file.
ISOLN	Unit no. for SOLUTION file.
ITN	No. of iterations since start of run.
ITNCG	No. of consecutive conjugate-gradient itns since restart.
ITNLIM	Maximum no. of iterations allowed.
I1FREQ	May be set by I1FREQ card in SPECS file. 1 causes the bump and spike pattern to be displayed after each INVERT.
I2FREQ	May be set by I2FREQ card in SPECS file.
JLBGN	Beginning in HL,DL of transformation pointed to by LBGN.
JP	The JP-th column of the basis will be replaced.
JPRC	Column no. marking end of last partition of A scanned by PRICE.
JQ	Column no. of variable selected by PRICE or CHUZQ to enter basis.
JQ1	Column no. of variable entering the basis (for printing).
JQ2	Column no. of variable entering the superbasic set.
JREJ(20)	Column nos. of variables rejected from basis by INVERT.
JR1	Column no. of variable leaving the basis.
JR2	Column no. of variable leaving the superbasic set.
KADD	Indicator for modification of R.
KCHK	Check frequency (testing row residuals).

KDL	Beginning of array DI in memory array Z. (Part of LU file.)
KDU	Beginning of array DU in memory array Z. (Part of LU file.)
KFF	Beginning of array F in memory array Z. (Part of LU file.)
KHL	Beginning of array HI in memory array Z. (Part of LU file.)
KHU	Beginning of array HU in memory array Z. (Part of LU file.)
KINV	Factorization (invert) frequency.
KL	First free position in L file. KL-1 = no. of nonzeros in L.
KLOG	Log frequency (iteration log).
KMODLU	Set to 1 if LU factors are to be modified.
KMODPI	Set to 1 if PI is to be recomputed.
KPRC	Which partition of A was last scanned by PRICE.
KRHS	Column no. of the RHS (treated as a variable fixed at -1.0).
KSAV	Save frequency (for NEW BIT MAP).
KSOLN	Indicator for final call to CALCFG at optimal solution.
KSUB	Indicator for modification of R.
KU	First free position in U file. KU-1 = no. of nonzeros in U. This includes nonzeros in columns of U that have been deleted during updating.
KZ1	Points to beginning of core available for INVERT.
KZ2	Not used.
KZ3	Points to end of core used by arrays other than for LU.
LBGN	The first L transformation to be used by FTRANL and BTRANL.
LCHKGR	Indicator for verifying gradient of objective.
LDERIV	Derivative level. Not used.
LPRINT	Print level. Not used.
LVLTOL	2 (3) means loose (tight) tols for normal (final) iterations.
MAXR	Maximum dimension of R. Max no. of superbasics for quasi-Newton.
MBND(2)	Name of BOUND set.
MER	Maximum no. of error messages (of each type) during MPS input.
MINIMZ	1 for min, -1 for max.

MINMAX	Contains relevant string 'MIN' or 'MAX'.
MLST	Maximum lines of MPS data to be listed during input.
MOBJ(2)	Name of OBJECTIVE row.
MODCG	Defines which conjugate-gradient algorithm is to be used.
MREJ	Maximum no. of rejected vars. to be saved for BANDAID in PRICE.
MRHS(2)	Name of RHS.
MRNG(2)	Name of RANGE set.
MSGCG	Used for detecting first entry to conjugate-gradient routine (CG).
MSOLN	Indicator of request to print solution.
MSPK	Maximum no. of spikes currently allowed for.
NAME(2)	Name of problem (on first card of MPS file).
NBELEM	No. of nonzeros in basis on entry to INVERT.
NBUMP	No. of bumps found by P4 during refactorization.
NETAL	No. of transformations in L after invert.
NETAP	No. of transformations added to L during updating.
NETAU	No. of transformations in U, including ones flagged as deleted.
NEWSB	No. of nonbasic variables selected to become superbasic.
NFAIL	No. of consecutive linesearch failures.
NFX	No. of evaluations of objective function and its gradient.
NINF	No. of infeasibilities.
NMULPR	Multiple price indicator. Overrides NPARPR.
NNLFCN	No. of nonlinear functions (0 or 1 in MINOS. More in GPG).
NONOPT	No. of nonoptimal columns during last PRICEing sweep.
NPARPR	Partial price indicator = no. of partitions into which the constraint matrix A is divided for the purposes of pricing.
NPHS	Current phase (1, 2, 3 or 4).
NPRC	Size of partitions of A for partial pricing.
NPROB	Problem no., for use in CALCFG.
NREJ	No. of variables rejected from basis by INVERT.

NSPIKE	No. of spikes in F (active spikes in U), set by P4,FACTOR,MODIU.
NSTATE	Parameter of CALCFG. 0,1,2 indicate normal, first, last entries.
NS1	No. of nonlinear variables currently in the basis.
NWORDH	No. of HALF INTEGER words per word of memory array Z.
NWORDI	No. of INTEGER words per word of memory array Z.
NWORDR	No. of REAL words per word of memory array Z.
NXTPHS	In RGITN, suggests NPHS for next itn. FORMC may override.
PINORM	Norm of PI.
PIVOT	Pivot element for column entering basis.
PLINFY	1.0E+30. Used for "infinite" bounds.
RESTRT	Requests restart of conjugate gradient algorithm.
RGNORM	Norm of reduced gradient (largest amongst superbasics).
SINF	Sum of infeasibilities.
THETA	Step moved in current search direction.
TOLDJ	Current reduced cost tol (scaled by PINORM).
TOLDJ1	Reduced cost tol while infeasible.
TOLDJ2	Initial reduced cost tol during feasible itns if partial pricing.
TOLDJ3	Final reduced cost tol during feasible itns.
TOLPIV	Minimum size for PIVOT during iterations.
TOLRG	Level to which RGNORM must be reduced before adding superbasics.
TOLROW	Tol used in measuring "ROW ERROR" (size of RHS - A*X).
TOLSWP	LU MOD TOLERANCE. Relative pivot tol for updating LU factors.
TOLX	Feasibility tolerance for structurals and logicals.
TRPIV1	LU ROW TOLERANCE. Relative pivot tol used in FACTOR.
TRPIV2	LU COL TOLERANCE. Relative pivot tol for spike swaps.
WTOBJ	Weight on the true linear objective while infeasible (for the composite objective method).
XNORM	Norm of X for basic variables.
XTOL(3)	Current, loose and tight tols on change in X.

4.4 Arrays

All arrays are segments of the array Z. Most are allocated after input, once the dimensions of the problem are known. Those containing the LU factorization of the basis are reallocated at each call to INVERT.

Array	Length	
A	NE	Nonzero coefficients in the constraint matrix (A RHS I), listed column-wise.
BL	N	Lower bounds.
BU	N	Upper bounds.
C	NNO	Gradient of nonlinear part of objective function.
DL	MAXL	Nonzeros in L transformations.
DU	MAXU	Nonzeros in U transformations.
F	NF	A full triangular matrix, part of LU factors of basis.
G	MAXS	Reduced gradient vector for superbasics.
GNEW	MAXS	New reduced gradient vector for superbasics.
GPD	MN	Gradient vector for basics and superbasics.
G1	NNO	Temporary gradient vector for nonlinear objective.
HA	NE	Row indices for each column of A.
HB	MN	Column nos. of basics and superbasics. For basics (1 <= J <= M), HB(J) is the column that pivots on row J.
HE	NP1	HE(J) points to the end of column J-1 in arrays A,HA.
HL	MAXL	Row indices for nonzeros in each transformation in L. The first index for each transformation is negative if the pivot element is 1.0, otherwise the pivot is in DL.
HPIVF	M2	HPIVF(J) is the row on which the J-th spike pivots.
HPIVL	M2	HPIVL(J) is the pivot row for the J-th transformation in L.
HPIVU	M2	HPIVU(J) is the row on which the J-th column of U pivots. If negative, this column of J has been deleted.
HS	N	HS(J) is the state of the J-th variable in (A RHS I). 0 => variable J is nonbasic at its lower bound. 1 => variable J is nonbasic at its upper bound.

2 => variable J is superbasic.
3 => variable J is basic.
Nonbasic FREE variables may have state 0 or 1.

HSPIKE	M	During INVERT defines the spike structure of the basis. (This structure may be displayed by setting ITFREQ = 1). During iterations, HSPIKE is used as a work vector in calls to BTRANL.
HU	MAXU	Row indices for nonzeros in U.
NL	M2	NEL = NL(J) is the number of nonzeros in the J-th transformation of L. If NEL < 0, the transformation is actually column "-NEL" of A, with non-unit pivot element pointed to (indirectly) by an element of the array HL.
NU	M2	NU(J) is the no. of nonzeros in the J-th transformation of U.
PI	M	The pricing vector (i.e. the Lagrange multipliers or the simplex multipliers or the shadow prices for the general constraints).
R	NR	Upper triangular matrix used to approximate the reduced Hessian, thus: R'R = Z'GZ approximately, where Z spans the null space of the active constraints.
X	MN	Values of the basic and superbasic variables.
XN	NNO	Values of the nonlinear variables.
Y	MNN	Work vector.
Y1	MNN	Work vector.
Z	MAXZ	The single array of memory in which all else resides.

5. TOLERANCES

Most tolerances are defined in terms of the machine precision EPS = ε, which is the smallest positive number such that 1.0 + EPS is greater than 1.0, when the arithmetic is performed in the appropriate precision (e.g. single on CDC and Burroughs machines, double on IBM and Univac).

For convenience, the labelled common block EPSCOM contains the following quantities, which are initialized in subroutine INITLZ and never altered:

EPS = ε
EPS0 = $\varepsilon^{4/5}$
EPS1 = $\varepsilon^{2/3}$
EPS2 = $\varepsilon^{1/2}$
EPS3 = $\varepsilon^{1/3}$
EPS4 = $\varepsilon^{1/4}$
EPS5 = $\varepsilon^{1/5}$
PLINFY = 10^{30}

A tolerance TOLZ is used for packing transformations (PACKLU, MODLU) and for detecting negligible elements in transformed vectors in order to speed transformation processing (BTRANL, BTRANU, FTRANL, FTRANU). This tolerance is set to EPS0 at the beginning of the subroutines mentioned. A larger value for TOLZ would reduce the accuracy with which the constraints Ax = b could be satisfied, without ensuring any significant improvement in storage or run time. (In fact, the effect could be negative.)

The following tolerances are set in subroutine INITLZ (see the listing of that subroutine in section 10). The values used should be satisfactory for most well-scaled problems.

Tolerance	Value	Description
TOLX	$\max(\varepsilon^{\frac{1}{2}}, 10^{-5})$	Primal feasibility tolerance. Structural and slack variables are allowed to lie outside their bounds by as much as TOLX.
TOLDJ1	$\max(\varepsilon^{\frac{1}{2}}, 10^{-6})$	Relative dual infeasibility tolerance while infeasible. In PRICE, the actual tolerance

		used to screen reduced costs (reduced gradients) on nonbasic variables is TOLDJ1*PINORM.
TOLDJ2 TOLDJ3	1.0 $\max(\varepsilon^{\frac{1}{2}}, 10^{-6})$	Initial and final relative tolerances on reduced gradients in PRICE when feasible. If partial pricing is in effect, a relative tolerance TOLDJ begins at TOLDJ2 and is reduced in stages to the level TOLDJ3. Again, the actual tolerance used in PRICE is TOLDJ*PINORM. If partial pricing is not in effect, the value used in PRICE is TOLDJ3*PINORM.
TOLPIV	$\varepsilon^{\frac{1}{2}}$	The smallest pivot allowed in CHUZR during simplex iterations (LPITN). For non-simplex steps (RGITN) the value used is TOLPIV*YNORM, where YNORM is the norm of the search direction.
TOLROW	$\max(\varepsilon^{1/3}, 10^{-4})$	The tolerance used in SETX to determine if the constraints Ax = b are sufficiently well satisfied. The error allowed on each row is measured relative to the norm of the corresponding row in the current basis. See the description of SETX in section 10.

Certain other tolerances are available to the user to alter if desired, via the SPECS file. (This is accomplished in subroutine SPECS2.) Default values are shown below. Fuller details are given in the MINOS User's Guide.

Tolerance	Default Value	Description
AIJTOL	10^{-10}	The smallest matrix coefficient accepted as input.
ETARG	0.2	Affects the accuracy of minimization within each subspace. Refer to REDUCED GRADIENT TOLERANCE.
ETASH	0.01	Affects the accuracy of the linesearch. Refer to LINESEARCH TOLERANCE.

TOLSWP	0.99	Relative pivot tolerance for Bartels-Golub updating. Refer to LU MOD TOLERANCE in the User's Guide.
TRPIV1	0.001	Relative pivot tolerance for accepting the preassigned pivot row in FACTOR. Refer to LU ROW TOLERANCE.
TRPIV2	0.1	Relative pivot tolerance for accepting one of several possible spikes as substitutes for a column whose preassigned pivot row is unsatisfactory. Refer to LU COL TOLERANCE.
WTOBJ	0.0	The weight to be placed on the true objective function when infeasible. (The sum of infeasibilities always carries a weight of 1.0.) Refer to WEIGHT ON OBJECTIVE.

Post-script:

TOLX and TOLDJ3 above may also be changed via the SPECS file.
See section 12.

6. DATA STRUCTURES

Here we describe the arrays and variables used to store the constraints, the basis factors, and the reduced Hessian approximation.

6.1 Constraint Matrix [A b I]

REAL	A(NE)	Nonzero matrix and rhs coefficients.
INTEGER*2	HA(NE)	Corresponding row indices.
INTEGER*2	HE(NP1)	HE(J) points to the end of column J-1 in arrays A and HA.
REAL	BL(N)	Lower bounds on all variables (including rhs and slacks).
REAL	BU(N)	Upper bounds.

Related INTEGERs:

M	Number of rows in A (including objective rows).
N	Total number of variables (structurals, rhs and slacks).
NE	Total number of matrix elements (including rhs and slacks).
NP1	N+1.
KRHS	Column number for rhs.

The constraint matrix [A b I] is stored column-wise in packed form, using the arrays A, HA and HE. The coefficients for column J are in elements HE(J)+1 through HE(J+1) of arrays A and HA. The row indices in HA for a particular column are not required to be in ascending order (say); they are stored in the same order as they occur in the input data (which in general is arbitrary).

The absolute value of a particular element HA(I) is taken as the required row index. Hence, negative entries in HA may be introduced in special applications. (For example, in MINOS/GRG they are used to distinguish variable Jacobian elements from normal (constant) coefficients.) Note that HA(I)=0 is not allowed.

For a typical use of arrays A, HA and HE, see subroutine UNPACK in section 10.

Explicit upper and lower bounds are stored in arrays BU and BL for all variables. The value

$$\text{PLINFY} = 10^{30}$$

is used to represent "infinite" bounds. For instance, the bounds on slack variables to implement the four types of inequality constraint are as follows:

Constraint Type	BL	BU
L	0.0	PLINFY
E	0.0	0.0
G	-PLINFY	0.0
N	-PLINFY	PLINFY

RANGES on rows are implemented by altering either BL or BU in a trivial way.

Note: It is more common for the constraint matrix to be stored in the form [I A b]. This certainly simplifies the input procedures. We have chosen the less conventional form [A b I] to ensure that the column numbers of the first NN structural variables correspond to the nonlinear variables in the array X of the user's subroutine CALCFG. This also allows the simple test IF (J.LE.NN) ... to determine whether the J-th variable is nonlinear.

6.2 LU Factors of the Basis

Ignoring certain permutations, subroutine INVERT produces an LU factorization of the current basis B in the form

$$B = L\begin{bmatrix} I & U \\ & F \end{bmatrix} = L\begin{bmatrix} I & \\ & F \end{bmatrix}\begin{bmatrix} I & U \\ & I \end{bmatrix}$$

where L is lower triangular and F is upper triangular. The "bump and spike" structure of B is vital to the success of this factorization. B is first permuted to block lower triangular form using the procedures of Duff and Reid (1975, 1976). Each block ("bump") is then processed by the algorithm P^3 of Hellerman and Rarick (1971), thereby dividing the columns of B into two sets, called triangle columns and spike columns. Spikes are those columns that have nonzeros above the diagonal of the permuted B.

The LU factorization amounts to Gaussian elimination with column interchanges. If a column in a particular bump has to be interchanged with some alternative column, the only alternatives that need be considered are the remaining spikes in the same bump. If a triangle column is interchanged with a spike, the number of spikes generally increases by one. If two spikes are interchanged the number of spikes generally stays the same.

The dimension of F is the final number of spikes.

The reasons for computing the factorization in this way, and the (numerically stable) method for updating L, U and F when a column of B is replaced, are fully described in Saunders (1976). The result is a practical implementation of the method of Bartels and Golub (1969, 1971). Our aim here is to describe how the various matrices are stored.

For an initial factorization (immediately after INVERT), the columns of L and U are stored conventionally in packed form, working forwards through the arrays HL, DL and HU, DU. The upper triangular part of F is stored columnwise as a dense matrix in the array F.

When the factorization is updated, U and F are modified explicitly and continue to be stored in the same way. Updates to L are accumulated in product form. They are stored in arrays HL, DL, much like columns of the initial L, but interpretation of the contents of these arrays is unusually intricate.

Disclaimer:

Since the density of F is typically about 10% (though in some cases as high as 50%), the storage required for F becomes excessive if the number of spikes is 200 or more. (The efficiency of FTRANU and BTRANU is not necessarily degraded, because many rows and columns of F can be skipped during the transformation processing.)

In a future version of MINOS it is intended to store F row-wise via a linked-list data structure.

6.2.1 Storage for L

REAL*8	DL(MAXL)	Packed nonzeros.
INTEGER*2	HL(MAXL)	Corresponding indices.
INTEGER	NL(M2)	Number of nonzeros in each transformation.
INTEGER*2	HPIVL(M2)	Pivot rows.

Related INTEGERs:

MAXL	Maximum number of nonzeros currently allowed for in the L file. (Reset during each call to INVERT.)
M2	An upper bound on the number of transformations there could be before the next call to INVERT. (Fixed; M2=M+KINV, where KINV is the refactorization frequency.)
NETAL	The number of nontrivial columns in L after INVERT. (Those corresponding to slacks are not stored.)
NETAR	Current number of nontrivial updates to L.
KDL, KHL, KL	See also.

The K-th transformation in the L file is represented by NL(K), HPIVL(K) and a set of contiguous entries i,d in the arrays HL,DL. Let JL be a pointer such that JL+j marks the j-th pair (i,d) thus:

Index	HL	DL
JL	KEY	PIVOT
JL+1	i_1	d_1
JL+2	i_2	d_2
.	.	.
.	.	.
JL+NEL	i_{NEL}	d_{NEL}

Not all items need be present. Where relevant, let the following quantities be defined:

NEL = NL(K)
IPIV = HPIVL(K) = pivot row
KEY = HL(JL)
PIVOT = DL(JL) = pivot element

Initial L

For K $\leq$ NETAL, the K-th transformation represents the K-th column of L, which may be one of three types as follows.

Type 1: NEL < 0. Triangle column of B imbedded in the constraint matrix.

This column of L is the J-th column of A, where J = -NEL. Only one entry exists in HL and DL (KEY and PIVOT). KEY is not used.

Type 2: NEL > 0, KEY > 0. Triangle column of B, not imbedded in the constraint matrix.

This column of L is the column vector

$$(\text{PIVOT}, d_1, d_2, \ldots, d_{\text{NEL}})$$

with coefficients in the rows

$$(\text{IPIV}, i_1, i_2, \ldots, i_{\text{NEL}}).$$

It has been copied explicitly from A into the L file.

Type 3: NEL > 0, KEY < 0. Bottom half of a transformed spike.

This column of L is the vector in Type 2 divided by the pivot element, i.e.,

$$(1.0, d_1/\text{PIVOT}, \ldots, d_{\text{NEL}}/\text{PIVOT}).$$

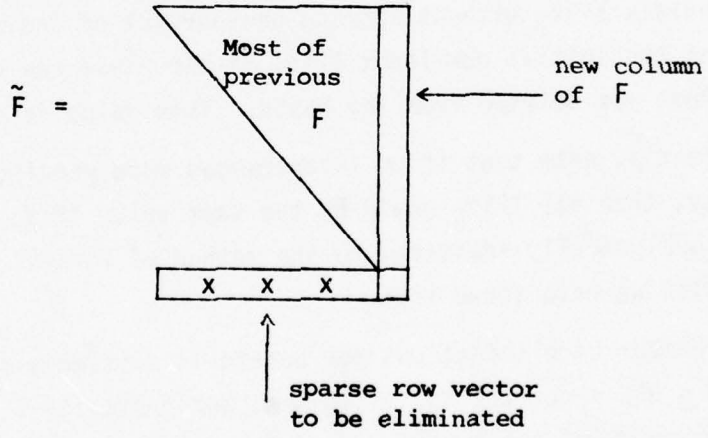
The divisions are not performed explicitly, as they are more economically accounted for whenever the transformation is used in FTRANL and BTRANL. Note that the pivot element for this spike transformation is deliberately chosen to be 1.0, and the element PIVOT is installed as the appropriate diagonal of F. (The two pivot elements could be PIVL and PIVF for any numbers such that PIVL*PIVF = PIVOT. However, the primary aim is to keep L well-conditioned, since no part of it is discarded, and to let the condition of the basis be reflected in F. Thus we choose L to be as near to the identity matrix as possible.)

Type 2 transformations would not be required if triangle columns were always imbedded in A. However, on a machine with virtual memory and/or a cache memory, transformation processing will be faster with Type 2 than with Type 1. Also for applications involving nonlinear constraints, the columns of A may contain variable elements of the Jacobian. Their present values can be retained with Type 2, but not with Type 1.

Updates to L

For NETAL < K $\leq$ NETAL + NETAR, the K-th transformation represents the (K-NETAL)th update to L. The items KEY and PIVOT do not exist.

Referring to section 4 of Saunders (1976), recall that any one such update comes from a product of elementary transformations E_j (j=1,2,...,NEL say), which reduce some intermediate matrix $\tilde{F}$ to upper triangular form $\bar{F}$, using Gaussian elimination with row interchanges. Thus



and

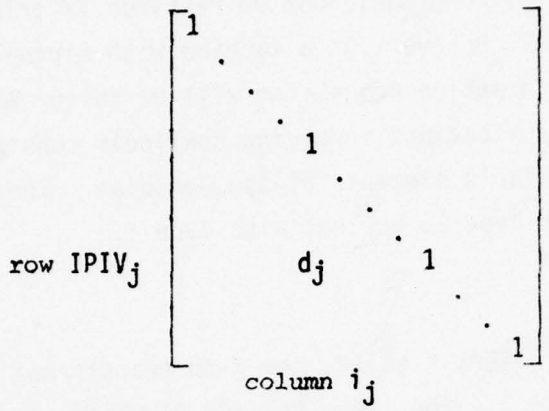
$$E_{NEL} \cdots E_2 E_1 \tilde{F} = \bar{F}.$$

If L is the previous (possibly updated) factor, the modified form of L is

$$\bar{L} = L M_1 M_2 \cdots M_{NEL}$$

where $M_j = E_j^{-1}$, but since M_j and E_j are trivially related, it is immaterial which of these matrices is stored. We choose to store E_j.

In the method of Bartels and Golub, each transformation E_j is a matrix of the form



which is possibly preceded by a permutation matrix that interchanges rows $IPIV_j$ and i_j. The indices i_j are the positions of nonzeros being eliminated from the bottom row of $\tilde{F}$. These indices and the corresponding multipliers are stored as the pair (i_j,d_j) in arrays HL,DL as shown earlier.

Now, the intricate part of the implementation lies in keeping track of the values $IPIV_j$ without storing another set of indices. First of all, note that the initial row index $IPIV_1$ is the pivot row corresponding to the column that was deleted from the basis. This value is stored as IPIV = HPIVL(K).

Secondly, note that if no interchanges were required to maintain stability, then all $IPIV_j$ would be the same value IPIV. (The update would then be analytically identical to the method of Forrest and Tomlin (1972).) In practice we have found that

(a) The number of eliminations per update is extremely small. It is bounded by $0 \leq NEL <$ *current no. of spikes*, but typically $0 \leq NEL \leq 20$ because of the sparsity of the bottom row of $\tilde{F}$.

(b) It is frequently the case that no interchanges are required, even if the tolerance TOLSWP is 1.0. (This means that the diagonal elements of F are usually larger than the elements being eliminated from $\tilde{F}$.) Perhaps this explains empirically the practical success of the method of Forrest and Tomlin on problems that are not severely ill-conditioned.

Because of point (b) we distinguish between the two updating cases.

Type 1. NEL > 0. No interchanges.
The row indices $IPIV_j$ are constant, namely IPIV. The indices i_j are stored in HL(JL+j). Transformation processing in FTRANL and

BTRANL can be coded somewhat more efficiently. In particular, a complete transformation can be skipped in BTRANL if the element Y(IPIV) in the vector being transformed is sufficiently close to zero.

Type 2. NEL < 0. Some interchanges occurred.
An interchange associated with E_j is recorded by storing HL(JL+j) negatively. During a forward pass through the E_j, the indices $IPIV_j$ and i_j are generated recursively as follows:

```
for j = 1 until abs(NEL) do
begin
        i = HL(JL+j);
        if i > 0 then ( IPIV_j = IPIV_j-1;  i_j = i )
                 else ( IPIV_j = abs(i);    i_j = IPIV_j-1 );
end.
```

For FTRANL the indices $IPIV_j$ and i_j are used as soon as they are generated. However, for BTRANL the indices $IPIV_j$ must be stored in a work vector during the above forward pass, and then used in reverse order.

Given from point (a) above that NEL is typically quite small, the double handling of the indices HL(JL+j) does not add up to a significant overhead, and in a pure-Fortran implementation the overall storage management is simplified by retaining just the two parallel arrays HL,DL. (In a machine-coded implementation the efficiency of FTRANL and BTRANL would be readily improved if the $IPIV_j$ for all updates were stored permanently, and the increase in storage would not be substantial.)

6.2.2 Storage for U

REAL*8	DU(MAXU)	Packed nonzeros for each column of U.
INTEGER*2	HU(MAXU)	Corresponding row indices.
INTEGER	NU(M2)	Number of nonzeros in each column of U.
INTEGER*2	HPIVL(M2)	Pivot rows.

Related INTEGERs:

MAXU	Maximum number of nonzeros currently allowed for in the U file. (Reset during each call to INVERT.)
M2	Previously defined.
NETAU	Current number of columns in U, including those that have been deleted.
KDU, KHU, KU	See also.

The K-th column of U is represented by NU(K), HPIVU(K) and a set of contiguous entries HU(JU+j), DU(JU+j) for j = 1,2,...,NU(K). Recall from the definition of U and F that each column of U represents a conventional eta transformation with pivot element 1.0 (which is therefore not stored).

When U is updated, new columns are added to the U file in the same format. If a spike was deleted during the basis change, the corresponding column of U is flagged by changing the sign of the entry in HPIVU. The storage for this column is not recovered. If a triangle column was deleted from the basis, any packed nonzeros in the relevant pivot row are physically reset to zero.

6.2.3 Storage for F

REAL*8	F(NF)	The elements of the upper triangular matrix F, stored column-wise.

Related INTEGERs:

MSPK	Maximum number of columns currently allowed for in F. (Reset during each call to INVERT.)
NF	Dimension of array F, namely MSPK*(MSPK+1)/2.
NSPIKE	The number of spikes found by P4. Later, the current number of columns in F (set in FACTOR and MODLU).
KFF	See also.

If NSPIKE = 3 and

$$F = \begin{bmatrix} 2 & 5 & 1 \\ & 8 & 0 \\ & & 9 \end{bmatrix}$$

then array F contains 3*(3+1)/2 = 6 elements, namely 2, 5, 8, 1, 0, 9 in that order.

If a spike is deleted during a basis change, the corresponding row and column of F are deleted and the storage is recovered.

6.3 Cholesky Factor of the Reduced Hessian

REAL*8 R(NR) The elements of the upper triangular matrix R, stored column-wise.

Related INTEGERs:

MAXS The maximum number of superbasic variables allowed.

MAXR The dimension of the reduced Hessian factor, R. $1 \leq MAXR \leq MAXS$. If the number of superbasic variables exceeds MAXR, the information in R is no longer updated. An exception is in DELCOL; if the JQ-th column is being deleted and $JQ < MAXR$, then R is updated normally, although R will not be used unless the number of superbasics decreases to MAXR. (It would probably be preferable to enforce R = I whenever the latter occurs, and skip any earlier updating in DELCOL.)

NR The dimension of array R, namely MAXR*(MAXR+1)/2.

Since R is in general a dense triangular matrix, it is stored in exactly the same manner as the matrix F on the previous page.

7. INVERT STATISTICS

The following items are printed in the Iteration Log whenever the basis matrix B is re-factorized. The factors L, U and F have been defined in section 6.2. The constraint matrix array is denoted by A.

Label	Description
FACTORIZE	Counts the number of factorizations since the start of the run.
DEMAND	Gives the reason for the present factorization, as described in section 8. (Variable INVRQ).
ITERATION	The current iteration number.
INFEAS	The number of infeasibilities at the start of the previous iteration.
OBJECTV	If INFEAS > 0, this is the sum of infeasibilities at the start of the previous iteration. If INFEAS = 0, this is the value of the correct objective function after the previous iteration.
SLACKS	The number of slack variables in the basis.
LINEAR	The number of basic structural variables that are not involved in the nonlinear objective.
NONLINEAR	The number of nonlinear variables currently in the basis.
ELEMS	The number of nonzero matrix coefficients in B.
DENSITY	Percentage nonzero density, 100*ELEMS/(M*M).
P4 BUMPS	The number of bumps or blocks found by P4 while permuting B to block lower triangular form.
SPIKES	The number of spikes tentatively assigned after applying P3 to each bump (prior to LU factorization).
CORE REQD	The number of words of core required for P4 and P3 to determine the bump/spike structure of B. This will always be substantially less than the total core required for the run (since the subsequent LU factorization requires significantly more core).

L LIMIT — The number of nonzeros allowed for in the LU factorization and subsequent updates to L.

U LIMIT — Similarly for U.

LU BUMPS — The number of bumps after LU factorization. This will invariably be the same as P4 BUMPS.

SPIKES — The number of spikes after LU factorization. Column interchanges may increase the number tentatively assigned by P3 above.

AIJ ELEMS — The number of nonzeros in L that are represented by pointers into A.
These belong to non-spike (i.e. triangle) columns of the basis, and will often comprise the bulk of L, unless there are very many spikes. This mode of storage can be suppressed by a card reading

IMBED NO

in the SPECS file. The L ELEMS count will then correspondingly increase.

L ELEMS — The number of nonzeros stored in the L file. This includes one for each column that is imbedded in A as above.

U ELEMS — The number of nonzeros stored as packed column vectors in the U file.

F ELEMS — The number of nonzeros in the (dense) triangular matrix F, followed by the percentage density of F. These figures are for interest only; they indicate the efficiency or otherwise of treating F as a dense matrix.

The following statistics refer to the LU factorization of B. They are printed only if one of the items TRISWAPS, SPKSWP or REJECTED is nonzero.

TRISWAPS — The number of times a triangle column was interchanged with a spike in the same bump to preserve numerical stability. The LU SPIKES count usually exceeds P4 SPIKES by this amount.

SPKSWP	The number of times two spikes in the same bump were interchanged to preserve numerical stability.
REJECTED	The number of variables rejected from the basis and replaced by suitable slacks. (This seldom occurs, unless the basis supplied at the start of a run proves to be singular, or else the current B is very ill-conditioned.) Any such rejected variables, up to a maximum of 20, are stored in a list and given priority for reintroduction to the basis during subsequent calls to PRICE.
MIN PIV RATIO	The smallest preassigned pivot ratio encountered and accepted. (This will always exceed LU ROW TOL.) For each bump column, the pivot ratio is measured as the size of the preassigned pivot element in the corresponding column of L, relative to the largest element in the same column of L and inside the relevant bump (including the pivot element itself).
TOLS	The values just used for LU ROW TOL and LU COL TOL respectively. (If numerical error is apparent following LU factorization, the factorization will be repeated with larger values for these tolerances.)

8. INVERT REQUESTS

The variable INVRQ is set in DRIVER and certain other subroutines to request a call to INVERT. If two such requests occur without an iteration being performed, an error exit is taken from DRIVER. If INVRQ > 20 an iteration was beginning but could not be completed.

INVRQ	Set in	Meaning
0	DRIVER	Start of run.
1	DRIVER	Invert frequency interrupt.
2	DRIVER	LU file has grown significantly.
3		
4	MODLU	No room to update L.
5	MODLU	No room to update U.
6	MODLU	No room to update F.
7	MODLU	New pivot element in F is too small.
8		
9		
10	DRIVER	Row error after row-check interrupt.
11		
12		
13		
14		
15		
21	DRIVER	No workspace in LU file for use by CHUZR.
22	RGITN	Linesearch failure.
23		
24		
25		

9. ERROR CONDITIONS

The variable IERR is set by many subroutines to indicate an error condition. Some values cause the run to terminate. If IERR < 10 it is safe to output the final basis and/or print the final solution.

IERR	Set in	Meaning	Suggested Remedy
-4			
-3			
-2			
-1	LPITN	With composite objective, the incoming variable will not improve the true objective.	
0	DRIVER	Proceeding, or solution is optimal.	
1	DRIVER	Problem is infeasible.	Check constraint data. Raise FEASIBILITY TOL.
2	LPITN, RGITN	Problem is unbounded.	Add realistic bounds to variables.
3	DRIVER	Too many iterations.	Continue run.
4	DRIVER	INVERT called twice in a row.	System error.
5	RGITN	Too many superbasics.	Increase SUPERBASICS.
6	SETX	Row error.	Scale data better.
7	RGITN	Unable to improve objective value after repeated attempts.	Is the nonlinear objective and gradient well-defined?
8	DRIVER, FUNGRD	User's CALCFG requests that the run be terminated.	
9	RGITN	CHUZQ failed too often to find a superbasic to replace a variable that wants to leave the basis.	Check scaling of the data.
10	INVERT	Not enough core for P4.	Increase core by a substantial amount.
11	FACTOR	Not enough core for L.	
12	FACTOR	Not enough core for U.	
13	FACTOR	Not enough core for F.	
14	INVERT	Wrong number of basic variables.	System error.
15			
16			
17			
18	DRIVER	Not enough core for INSERT.	Increase core.
19	DRIVER	Not enough core for LOADN.	Increase core.

IERR	Set in	Meaning	Suggested Remedy
20	LOADB	Incompatible basis map supplied.	Check OLD BASIS FILE.
21	DRIVER	Not enough core to output solution.	Increase core.
22			
23			
24			
25			

In general, error conditions and warnings are brought to the user's attention by a printed message starting with XXX .

Caution:

The message EXIT -- OPTIMAL SOLUTION FOUND must always be interpreted in terms of the size of the reduced-gradient vector, particularly if messages of the form

XXX NO FUNCTION DECREASE.

occur at the end of the run. The final computed solution may or may not be acceptably close to optimal, depending on the "friendliness" of the nonlinear objective and the condition of the current basis. The latter message occurs in RGITN if the linesearch procedure was unable to obtain an improved objective value, at a time when various tolerances suggest that it should have. The following information is also printed:

NORM RG = the largest reduced-gradient component amongst superbasics.
NORM PI = a measure of the size of the current π vector.
LAST SB = the column no. of the last superbasic variable. This was previously nonbasic with reduced-gradient shown.

The recovery action taken by MINOS is to add the most favorable nonbasic variable to the superbasic set and continue optimization in a larger subspace. If PRICE finds there are no favorable nonbasics, the solution is accepted as the best attainable and is declared OPTIMAL.

Broadly speaking, if (NORM RG)/(NORM PI) = 10^{-d}, then the objective function would probably change in the d-th significant figure if optimization could be continued. The user must decide for himself whether d is sufficiently large.

10. SUBROUTINE SPECIFICATIONS

Each of the subroutines in MINOS is documented under the following headings:

- Subroutines called and called by
- Purpose
- Parameters
- Method

The Parameters description does not include any variables with the same name and function as those defined in section 4.

```
      SUBROUTINE ADDCOL( M,M2,MNN,NE,NP1,NF,NP,NN,NS,MS,MAXL,MAXU,
     1   HSPIKE,HA,HE,HPIVL,HPIVU,HPIVF,HB,NL,NU,
     2   A,F,X,GRD,Y,Y1,XN,C,G1,GNEW,HL,DL,HU,DU,F,UNIT,JQ )
```

Subroutines called:

BTRANU CALCG FTRANI FUNGRD
GETGRD SETPI UNPACK

Called by:

RGITN

Purpose:

To construct a new column for R when the no. of superbasics is being increased by 1, from NS-1 to NS.

Parameters:

INTEGER	JQ	(Input) Column no. of variable being added to the set of superbasics.
LOGICAL	UNIT	(Input) Specifies if new column of R is to be a unit vector or not.

Method:

If UNIT = TRUE, the unit vector e(NS) will be used. This is the case if MULTIPLE PRICEing is invoked.

If UNIT = FALSE, an attempt will be made to estimate curvature in the new subspace by finite-differencing the gradient vector along the new column of Z (the null space matrix). See Murtagh and Saunders (1978), section 3.3.4.

If the off-diagonal elements of the new column show signs of being extremely big or small, further work is avoided and the unit vector e(NS) is used instead. If the new diagonal element proves to be significantly larger or smaller than the norm of R, again a unit vector is used and the work required to estimate the new column is wasted. The aim here is to avoid constructing a new R that is significantly more ill-conditioned than the previous R.

SUBROUTINE ALIGN(N,NS,HE,BL,BU,X,Y,THETAS,NALIGN,TSPECL)

Subroutines called:

DOT1

Called by:

RGITN

Purpose:

To scale the search direction for superbasics so that if several are about to reach a bound then all of them do so simultaneously.

Parameters:

INTEGER*2	HB(NS)	(Input)	Column nos. of superbasics (only).
INTEGER	NALIGN	(Output)	No. of variables finally aligned.
REAL*8	THETAS(NS)	Workspace to store steps to reach bounds.	
REAL*8	TSPECL	(Output)	Final step size, if NALIGN>0.
REAL*8	X(NS)	(Input)	Values of superbasics (only).
REAL*8	Y(NS)	(Input)	Search direction for superbasics.

Method:

1. Classify each superbasic variable X(j) as "normal" or "special" as follows. If the search direction Y(j) is larger than some tolerance ZTOLY, and if the distance to the relevant bound (depending on the sign of Y(j)) is less than some tolerance ZTOLY, then X(j) is special, otherwise normal.

2. Compute TNORML (or TSPECL), the step that makes the first normal (or special) variable hit its bound. Store steps for all special variables in array THETAS.

3. If the first superbasic to reach a bound is special (i.e., if TSPECL <= TNORML), scale down the components Y(j) for special variables so that all special variables will reach their bounds if a step TSPECL is taken along the modified search direction. (A basic variable may reach a bound earlier, in which case the alignment will have been to no avail.)

SUBROUTINE BTRANL(M,M2,NE,NP1,MAXL,HA,HE,R,HPIVL,NL,HL,DL,HW,Y)

Subroutines called:

None

Called by:

CHUZQ FACTOR PRICE

Purpose:

To solve the system I(transpose)*x = y, where both x and y are stored in the parameter Y. This is the 2nd part of a conventional "BTRAN" operation.

Parameters:

INTEGER*2 HW(M) — A work vector required to re-generate the sequence of row interchanges that occurred during each update to the LU factors. (Not required during INVERT).

Method:

The reverse of FTRANL. Updates are treated first, followed by the transformations constructed by FACTOR.

Treatment of the update transformations is rather complex because just one integer is stored per nonzero, rather than two, and a forward pass through the integers in each transformation is required to regenerate the second set.

During INVERT, only a partial sweep through the normal column transformations is needed. The variable LBGN allows the backward sweep to stop at the beginning of the current bump. It is reset to point to the beginning of I before exit from FACTOR.

SUBROUTINE BTRANU(M,M2,NF,MAXU,HPIVU,HPIVF,NU,HU,DU,F,Y)

Subroutines called:

None

Called by:

ADDCOL LPITN RGITN SETX

Purpose:

To solve the system U*x = y, where both x and y are stored in the parameter Y. This is the 2nd part of a conventional "FTRAN" operation.

Method:

Normal column-wise back-substitution, with each column of U and F being treated as a column of the full matrix "U". Columns are processed backwards. If a computed component of x is essentially zero, the corresponding columns of U and F can be skipped.

SUBROUTINE BUMPS(N,N1,NZ,IP,IRN,OUTP,NUMB,B,LOWL,PREV,NBLK)

Subroutines called: | Called by:

None | F4

Purpose:

Given the row numbers of the nonzeros in each column of a square sparse matrix A, this routine finds a permutation P such that P'AP is block upper triangular.

Parameters:

INTEGER	N	(Input)	Dimension of the matrix.
INTEGER	N1	(Input)	N+1.
INTEGER	NZ	(Input)	No. of nonzeros in the matrix.
INTEGER*2	IP(N1)	(Input)	Points to the first nonzero in each column.
INTEGER*2	IRN(NZ)	(Input)	List of row numbers for nonzeros.
INTEGER*2	OUTP(N)	Work vector.	
INTEGER*2	NUMB(N)	(Output)	NUMB(K) will contain the permuted position of column K.
INTEGER*2	B(N)	(Output)	B(I) will contain the row no. in the permuted matrix of the beginning of the I-th block.
INTEGER*2	LOWL(N)	Work vector.	
INTEGER*2	PREVL(N)	Work vector.	
INTEGER	NBLK	(Output)	No. of blocks found (including ones of dimension 1).

Method:

See R. Tarjan, SIAM J. Computing (1972), 1, pp. 146-160.

Subroutine BUMPS is derived directly from Harwell subroutine MC13A by J.K. Reid (1975). (MC13A has since been replaced in the Harwell Subroutine Library by MC13D; see I.S. Duff and J.K. Reid, "An implementation of Tarjan's algorithm for the block triangularization of a matrix, Report No. CSS 29, Harwell, April 1976).

Note: It is assumed that the matrix A has already been permuted to have nonzeros on its diagonal. See the transversal finder, subroutine TRNSVL.

SUBROUTINE CALCFG(MODE,N,X,F,G,NSTATE,NEROR)

Subroutines called:

None

Called by:

FUNGRD

Purpose:

To evaluate a user's particular nonlinear objective function f(x) and its gradient vector g(x) at a given point x.

Parameters:

The required form for this sutroutine is documented in the MINOS User's Guide, section III.3.

SUBROUTINE CALCG(M,NF,NP1,NS,MS,HA,HB,HE,A,PI,GRD,G,RGNORM)

Subroutines called:

None

Called by:

ADDCOL DRIVER RGITN

Purpose:

To calculate the reduced gradient vector G for the current set of superbasic variables.

Parameters:

REAL*8	GRD(MS)	(Input) Contains the current gradient vector for superbasics in positions M+1, ..., M+NS.
REAL*8	RGNORM	(Output) Returns the norm of the reduced gradient vector. This is taken to be the largest element of G in absolute size.

Method:

The J-th component of the reduced gradient is

$$G(J) = GRD(M+J) - PI*A(J)$$

where A(J) is the corresponding column of A.

SUBROUTINE CHKDIF(N,BL,BU,X,D,NFEAS)

Subroutines called:

None

Called by:

CHKGRD

Purpose:

CHKGRD is about to use direction D for forward differencing the nonlinear objective function, starting from the first feasible point X. This routine checks that D is a feasible direction, and modifies D if necessary.

Parameters:

INTEGER	N	(Input)	Will be NN, the no. of nonlinear vars.
REAL	BL(N)	(Input)	Lower bounds for the nonlinear vars.
REAL	BU(N)	(Input)	Upper bounds for the nonlinear vars.
REAL*8	X(N)	(Input)	Will be XN, the nonlinear variables.
REAL*8	D(N)	(In,out)	The direction to be checked.
INTEGER	NFEAS	(Output)	The no. of feasible components in the (possibly modified) vector D.

Method:

If variable X(J) is fixed (BL(J) = BU(J)), D(J) is set to zero and NFEAS is not incremented. Otherwise, D(J) is left unaltered or perhaps is reversed in sign, and NFEAS is incremented.

SUBROUTINE CHKGRD(N,BL,BU,X,G,IFAIL,A,B,C,EPSMCH)

Subroutines called: | Called by:

CALCFG CHKDIR DOT | DRIVER

Purpose:

To verify to some extent that the user has correctly programmed the gradient of the objective function in subroutine CALCFG.

Parameters:

INTEGER	N	(Input) Will be NN, the no. of nonlinear vars.
REAL	BL(N)	(Input) Lower bounds for the nonlinear vars.
REAL	BU(N)	(Input) Upper bounds for the nonlinear vars.
REAL*8	X(N)	(Input) Will be XN, the nonlinear variables.
REAL*8	G(N)	(Input) Will be C, the gradient at the first feasible point.
INTEGER	IFAIL	(Output) Will be zero if the gradients look OK, 2 if they don't, or 8 if the user sets MODE to a negative number in subroutine CALCFG, to terminate the run.
REAL*8	A(N)	Work vector for 1st difference direction.
REAL*8	B(N)	Work vector for 2nd difference direction.
REAL*8	C(N)	Work vector to hold gradients at X+H*A, X+H*B. Contents not used.
REAL*8	EPSMCH	(Input) Machine precision.

Method:

This routine was taken directly from subroutine CHKGRD by Gill, Murray, Picken and Barber. See Document no. F4/05/0/P/11/75, DNAC, National Physical Laboratory.

Slight changes were made, e.g. to make sure A and B were feasible directions.

SUBROUTINE CG(NS,G,GNEW,D,THETA)

Subroutines called:

DOT DOT1

Called by:

RGITN

Purpose:

To compute a search direction for superbasics, using one of several conjugate gradient methods for unconstrained minimization.

Parameters:

REAL*8	G(NS)	(Input) Reduced gradient at previous step.
REAL*8	GNEW(NS)	(Input) Reduced gradient at current step.
REAL*8	D(NS)	On input, the previous search direction for superbasics. On output, the new search direction for superbasics.
REAL*8	THETA	(Input) The step moved during the previous itn along the (old) search direction D.

Method:

1. Use MSGCG to print a message if this is the first entry to CG.

2. Use ITNCG to determine if NS consecutive iterations have been performed since the last restart. If so, request a restart.

3. Use MODCG to determine which version of CG has been requested.

4. Most methods (Fletcher-Reeves, Polak-Ribiere, Perry) compute the new D according to
   D(J) = -GNEW(J) + CGBETA*D(J)
   for some scalar CGBETA. Memoryless DFP, COMDFP, etc. add a term
   GAMMA*(GNEW(J) - G(J))
   for some scalar GAMMA.

Note: CG is called from RGITN at the end of an iteration, but only if that iteration was unconstrained. The required search direction D is then still available for use during the next call to RGITN. Note that in RGITN, D resides in Y(M+1), ..., Y(M+NS). This is overwritten if the ensuing step is constrained.

```
      SUBROUTINE CHUZQ( M,N,NE,NP1,MS,
     1    HA,HB,HE,A,BL,BU,X,Y,JQ,PIVOT,VARMET )
```

Subroutines called:

FTRANU BTRANL

Called by:

PGITN

Purpose.

When the JP-th variable leaves the basis, this routine selects one of the superbasic variables to replace it. The primary aim is to keep the basis well-conditioned. Where possible the choice is biased towards a superbasic that is away from its bounds, in the hope that it will be less likely to leave the basis during subsequent iterations.

Parameters:

REAL*8	Y(MS)	The first M components of Y contain the JP-th row of B inverse. On output, Y(M+1), ..., Y(M+NS) will contain a vector v required later by R1PROD.
INTEGER	JQ	(Output) Will point to the chosen superbasic. M < JQ <= M+NS.
REAL*8	PIVOT	(Output) The value of Y'*A(K) for the chosen column, K = HE(JQ).
LOGICAL	VARMET	(Input) Used to skip computing v if it is not going to be used later.

Method:

As described in Murtagh and Saunders (1978), section 3.3.2.

```
      SUBROUTINE CHUZP( M,N,HP,BL,BU,X,Y,ATBND,DJQ,TOLX,TOLPIV,THETA,JP,
     1                  RTHETA,TAU,MPT,MPI,IPMAX )
```

Subroutines called:

None

Called by:

IPITN RGITN

Purpose:

To choose an index JP and a step size THETA >= 0.0, such that variable HB(JP) reaches one of its bounds when X changes to X - THETA*Y. If X is feasible, HB(JP) will be the first variable to reach a bound. If X is infeasible, JP is chosen to minimize the sum of infeasibilities without regard to the number (which may increase).

Parameters:

INTEGER	M	(Input) This will really be M+1 when CHUZP is called from LPITN. (The variable entering the basis will be in position M+1, and Y(M+1) will be 1.0.) It will be M+NS when called from RGITN, and the last NS values of Y will be the search direction for superbasics.
REAL*8	Y(M)	(Input) The search direction.
LOGICAL	ATBND	(Input) TRUE for LPITN if variable entering basis is currently at its upper bound. In effect, ATBND causes -Y to be used in place of Y in the usual ratio test. In general, ATBND should therefore be TRUE if Y is being changed to X + THETA*Y.
REAL*8	DJQ	(Input) This is used only if the current X is infeasible. It is the slope of the piece-wise linear graph of the sum of infeasibilities versus THETA. More generally, it should be g'p, the usual innerproduct of the objective gradient with the search direction, p=Y. For LPITN, this is the reduced gradient of the incoming variable. For RGITN, it is g'p = g'(Zq) = (g'Z)*q = (g'Z)*(-Z'g) = - || Z'g || (squared) = - || G || (squared) where G contains the reduced gradient, Z'g; q is the part of p for superbasics, namely the negative reduced gradient; and Z is the null-space matrix.
REAL*8	TOLX	(Input) Feasibility tolerance.
REAL*8	TOLPIV	(Input) A component of Y will be treated as zero if smaller than this tolerance.
REAL*8	THETA	(Output) The chosen step size.
INTEGER	JP	(Output) The chosen index.

INTEGER*2	MPT(IPMAX)	(Workspace) Holds pointers to maintain a linked list in the parallel arrays MRI, RTHETA and TAU. The list is ordered according to increasing values of TAU, starting from the negative value, -abs(LJQ).
INTEGER*2	MRI(IPMAX)	(Workspace) Holds pivot rows corresponding to the steps in RTHETA.
REAL*8	RTHETA(IPMAX)	(Workspace) Holds list of step-sizes, THETA.
REAL*8	TAU(IPMAX)	(Workspace) Holds list of slopes. TAU(I) is the slope after a step RTHETA(I) is taken.
INTEGER	IPMAX	(Input) The maximum length allowed for the linked list. Storage for the 4 above arrays comes from unused space in the LU file and therefore varies between calls to CHUZP. It will always be at least 5 (otherwise INVERT will be called).

Method:

The following two methods, as implemented by John Tomlin, November 1975:

* If feasible, Paula Harris's two-pass method for taking advantage of near ties.

* If infeasible, Dennis Rarick's one-pass method for minimizing the sum of infeasibilities.

Harris

The aim here is to take advantage of the likelihood that several variables may reach a bound almost simultaneously when the final step THETA is taken. (This is especially probable on degenerate steps, i.e., when THETA = 0.0) The method does not expect to find exact ties but instead looks for near ties, which are considerably more frequent and mean much the same in practice, since the vertices of a simplex are in some sense "fuzzy", at least to the extent of the uncertainty in the original data.

1. (First pass) Compute a quantity PSI such that at least one variable would overshoot its bound by an amount PERTBN if a step PSI were taken. Since FORMC regards variables as being feasible only to within a tolerance TOLX, we won't mind if certain new variables go outside their bounds by a similar amount. PERTBN can therefore, with safety, be as large as TOLX. Thinking further along these lines, observe that the actual step THETA that is finally taken will be strictly less than PSI. Hence it is OK to allow PERTBN to be slightly larger than TOLX. We take the value 1.1*TOLX. More thinking required to decide what the maximum value should really be. (Actually, the real reason for taking PERTBN > TOLX is simply on principle, to avoid getting a zero numerator in the ratios defining PSI.)

2. (Second pass) Now go through computing the usual ratios, this time without perturbation. Skip any that are larger than PSI. Of the remainder, find the one for which the pivot element Y(J) is

largest.

Barick

A full description will not be attempted here. Briefly, the method would be conceptually simple if all the THETA's were computed and sorted into ascending order. Barick showed that with the aid of a linked list the sorting can be avoided and everything can be done during a single pass through X and Y.

For further details, see John Tomlin.

SUBROUTINE COMDFP(B,Y,G,NR,N,THETA,YTP,GTP,KADD,KSUB)

Subroutines called:

R1MOD

Called by:

BGITN

Purpose:

To implement the "Complementary DFP" or "BFGS" quasi-Newton update to a given Hessian approximation, stored in factorized form as R'R, thus:

(new R'R) = (old R'R) + c1 yy' + c2 gg'

where

g = new reduced gradient

y = g - (old g)

p = search direction

c1 = 1/(THETA*y'p)

c2 = 1/g'p.

Parameters:

REAL*8	THETA	(Input) The step just taken along direction p.
REAL*8	YTP	(Input) y'p.
REAL*8	GTP	(Input) g'p.
INTEGER	KADD	(Output) Error flag for adding the rank-1 matrix, c1 yy'.
INTEGER	KSUB	(Output) Error flag for subtracting the rank-1 matrix, c2 gg'.

Method:

Call R1MOD twice with appropriate parameters.

```
      SUBROUTINE CRASH( M,MN,N,NE,NP1,NN,NN0,NS,
     1   HA,HB,HE,HS,HPIVL,A,BI,BU,X,XN )
```

Subroutines called: | **Called by:**

None | DRIVER

Purpose:

To select M vectors from A in a way that will produce a lower triangular, nonsingular basis B. Three options are implemented, according to the value of IPARM(1).

IPARM(1)	Meaning
0	Set up the all slack basis.
1	Scan all columns of A.
2	Scan only those columns of A corresponding to linear variables.

Parameters:

```
INTEGER*2  HS(N)        (Output) Will define chosen basis.
INTEGER*2  HPIVL(M)     Work vector to record state of rows.
```

Method:

1. Initialize state vector HS(j) so variables are nonbasic at their smallest bound.

2. If NS>0, make relevant nonlinear variables superbasic, as specified by the INITIAL bounds set.

3. Look at the values given to nonlinear variables during input of the INITIAL bounds set. Set nonbasic nonlinears at the bound nearest to the values in XN, in case the LO or UP state specified during input is opposite to the default in 1 above.

4. Proceed to construct a triangular basis. The array HPIVL will have the following interpretation:

HPIVL(i)	Label	Meaning
K>0	Pivoted	Row i has been assigned as the pivot row for column K.
-1	Marked	Row i occurred in some previous column but has not yet been assigned as a pivot row (i.e. has not been labelled Pivoted).
0	Virgin	Row i has not yet occurred in any chosen column.

   Naturally, all rows start out labelled Virgin.

5. First, make any free logicals basic.

6. Now do two passes of the following, first on free structurals only, then on all structurals. (This gets free structurals into the basis first, subject of course to the basis staying triangular. It is quite possible that some free variables will still be nonbasic after CRASH.)

7. (a) For any given column, find

AIMAX	the biggest element in the whole column;
APIV(1)	the biggest element in Marked rows;
APIV(2)	the biggest element in Virgin rows;
NPIV	the no. of Pivoted rows in this column.

The rows corresponding to APIV(i) are the "best" Marked and Virgin rows respectively.

(b) Select a potential pivot row. If possible, take the best Virgin row. If none, and if NPIV=0, choose the best Marked row. Otherwise, skip the column altogether.

(c) TOLPR is some relative pivot tolerance, e.g. 0.01. To prevent getting a badly conditioned basis, skip the column if the chosen pivot element (APIV(1) or APIV(2)) is smaller than TOLPR*AIMAX.

(d) Otherwise, label the chosen row as Pivoted, set the state of the column to be basic, and label any remaining Virgin rows in the column as Marked.

8. Finally, fill up the basis with the logicals on any rows that did not get labelled as Pivoted.

SUBROUTINE DELCCI(M,NR,NS,MS,HB,R,X,JQ,DELETR)

Subroutines called:

None

Called by:

INVERT RGITN SQUEEZ

Purpose:

To perform housekeeping when the JQ-th superbasic variable is deleted. This may include deleting the JQ-th column of R and restoring R to upper triangular form.

Parameters:

INTEGER	JQ	(Input) The position of the superbasic being deleted.
LOGICAL	DELETR	(Input) TRUE if quasi-Newton or conjugate-gradient methods are in effect. FALSE if infeasible or problem is linear.

Method:

Perform a partial forward sweep of plane rotations, affecting the last rows and columns of R. Also, move columns JQ+1, ..., NS of R one place to the left. Adjust arrays HB and X similarly. Decrease NS by 1.

SUBROUTINE DOT(N,V,W,S)

Subroutines called:

None

Called by:

CG CHKGRD SEARCH

Purpose:

To compute the inner product S = V'*W for the two N-vectors V and W.

Parameters:

REAL*8	V(N)	(Input)
REAL*8	W(N)	(Input)
REAL*8	S	(Output)

SUBROUTINE DOT1(N,V,S)

Subroutines called:

None

Called by:

ALIGN CG RGITN

Purpose:

To compute the sum of squares S = V'*V for the N-vector V.

Parameters:

REAL*8	V(N)	(Input)
REAL*8	S	(Output)

```
      SUBROUTINE DRIVER( Z,MAXZ,IPSCLN,
     1   M,M2,MN,MNN,N,NE,NP1,NR,NN,NNO,MAXS,NS,
     2   HSPIKE,BE,HA,A,BI,BU,HB,XN,
     3   HS,HPIVL,HPIVU,HPIVP,
     4   NL,NU,A,C,X,BI,
     5   GRD,Y,Y1,G1,G,GNEW )
```

Subroutines called:

```
CALCG   CHKGRD CRASH  DUMEN
FORMC   FUNGRD GETGRD INSERT
INVERT  ITEROP LOADB  LOADN
LPITN   MODLU  PRICE  PUNCH
RESETR  RGITN  SAVEB  SETBI
SETX    SOLN   STATE  UNPACK
```

Called by:

MINOS

Purpose:

To invoke all necessary algorithmic routines for solving a problem, once it has been input. This includes setting up a starting basis, calling INVERT when required, switching between simplex and non-simplex iterations, checking that tolerances are tight enough to declare optimality, saving basis files, and outputting the solution. Much of the complexity of the code has to do with setting the Phase parameter NPHS, which can take the following values:

NPHS	Meaning
1	Ordinary Phase 1 simplex method (solution infeasible). Any superbasics will be ignored by PRICE.
2	Ordinary Phase 2 simplex method (solution feasible). Again, any superbasics will be ignored by PRICE.
3	Non-simplex iteration, (feasible or infeasible), including a PRICE operation to select additional superbasics.
4	Non-simplex iteration, (feasible or infeasible), operating on the current basics and superbasics, without a call to PRICE.

Using NXTPHS, subroutine RGITN suggests a value for NPHS for the next iteration, but FORMC may override, e.g. after INVERT.

All arrays required have already been allocated and are parameters to DRIVER, except for those allocated by INVERT for the LU file, namely HI, DL, HU, DU and P.

It is conceivable that DRIVER could be used as a subroutine within some different environment, but enormous care would be required to ensure that the various arrays and labelled COMMON areas were set up correctly, in the way that is presently accomplished by INITLZ, SPECS, SPECS2 and MPSIN.

Parameters:

REAL*8	Z(MAXZ)	(In,out) The array of core. INVERT must be free to use core from Z(KZ1) up (temporarily, while factorizing the basis), and from Z(KZ3) up thereafter (for the LU file), where KZ1, KZ3 are preset variables in COMMON /CORE /. Of course, all the array parameters of DRIVER also reside in Z in the current implementation of MINOS, but that is not necessary as far as DRIVER is concerned, and need not be true in some other environment. For example, KZ1 and KZ3 could both be 1. On output, the solution vector begins at Z(KZ1). This is a vector of dimension N, and contains values for structurals, RHS, and logicals, in that order.
INTEGER	IPSOLN	(Input) Should be positive if the solution is to be output using the standard output routine, SOLN. Should be zero otherwise (e.g. if the user wishes to do further computation on the solution).

INTEGERs	M,M2,MN,MNN,N,NE,NP1,NR,NN,NNO,MAXS,NS	are all input.
Arrays	HE,HA,A,BL,BU,HE,XN	are all input.
Arrays	HSPIKE,HS,HPIVL,HPIVU,HPIVP,NL,NU,R,C, X,PI,GRD,Y,Y1,G1,G,GNEW	provide workspace.
Parameters	M,M2,MN,MNN,N,NE,NP1,NR,NN,NNO,MAXS, HE,HA,A,BL,BU	will be unchanged.

Most other parameters may be used as output, depending on the application.

Method:

1. Initialize numerous scalar quantities. This includes setting the tolerances XTOL(1), FTOL(1), GTOL(1) to their "loose" values, as indicated by LVLTOL=2.

2. Select a starting basis, with the following order of preference: OLD BIT MAP, INSERT FILE, LOAD FILE, CRASH.

3. Call INVERT to factorize initial basis.

4. (Main loop)
   (a) Call FORMC to test feasibility and set NPHS.

   (b) Call CHKGRD if first feasible point has just been found.

   (c) Call RESETR if the Hessian is about to be used for the first time.

   (d) If NPHS>2 and feasible, call GETGRD to get an appropriate rhs for computing PI.

   (e) Call SETPI, unless PI has already been computed.

(f) If NPHS>2 and NS>0, call CALCG to compute the reduced gradient.

(g) If NPHS<4, call PRICE to select one or more additional superbasics.

(h) Skip to 5 if PRICE found nothing. Otherwise, reset XTOL(1), etc. to their "loose" values, in case they have been tightened up by a previous skip to 5. (Since PRICE found a variable to release from its bound, we were wrong in thinking earlier that the final set of active constraints had been determined.)

(i) If NPHS<=2, call LPITN.

(j) If NPHS>=3, call PGITN. On exit, if NXTPHS=0, PGITN will not have performed an iteration after all, but wants to try again with NPHS=3 (following a PRICE).

(k) Similarly, a iteration could not be performed if IERR=-1 (WTOBJ must be reduced) or if INVRQ>=20 (must call INVERT).

(l) Otherwise, an iteration has been successfully completed. Call ITEROP to output a line of Iteration Log.

(m) Test for various frequency conditions and/or excessive growth of the LU file. Save basis and/or call INVERT if necessary. Call MODLU if requested, to update basis factors. Check residuals every KCHK iterations.

5. (Apparently optimal)

PRICE found no favorable candidate. If infeasible, make sure that WTOBJ is zero; if feasible, check that tolerances XTOL(1), etc. are at their "tight" values (LVLTOL=3). If so, we're probably done; check residuals one last time before terminating. Otherwise, reset all relevant parameters and repeat from 4.

6. (Exit)

(a) Save basis map.

(b) Call FUNGRD one last time, if wanted.

(c) Set up solution vector in Z(KZ1), ..., using entry 1 of SOLN.

(d) Save PUNCH and/or DUMP files.

(e) Output solution to printer using entry 2 of SOLN (unless not wanted).

(f) Output SOLUTION FILE if required.

SUBROUTINE DUMEN(M,N,KRHS,HS,BL,BU,Y,ID1,ID2)

Subroutines called: | Called by:

None | DRIVEP

Purpose:

To output a basis description to file IDUMP. One card image containing KEY, NAME and VALUE is output for each variable, except for nonbasic variables that are fixed or at a zero bound. Logicals are treated the same as structurals. It is intended that a DUMP file be easy to modify when necessary (that is, somewhat easier than a PUNCH file).

Warning: Nonbasic variables that are fixed or at a zero bound will not be output. If the file is used as a LOAD file for a modified problem, it may be necessary to append LL, UL or SB cards specifying the state of such variables explicitly, if their bounds are relaxed. Otherwise, the initial solution obtained may be slightly different (and in some cases may be seriously infeasible). Similar problems arise with PUNCH files, but they can be overcome in both cases with a little care.

Parameters:

REAL*8	Y(N)	(Input) Contains solution values for all variables (structurals and logicals).
INTEGER	ID1(M)	Work vector to hold left part of row names.
INTEGER	ID2(M)	Work vector to hold right part of row names.

Method:

1. Read row names from the scratch file into ID1, ID2.

2. Process each variable in turn. Some monkeying around is necessary because structurals are output before logicals. For each variable J, if J<=KRHS then the relevant column name is to be read from the scratch file, otherwise the logical name is already in ID1, ID2.

3. Skip nonbasics that are at zero bounds or are fixed. Otherwise, use the state vector HS to index the local array KEY to get the correct indicator, then output it with the Name and Value.

4. Print message to indicate successful DUMP.

```
      SUBROUTINE FACTOR( M,M2,N,NE,NP1,NF,MAXL,MAXU,
     1    HSPIKE,HA,HB,HE,HS,HPIVI,HPIVU,HPIVF,NL,NU,
     2    A,BI,BU,HPIVR,HPIVI,X,Y,Y1,
     3    HL,DL,HU,DU,F,
     4    TPIVR,TPIVC,TMIN,NSWAP1,NSWAP2,NBELEM,KDONE )
```

Subroutines called: **Called by:**

BTRANL FTRANL PACKLU UNPACK INVERT

Purpose:

To compute an LU factorization of the basis matrix B, as specified by the list of column numbers in HPIVU. Arrays HSPIKE, HPIVR and HPIVI are assumed to be set up as described below.

Parameters:

INTEGER*2	HSPIKE(M)	(Input) Codes bump and spike structure as described in subroutine P4.
INTEGER*2	HPIVU(M2)	(Input) First M components are column nos. in the preassigned order from P4.
INTEGER*2	HPIVR(M)	(Input) The corresponding preassigned pivot rows.
INTEGER*2	HPIVI(M)	(Input) The inverse of HPIVR, used to test if a given row lies within a given bump.
REAL*8	X(M)	Work vector for holding a row of L inverse.
REAL*8	Y(M)	Work vector for holding transformed columns.
REAL*8	Y1(M)	Work vector for holding pivot elements of alternative spikes.
REAL*8	TPIVR	(Input) Row tolerance. Before column interchanges are invoked, a preassigned pivot element is tested to make sure it is within TPIVR of the biggest remaining unpivoted element in the current bump.
REAL*8	TPIVC	(Input) Column tolerance. If several spikes are available for column swapping, the one finally chosen must have a pivot element within TPIVC of the largest.
REAL*8	TMIN	(Output) The smallest row pivot ratio that was encountered during the factorization. If the factorization has to be done again for numerical reasons, TPIVR must be made bigger than TMIN or no improvement will occur.
INTEGER	NSWAP1	(Output) The no. of times a triangle column was swapped with a spike.
INTEGER	NSWAP2	(Output) The no. of times a spike was swapped with some other spike.
INTEGER	NBELEM	(Output) The no. of elements of L that have been imbedded directly in A (those in non-spike columns if IMBED = TRUE, otherwise 0).
INTEGER	KDONE	(Output) A counter to save work if the LU factorization has to be redone for storage (rather than numerical) reasons. The existing row and column ordering will be retained up to the point KDONE, which was where the previous

factorization got interrupted.

Method:

Gaussian elimination, with column interchanges where necessary to preserve numerical stability. Interchanges can safely be restricted to each bump separately. If necessary, some columns may be rejected from B and replaced by suitable unit vectors.

For K = 1, ..., M, the K-th column is treated according to the following steps.

1. Determine if K lies within the current bump boundaries, thus: IBGN <= K <= IEND. If not, use the bump and spike information in HSPIKE to reset IBGN and IEND.

2. The K-th basis column corresponds to variable J = HPIVU(K), and is supposed to pivot on row IPIV = HPIVR(K). Skip the column immediately if J is a slack pivoting on its own row. Otherwise, HSPIKE(K) is negative if the column is a spike.

3. (Non-spike) The column will become part of L directly, unless its pivot element is too small. A certain "row test" must now be applied. Use HPIVI to find TMAX, the largest of those elements in the column that lie inside the current bump, in rows that have not yet been pivoted on. The column fails the row test if its pivot element is smaller than TOLPIV (an absolute test), or TPIVR*TMAX (a relative test); in such cases, skip to 5. Otherwise, copy column J into the L-file (if IMBEDL = FALSE), or set up a pointer to the column within the constraint matrix A. Then repeat from 1.

   Note: the relative row test is not applied if we previously had trouble with the K-th column. The present column has already been determined to be the best substitute available, and GIVEUP has been set to TRUE to signify this fact.

4. (Spike) Unpack column J and transform it by that part of the L file belonging to the current bump (with the help of variables LBGN and JBGN that are available to FTRANL), thus obtaining a vector Y. Check the pivot element as before, this time looking at elements in Y that are inside the current bump. If the pivot fails the row test, skip to 5. Otherwise, call PACKLU to put the bottom half of Y into L and the top half into U. Repeat from 1.

5. (Row test failed. Look for a substitute spike)

   (a) If GIVEUP = TRUE, skip to 7. We have already tried to find a substitute for this position.

   (b) Scan the remaining spikes in the current bump. If none, skip to 7. If only one, grab it immediately and skip to 6.

   (c) Otherwise, find the IPIV-th row of (L transpose)-inverse, and use it to compute the pivot elements for all eligible spikes. This requires simple innerproducts of the dense vector X with the sparse spike columns. At the same time, find PMAX, the largest potential pivot element on row IPIV.

   (d) Now go through the spikes from left to right, skipping those

whose pivot element is not within TPIVC of PMAX. This is a "column test" that makes the process equivalent to Gaussian elimination with column interchanges. If the current column is not a spike, take the first spike that satisfies the column test, because it is likely to be the shortest, and skip to 6. Otherwise, for each remaining spike that satisfies the column test, extract its preassigned pivot row (LPIV, say), and find the one that maximizes Y(LPIV). (This heuristic for selecting an alternative spike is due to Charles Krabek of CDC. It is an attempt to prevent swapping the bad spike into a position that will again require a swap.)

6. Interchange appropriate elements of HPIVU and HSPIKE, and set GIVEUP = TRUE. Repeat from 2.

7. (Reject column) Replace column J by the corresponding slack (column no. KRHS+IPIV). Accumulate rejected columns in array JREJ, for preferential treatment by PRICE later on (BANDAID). Repeat from 2.

SUBROUTINE FORMC(M,N,NN,NNO,NS,MS,HB,HS,BL,BU,X,GRD,XN,C,TOLX)

Subroutines called: **Called by:**

FUNGRD DRIVER

Purpose:

To determine if the current X is feasible, and to set up a gradient vector GRD to be used for computing PI.

Also, to reset the phase indicator NPHS when needed (for the first iteration, and following INVERT).

Parameters:

REAL*8	GRD(MS)	(Output) The gradient vector for the current linear objective (whether feasible or not). GRD(j) = 0.0 if X(j) is feasible; = 1.0 if X(j) is above its upper bnd; = -1.0 if X(j) is below its lower bnd. GRD(ICBJ) is treated specially as noted below.
REAL*8	XN(NNO)	(In,out) If X is feasible but the previous iteration was not (or if INVERT has just been called), any nonbasic nonlinear variables will be given the appropriate bound values in XN. These will be retained during subsequent iterations.
REAL*8	C(NNO)	(Output) Again, on the first feasible iteration or after INVERT, the gradient vector for the nonlinear variables will be computed in C, by one call to FUNGRD.
REAL*8	TOLX	(Input) The feasibility tolerance for all variables.

Method:

1. Run through the X(j) values, computing the number and sum of infeasibilities (NINF and SINF) and setting GRD(j) as above.

2. (Infeasible)
   (a) If this is the beginning of the run (ITN=0), set NPHS=1 as first preference, to get regular Phase 1 simplex method. Even if there are some superbasics, NPHS remains 1 unless MULTIPLE PRICE is in effect, or a basis file was used to start the run. (The aim here is to leave superbasics fixed at the values specified by an INITIAL bounds set and perform normal Phase 1 - Phase 2 simplex iterations on the remaining variables as long as possible.)

   (b) At this stage, GRD(IOBJ) will be zero unless TARGETing is in effect (in which case X(ICBJ) may be infeasible). Resetting GRD(IOBJ) to GRD(IOBJ) - MINIMZ*WTOBJ has the required effect for targeting and/or the composite objective method, and also gives zero as required if neither is in effect.

3. (Feasible)
(a) If this is the beginning of the run, set NPHS=2 for cold starts where possible, for the reasons given in 2(a) above.

(b) In all cases, reset GRD(IOBJ) = -MINIMZ. This will be the only nonzero component of GRD, and gives the correct gradient vector for the linear objective. GETGRD overwrites any nonlinear components later on.

(c) If this is the first feasible iteration, or the first iteration after INVERT, set up nonbasic nonlinear values in XN and compute the nonlinear objective and gradient by a call to FUNGRD.

SUBROUTINE FTRANI(M,M2,NE,NP1,MAXL,HA,HE,A,HPIVL,NL,HL,DL,Y)

Subroutines called:

None

Called by:

ADDCOL FACTOR LPITN PGITN
SETX

Purpose:

To solve the system L*x = y, where both x and y are stored in the parameter Y. This is the 1st part of a conventional "FTRAN" operation.

Parameters:

All previously defined.

Method:

The column transformations constructed by FACTOR (during INVERT) are processed first, beginning at the one marked by LBGN and JLBGN. During INVERT, these parameters point to the start of the current bump, since any earlier transformations would have no effect on the vector being transformed. During subsequent iterations, LBGN and JLBGN point to the beginning of the L file.

If NETAB > 0, some additional transformations of a different type have been added to the L file during iterations. These are processed next. Each transformation is really a sequence of more elementary transformations of the type

"Add DL(J) times row I to the pivot row IPIV"

where I is either HI(J) (if that quantity is positive) or the previous value of IPIV. Thus, a negative value of HL(J) signals that the transformation is to be preceded by an interchange, as required for stability by the method of Bartels and Golub.

SUBROUTINE FTRANU(M,M2,NF,MAXU,HEIVU,HPIVF,NU,HU,DU,F,Y)

Subrcutines called:

None

Called by:

CHUZQ PRICE

Purpcse:

To sclve the system U(transpose)*x = y, where both x and y are stored in the parameter Y. This is the 1st part of a conventional "BTRAN" operation.

Method:

Forward-substitution. In this case U and F are treated separately. Columns of U (rows of U(transpose)) are processed from front to back. No transformations can be skipped. However, since F is a dense matrix it is possible to traverse either by rows or by columns as convenient. Here the forward-substitution runs across the rows of F and thereby allows a row to be skipped if the corresponding component of the computed solution is negligible.

SUBROUTINE FUNGRD(NN,MS,HE,XN,C,X,FN)

Subroutines called:

CALCFG

Called by:

ADDCOL DRIVER FORMC RGITN SEARCH

Purpose:

To call the user-written subroutine CALCFG with appropriate parameters, and return the current value of the objective function (both linear and nonlinear terms). Will be called only if the current point is feasible.

Parameters:

REAL*8	XN(NN)	(Input)	Current value of nonlinear variables.
REAL*8	X(MS)	(Input)	Current value of basics and superbasics.
REAL*8	C(NN)	(Output)	Computed gradient of nonlinear obj.
REAL*8	FN	(Output)	Returns the objective value.

Method:

1. Extract basic and superbasic values from X and store in XN in natural order. It is assumed that nonbasic nonlinear variables are already stored in XN; this is arranged by FORMC after each call to INVERT.

2. Test NFX for first entry to FUNGRD; set parameter NSTATE accordingly and call CALCFG to obtain nonlinear objective value FN and gradient vector C.

3. Test mode to see if user's subroutine CALCFG requests termination of the run.

4. If a linear objective row exists, change FN to FN - X(IOBJ) to include the linear part of the objective.

SUBROUTINE GETGRD(NN,NNO,MS,HB,C,GRD)

Subroutines called:

None

Called by:

ADDCOL DRIVER RGITW SEARCH

Purpose:

To set up the gradient vector for basic and superbasic variables in the order defined by HB.

Parameters:

All previously defined.

Method:

GRD(j) is zero for linear variables, +C(k) or -C(k) for nonlinears, where k=HB(j). GRD(ICBJ) = -MINIMZ.

SUBROUTINE GO(Z,NWCORE)

Subroutines called:

MINOS

Called by:

MAIN PROGRAM

Purpose:

To call MINOS repeatedly until parameter IERROR is negative, which signals that no further SPECS exist on file ISPECS and hence all problems have been processed.

This routine is the one that may be altered if MINOS is to be used as a subroutine for any special purpose. For example it may include calls to a matrix generator and a report writer before and after the call to MINOS.

Parameters:

REAL*8	Z(NWCORE)	(Input)	The available array of core.
INTEGER	NWCORE	(Input)	The no. of words of core in Z.

Method:

The standard version of GO is shown on the following page.

```
      SUBROUTINE GO( Z,NWCORE )
      IMPLICIT    REAL*8(C-G,O-Z)
      REAL*8      Z(NWCORE)
      COMMON     /FILES / ISCR,INPUT,IOLDB,INEWB,INSRT,IPNCH,ILOAD,IDUMP
      COMMON     /SOLNCM/ ISOLN,KSOLN,MSOLN,NSTATE,LCHKGR
C     ------------------------------------------------------------------
C     STANDARD CALLING ROUTINE FOR  M I N O S
C     ------------------------------------------------------------------
C     SOLVES PROBLEMS ONE AT A TIME, READING SPECS FROM CARD READER.
C     STANDARD OUTPUT ROUTINE REQUESTED.  NROWS,NCOLS, ETC. IGNORED.
C
C
C
  100 ISPECS = 5
      ISCRCH = 8
      IPSOLN = 1
C
C
C
C     CALL MATGEN( Z,NWCORE,ISPECS,INPUT,IERROR )
C     IF (IERROR.NE.0) RETURN
C
C
C
      CALL MINOS( Z,NWCORE,ISPECS,ISCRCH,IPSOLN,
     1            IERROR,NROWS,NCOLS,LXS,LXL,LPI,LHS,LFREE,NFREE )
C
C
C
C     IF (IERROR.NE.0) GO TO 900
C     CALL REPORT( ISCRCH,ISOLN,NROWS,NCOLS,
C    1             Z(LXS),Z(LPI),Z(LFREE),NFREE )
C
C
C
  900 IF (IERROR.GE.0) GO TO 100
      RETURN
C     END OF GO
      END
```

```
      SUBROUTINE HASH( LEN,NEN,NCOLL,
     1    KEY1,KEY2,MODE,KEYTAB,NAME1,NAME2,KA,FOUND )
```

Subroutines called: | **Called by:**

None | MPS

Purpose:

To look up and/or insert entries in a table.

Parameters:

INTEGER	LEN	(Input) The length of the hash table.
INTEGER	NEN	(Input) Maximum number of entries allowed. (LEN >= NEN. Usually, LEN is about 2*NEN.)
INTEGER	NCOLL	Accumulates total no. of collisions.
INTEGER	KEY1	(Input) Left part of current entry.
INTEGER	KEY2	(Input) Right part of current entry.
INTEGER	MODE	(Input) 1 for look-up only; 2 for look-up and entry into table if KEY1 and KEY2 are not already in table.
INTEGER	KEYTAB(LEN)	A table of keys pointing into the list of distinct entries.
INTEGER	NAME1(NEN)	Left part of list of distinct entries.
INTEGER	NAME2(NEN)	Right part of list of distinct entries.
INTEGER	KA	(Output) If MODE=1 (look-up), KA points to the position in the NAME1-NAME2 list where KEY1-KEY2 was found; will be zero if not found. If MODE=2, KA points to position where new entry was made (except KA=0 if table is full) or to position where existing entry was found.
LOGICAL	FOUND	(Output) TRUE if entry KEY1-KEY2 already existed.

Method:

See B.P. Brent, "Reducing the retrieval time of scatter storage techniques," Comm. ACM 16 (1973), pp. 105-109. This version has been simplified for the case where no entries will ever be deleted.

Brent's method is well suited to this particular application because each entry (ROW NAMES in the MPS constraint data) is quite likely to be looked up several times.

Warning: the hash function to be used in two places in this routine is machine-dependent. The required properties for it are documented in the MINOS User's Guide, section IX.2.

SUBROUTINE INITLZ

Subroutines called:

None

Called by:

MINOS

Purpose:

To initialize the machine precision EPS and the word-length indicators NWORDR, NWORDI, NWORDH, and to compute numerous tolerances, in terms of EPS where applicable.

Method:

The IBM 370 version of INITLZ is shown on the following page.

```
      SUBROUTINE INITIZ
      IMPLICIT   REAL*8(C-G,O-Z)
      COMMON     /DJCOM / TOLDJ1,TOLDJ2,TOLDJ3,TOLDJ
      COMMON     /EPSCOM/ EPS,EPS0,EPS1,EPS2,EPS3,EPS4,EPS5,PLINFY
      COMMON     /RGTOLS/ XTOL(3),FTOL(3),GTOL(3),PINORM,RGNORM,TOLRG
      COMMON     /TOLS  / TOLX,TOLPIV,TRPIV1,TRPIV2,TOLROW,XNORM
      COMMON     /WORDS2/ NWORDR,NWORDI,NWORDH
C
C     THE FOLLOWING 4 NUMBERS ARE THE ONLY MACHINE-DEPENDENT PARAMETERS
C
C     EPS    = THE MACHINE'S FLOATING-POINT PRECISION
C     NWORDR = NO. OF "REALS"         PER WORD (VARS STARTING WITH A-B)
C     NWORDI = NO. OF "INTEGERS"      PER WORD (VARS STARTING WITH I-N)
C     NWORDH = NO. OF "HALF INTEGERS" PER WORD (VARS STARTING WITH  H )
C        WHERE "WORD" MEANS SPACE USED BY VARS STARTING WITH C-G,O-Z.
C
C
C     IBM 360 AND 370
C
      EPS    = 16.0**(-13)
      NWORDR = 2
      NWORDI = 2
      NWORDH = 4
C
C     USE EPS TO SET OTHER MACHINE PRECISION CONSTANTS
C
      EPS0 = EPS**(4.0/5)
      EPS1 = EPS**(2.0/3)
      EPS2 = EPS**(1.0/2)
      EPS3 = EPS**(1.0/3)
      EPS4 = EPS**(1.0/4)
      EPS5 = EPS**(1.0/5)
      PLINFY = 1.0E+30
C
C     SET TOLERANCES
C
      TOLX   = DMAX1(EPS2, 1.0D-5)
      TOLPIV = EPS2
      TRPIV1 = 0.001
      TRPIV2 = 0.1
      TOLROW = DMAX1(EPS3, 1.0D-4)
      TOLDJ1 = DMAX1(EPS2, 1.0D-6)
      TOLDJ2 = 1.0
      TOLDJ3 = DMAX1(EPS2, 1.0D-6)
      XTOL(2)= 0.1
      XTOL(3)= TOLX
      FTOL(2)= XTOL(2)*0.1
      FTOL(3)= XTOL(3)**2
      GTOL(2)= DMAX1(EPS4, 1.0D-3)
      GTOL(3)= DMAX1(EPS2, 1.0D-7)
C
      RETURN
C     END OF INITIZ
      END
```

SUBROUTINE INSERT(M,N,NN,NNO,MN,MAXS,NS,HS,HB,BL,BU,X,XN,ID1,ID2)

Subroutines called: NMSRCH

Called by: DRIVER

Purpose:

To set up a starting basis using data on file INSRT. This routine is intended to provide compatibility with commercial systems by accepting data in the format produced by the conventional PUNCH command. (It also reads a file produced by subroutine PUNCH in MINOS.) This data is of the form

KEY NAME1 NAME2 VALUE

where KEY may be one of the standard set LL, UL, XL, XU or else SB to indicate superbasic. VALUEs are used only if KEY = SB.

Parameters:

INTEGER	NS	(Output) The final no. of superbasics.
INTEGER*2	HB(MN)	(Output) Will contain a list of basic and superbasic column nos.
REAL*8	X(MN)	(Output) X(j) will contain values for j=M+1, ..., M+NS.
INTEGER	ID1(N)	Workspace to hold left part of variable names.
INTEGER	ID2(N)	Workspace to hold right part of variable names.

Method:

1. Read row and column names from scratch file into ID1, ID2.
2. Set structurals to be nonbasic at their smallest bound.
3. Set state of logicals to be basic.
4. Process data cards one at a time. Exchange cards are ignored if the incoming variable (NAME1) is already basic or superbasic, or if the outgoing variable (NAME2) is not basic. This guarantees that there will always be M basic variables on exit.

```
      SUBROUTINE INVERT( MAXZ,M,M2,MN,N,NE,NP1,NP,NB,NN,NS,MAXL,MAXU,
     1   HSPIKE,HA,HE,BE,HS,HPIVL,HPIVU,HPIVP,NL,NU,
     2   A,BL,BU,R,X,HPIVB,HPIVI,Y,Y1,Z )
```

Subroutines called:

DELCOL FACTOR P4 RESETB
SETX

Called by:

DRIVER

Purpose:

To compute an LU factorization of the current basis, B.

Parameters:

INTEGER*2	HPIVR(M)	Workspace to hold row permutation.
INTEGER*2	HPIVI(M)	Workspace to hold inverse of HPIVR.
REAL*8	X(MN)	The first M locations in X are available for workspace during FACTOR, but X(M+1), ..., X(M+NS) contain values for superbasics and must not be overwritten. On exit, values for basic variables will be in X(1), ..., X(M).
REAL*8	Y(M)	Workspace for FACTOR, SETX.
REAL*8	Y1(M)	Workspace for FACTOR, SETX.
REAL*8	Z(MAXZ)	Available core begins at Z(KZ1) for P4, and at Z(KZ3) for FACTOR. (KZ1 < KZ3, but as it happens, P4 needs much less core than FACTOR.)

Method:

1. Use the state vector HS to select column numbers of basic variables. Store these backwards into HB, so that slacks will appear first and get dealt with directly by the transversal finder, TRNSVL. Count nonzeros, slacks, etc. in B and print one line of basis statistics.

2. Call P4 to determine the bump and spike structure of B. Most parameters just provide workspace. Useful output is
   - HPIVB — Row permutation
   - HPIVU — Permuted column numbers
   - HSPIKE — Coded bump and spike structure.

3. If there is a linear objective, make sure it pivots on its own row (because we want X(IOBJ) to be the objective value).

4. If I1FREQ=1, use HSPIKE to display the bump and spike structure on the printer.

5. Set HPIVI = inverse of HPIVR, ready for FACTOR. Initialize pivot tolerances TPIVB, TPIVC.

6. (Allocate core) Allocate storage for F, L and U, using the maximum value of NSPIKE and the values of KL and KU for the previous LU file (which usually will have been updated about 50 times and should give a fair indication of the relative sizes of L

and U during the next 50 updates). The five arrays HL and DL, HU and DU, and F, will occupy the whole of available core, Z(KZ3), ..., Z(MAXZ), in that order.

(a) Storage for F is allocated first, from the top down. The dimension of F is taken to be MAXSPK + NEWSPK, where MAXSPK is the maximum no. of spikes so far (in case of multiple attempts to factorize the same basis), and NEWSPK is estimated to include about 10 percent triangle swaps in FACTOR (each of which adds one spike), and/or up to somewhat less than KINV updates. (Not every update results in an increase in spikes, since a spike may be deleted during the update.)

(b) The remaining core is allocated to L and U in the same ratio as the previous LU file, just prior to the current INVERT, namely the ratio B = KU/KL. For safety, this ratio is moved inside the range (0.2, 5). At the start of a run, KL and KU are initialized in DRIVER to provide a rough guess for B as follows. For a warm start (anything other than CRASH), we guess that L will be somewhat bigger than U and therefore take KL=6, KU=4 to get B = 0.66. For CRASH, although L will be the whole of the initial (triangular basis) and U will be I, we guess that the rate of growth of U will be substantial, and therefore set KU = 2*KL to get B = 2.0.

7. (Factorize basis) Call FACTOR with the current list of basic columns in HPIVU. (This list will be permuted by FACTOR if column interchanges occur, and will be altered if any columns are rejected from the basis and replaced by slacks.) Test for error condition, which can only occur if insufficient storage was allocated for L, U or F. If necessary, repeat from 6 to try a different allocation of storage.

   (N.B. The logic for re-allocating storage for L, U and F should be improved for the case where there are many spikes and not much core. At present, if the total core available is only marginally greater than absolutely necessary, the estimates may fail again. It is far better to restart the run with more core for Z.)

8. (Compute basic X) Call SETX to compute the basic variables X(1), ..., X(M), using one step of iterative refinement (without accumulation of residuals in extended precision). The error flag will be set if the row check fails (i.e., if Ax = b is not sufficiently well satisfied). If necessary, tighten up the tolerances TPIVR, TPIVC and try again from 6. TPIVR must be made smaller than TMIN (output from FACTOR) or the pivot order will not change.

9. Initialize array HPIVU to be HPIVF (they start to differ during updates).

10. If any variables were rejected by FACTOR, check for superbasic slacks that are now in the basis. Call DELCOL if necessary to eliminate them from the superbasic set.

SUBROUTINE ITERCP(NN,NS)

Subroutines called:

None

Called by:

DRIVER

Purpose:

To output one line of the Iteration Log, according to the required Log Frequency, KLOG.

Parameters:

INTEGER	NN	(Input)	The no. of nonlinear variables.
INTEGER	NS	(Input)	The no. of superbasics.

Method:

1. If this is the first iteration since INVERT, initialize COMMON variable IHEAD to zero. (IHEAD is in COMMON so that it will retain its value between entries.)

2. Exit if MOD(ITN, KLOG) is nonzero. Output not wanted.

3. Increment IHEAD.

4. Branch if problem is nonlinear, or if NPHS>2. (More information is printed per line for non-simplex iterations.)

5. If IHEAD=1, this is the first iteration to be printed since INVERT. Print relevant Log heading.

6. Print one line of the Log.

SUBROUTINE LOADB(M,N,NN,NNO,MN,MAXS,NS,HS,HB,BL,BU,X,XN)

Subroutines called: | **Called by:**

None | DRIVEP

Purpose:

To input a bit map from file IOLDB describing the state of each variable, along with the column numbers and values for superbasic variables. (I.e., the format output by subroutine SAVEB.)

Parameters:

INTEGER	NS	(Output) The no. of superbasics loaded.
INTEGER*2	HS(N)	(Output) The state vector.
INTEGER*2	HB(MN)	Superbasic cols HB(M+1), ..., HB(M+NS) will be output.
REAL*8	X(MN)	Superbasic values X(M+1), ..., X(M+NS) will be output.
REAL*8	XN(NNO)	Any values corresponding to nonlinear variables will be output in XN.

Method:

Essentially the reverse of SAVEB, except that several consistency checks must be performed.

1. Print message 'BASIS TO BE LOADED FROM FILE <IOLDB>'.

2. Read and print first two card images. Extract M, N and NS values from the second card; check M and N for consistency with the current problem. Error exit if check fails.

3. Read the state vector HS. Exit if End of File.

4. Set NS=0. Load column numbers J and values XJ for superbasics, one pair per card.

   (a) Move XJ values inside bounds if necessary.

   (b) Skip if HS(J)=3 (basic).

   (c) Overwrite HS(J) with the value 2 (superbasic). This allows the user to change nonbasics to superbasics at specified values by simply adding cards at the end of a saved bit map, without having to change the HS vector.

   (d) If the value XJ is essentially on a bound, or if the limit on superbasics has already been reached, make variable J nonbasic at the appropriate bound.

   (e) Otherwise, increment NS and store J, XJ in HB(M+NS), X(M+NS) respectively.

(f) Store XJ into XN(J). (Not necessary for MINOS, but required for MINOS/GRG.)

5. Check consistency of the state vector HS.

(a) Check that nontasic variables are not specified to be at infinite bounds. Switch their state if necessary.

(b) Count the number of variables of each state 2 and 3. Error exit if these do not agree with NS and M.

SUBROUTINE LOADN(M,N,NN,NNO,MN,MAXS,NS,HS,HB,BL,BU,X,XN,ID1,ID2)

Subroutines called:

NMSRCH

Called by:

DRIVER

Purpose:

To input a basis description from file ILOAD, in the format output by subroutine DUMP. Variables are specified by state, name and value. This type of basis file is intended to be easier to modify than a PUNCH/INSERT deck.

Parameters:

INTEGER	NS	(Output) The no. of superbasics loaded.
INTEGER*2	HS(N)	(Output) The state vector.
INTEGER*2	HB(MN)	Superbasic cols HB(M+1), ..., HB(M+NS) will be output.
REAL*8	X(MN)	Superbasic values X(M+1), ..., X(M+NS) will be output. The first M locations of X contain the RHS, on entry and on exit.
REAL*8	XN(NNO)	Any values corresponding to nonlinear variables will be output in XN.
INTEGER	ID1(N)	Workspace to hold the left- and right-hand
INTEGER	ID2(N)	halves of the row and column names.

Method:

Straightforward, except that many consistency checks must be applied to ensure that exactly M variables end up basic, that no infinite bounds are specified, etc.

1. Print message 'LOAD BASIS BY NAMES -- FILE <ILOAD>'.

2. Read and print the first card image from file ILOAD. This should contain the problem name and the characters 'DUMP/LOAD'. The user can check visually that the correct file was loaded; otherwise it is not really possible to determine that the file does not belong to some other problem.

3. Read row and column names from scratch file into ID1 and ID2. Note that row names are first on the scratch file, but belong at the end of ID1 and ID2.

4. Set the state vector HS to make all variables nonbasic at their smallest bound (in absolute value).

5. Initialize counters and proceed to read and process cards one at a time, until an ENDATA card is found, or End of File. Each card contains a KEY, NAME and VALUE.

   (a) Call NMSRCH to determine a column no. J corresponding to NAME. Skip if not found (NMSRCH will have printed an error message and incremented the error counter).

(b) If J is a slack, convert row value XJ to logical value, using the RHS value stored in X. If necessary, move XJ values inside bounds.

(c) If KEY = 'BS', make the variable basic. Skip if it already has state 3, or if M basics have already been specified.

(d) If KEY = 'LL' or 'UL', make the variable nonbasic at the appropriate bound. Switch state if necessary to avoid infinite bounds.

(e) If KEY = 'SB', make the variable superbasic as long as it is not already at state 2 or 3. Skip if the no. of superbasics has already reached MAXS (the variable will remain nonbasic).

6. Print no. of basics and superbasics specified, the no. of cards read and the no. ignored.

7. (Partial basis) If fewer than M basics were specified, proceed to make logicals basic, starting from the left and skipping any that are already basic or superbasic. This will not necessarily give a good final basis. Actually it will fail if too many logicals were already superbasic -- there will not be a full set of M variables at state 3 on exit. The only safe way would be to add a loop to run through the superbasic list backwards, changing as many states to 3 as necessary.

8. Check that the logical on the objective row is basic. If not, swap it with the last basic variable.

```
      SUBROUTINE LPITN( M,M1,M2,N,NE,NP1,NF,MAXL,MAXU,NN,
     1   HA,HB,HE,HS,HPIVL,HPIVU,HPIVP,NL,NU,A,BL,BU,PI,X,Y,Y1,
     2   HL,DL,HU,DU,F )
```

Subroutines called: **Called by:**

BTRANU CHUZR FTRANL UNPACK DRIVER

Purpose:

To perform a normal simplex iteration after PRICE has selected variable JQ to enter the basis.

Parameters:

INTEGER	M1	(Input) M+1.
REAL*8	PI(M)	Not used.
REAL*8	X(M1)	(In,out) The first M locations contain values for the basic variables. X(M1) will be used for the incoming variable.
INTEGER*2	HB(M1)	(In,out) The list of basic variables. HB(M1) will similarly be used for the incoming var.
REAL*8	Y(M1)	Workspace to hold the search direction (which will be the updated column A(JQ)).
REAL*8	Y1(M)	(Output) Will contain the partially updated column A(JQ).

Method:

1. Set logical ATBND to indicate whether variable JQ is coming in from its upper bound or not. The sign of DJQ tells the story.

2. Set XJQ to the appropriate bound value. Must be zero if JQ is a free variable.

3. Unpack column A(JQ) into Y. Solve B*Y = A(JQ) by solving L*Y1 = Y, U*Y = Y1. The intermediate vector Y1 is required on exit by MODLU to update the LU factors of B.

4. If infeasible and a composite objective is being used, modify DJQ to be the reduced gradient associated with the sum of infeasibilities (uncontaminated by a multiple WTOBJ of the real objective). Exit with error flag set if the modified DJQ indicates that the sum of infeasibilities would not decrease. (DRIVER will reduce WTOBJ and try again.)

5. Exit with an INVERT REQUEST if the LU file has essentially no space left for CHUZR. (Actually, CHUZR does not need space if the solution is feasible, but INVERT would be needed pretty soon anyway.) Otherwise, save the current values of HB(M1) and X(M1), which may belong to some genuine superbasic, and install the corresponding values for variable JQ.

6. Call CHUZR to select JP -- the variable leaving the basis will be the JP-th variable in the list HB. Exit if JP=0 or THETA is very

large (problem is unbounded). Treat JP=M1 specially -- this means the incoming variable has reached its opposite bound. Sometimes, CHUZR will return JP<0. This means that the (-JP)-th variable is currently infeasible, and will become feasible this iteration when it leaves the basis. Record this fact and reverse the sign of JP.

7. Perform housekeeping to update the list of basic variables and to reset the state vector HS. Recompute THETA exactly, in case CHUZR used a perturbation.

8. If THETA > EPSO, update X to become X + (or -) THETA*Y. The value for the incoming variable is then in X(M1).

9. Restore the original values of HS(M1), X(M1) and exit.

```
      SUBROUTINE MINOS( Z,NWCORE,ISPECS,ISCRCH,IPSOLN,
     1                  IERROR,NROWS,NCOLS,LXS,LXL,LPI,LHS,LFREE,NFREE )
```

Subroutines called: **Called by:**

DRIVER INITLZ MPSIN SPECS GO
SPECS2

Purpose:

To request input of the SPECS file and MPS file, and to call the solving routine DRIVER. The argument lists in the calls to other routines here look pretty ugly, but the actual routines being called (in particular, MPSIN and DRIVER) are reasonably normal. However, great care is required if ever anything is modified.

Parameters:

REAL*8	Z(NWCORE)	(Output) The array of core, used for all workspace and to return certain array information.
INTEGER	NWCORE	(Input) Dimension of Z.
INTEGER	ISPECS	(Input) Unit no. from which the SPECS file is to be read.
INTEGER	IPSOLN	(Input) Should be positive if the solution is to be output using the standard output routine, SCIN. Should be zero otherwise (e.g. if the user wishes to do further computation on the solution).
INTEGER	IERROR	(Output) -1 if EOF occurred while trying to read from file ISPECS (signals end of problems). 0 if optimal solution was found. 1 if problem was infeasible. 2 if problem was unbounded. 3 if iteration limit was exceeded. 4 if iterations were terminated by some other error condition. 30 if there was not enough core to input the MPS file (NWCORE too small). 40 if some other fatal error occurred during input of the MPS file.
INTEGER	NROWS	(Output) No. of rows in the constraint matrix.
INTEGER	NCOLS	(Output) No. of cols in the constraint matrix.
INTEGER	LXS	(Output) Address in Z of the solution vector. The structurals form a vector of length NCOLS, starting at Z(LXS).
INTEGER	LXL	(Output) Address in Z of the slack variables. These form a vector of length NROWS, starting at Z(LXL). (Alternatively, the structurals, RHS and slacks form a vector of length NCOLS+1+NROWS, starting at Z(LXS).)
INTEGER	LPI	(Output) Address in Z of the dual solution vector, PI.
INTEGER	LHS	(Output) Address in Z of the state vector HS.

		This is an INTEGER*2 array of length NCOLS+1+NROWS.
INTEGER	LFREE	(Output) Address in Z of the first free location in Z.
INTEGER	NFREE	(Output) No. of free words in Z. Words Z(LFREE), ..., Z(NWCORE) are not used to return any of the above solution vectors, and hence may be used as workspace by the calling routine.

If necessary, various other address pointers could be added as output parameters. Those pointing to the constraint matrix and bounds are the most likely, namely KHE,KHA,KAX,KBL,KBU.

Method:

1. Call INITLZ to initialize a few COMMON variables.

2. Call SPECS to look for a SPECS file and output it in special format to the scratch file. If no SPECS file, exit with IERROR = -1.

3. Call SPECS2 to input the SPECS parameters from the scratch file. This returns various dimension parameters.

4. Call MPSIN to input the MPS file and to allocate all array storage. Most of the parameters return starting addresses within Z, for the arrays used by DRIVER.

5. Call DRIVER to solve the problem.

6. Exit.

SUBROUTINE MKLIST(M,M1,NE,NP1,NZ,HA,HE,HB,IP,IRN)

Subroutines called:

None

Called by:

P4

Purpose:

To set up a column list, containing the positions of nonzeros in the set of columns defined by array HB.

Parameters:

INTEGER	M	(Input)	M+1.
INTEGER	NZ	(Input)	Total no. of nonzeros in the list.
INTEGER*2	IP(M1)	(Output)	IP(j) will point to the start of the j-th column, in IRN.
INTEGER*2	IRN(NZ)	(Output)	The column list.

Method:

Obvious.

```
      SUBROUTINE MODLU( M,M2,NP,MAXL,MAXU,HPIVL,HPIVU,HPIVP,NL,NU,
     1                  HL,DL,HU,DU,P,HB,X,Y,Y1,JP )
```

Subroutines called: | **Called by:**

None | DRIVER

Purpose:

To modify the LU factorization of the basis when the JP-th column is replaced by some vector A(JQ), using Gaussian elimination with row interchanges to maintain numerical stability, a la Bartels and Golub.

Parameters:

INTEGER*2	HB(M)	(In,out) List of basic variables. Must be permuted to match any row interchanges.
REAL*8	X(M)	(In,out) Values of basic variables. Same comment.
REAL*8	Y(M)	Work vector to hold the JP-th row of U.
REAL*8	Y1(M)	(Input) Contains the solution of the system L*Y1 = A(JQ). Gets overwritten.

Method:

That of Bartels and Golub, as implemented by Saunders (1976). The tolerance TOLSWP (0.0 <= TOLSWP <= 1.0) is used in the test for row interchanges. 1.0 gives the original Bartels and Golub; 0.0 simulates Forrest and Tomlin. In practice, 0.99 or 0.9 is plenty big enough to preserve stability, and reduces work very slightly by avoiding a few interchanges.

1. Increase NSPIKE and make JP a new spike row.

2. Use HPIVP to pick out all spike-row elements from Y1 and form a new column of P. Reset such elements of Y1 to zero, so they will not be packed up into U. Save pivot element, which will become part of the JP-th row of [U Y1].

3. Pack remaining nonzeros in Y1 to form a new column of U.

4. Test if the JP-th column of B is a spike. If so, the JP-th row of "U" comes from P, otherwise from U. The test involves jumping out of a loop on JQ, if HPIVP(JQ) = JP. This test will always be satisfied at least the last time round, for JQ=NSPIKE, because of 1 above. Hence, program will not fall through the loop.

5. Set Y equal to the JP-th row of "U". If a spike is not being deleted, this means running through HU, DU looking for nonzeros in row JP, which are physically reset to zero. Otherwise, JQ points to the correct row of P. Install this row into Y while deleting the JQ-th row and column from P. Also, JU points to the spike being deleted; make HPIVU(JU) negative to indicate to FTRANU and BTRANU that this column has been deleted.

6. At this stage, F is almost upper triangular, except for its bottom row, which is stored in Y. Now eliminate any nonzeros in Y. This will be straightforward Gaussian elimination with row interchanges. The pivot row for Y (in IPVIY) is initially JP. All pivot rows (IPIVY, HPIVF, HPIVU) will remain unaltered unless a row interchange is required.

7. Scan Y forwards looking for a nonzero. Suppose Y(j) is found to be significant. Compare with the corresponding diagonal of F as follows:

```
if  abs( diagonal of F )  <  TOLSWP*abs( Y(j) )
then interchange.
```

8. If no interchange, add appropriate multiple of row of F to Y.

9. Otherwise, interchange Y with the row of F and perform the elimination simultaneously. In this case, the pivot row for F (and U) is swapped with IPIVY. Also, the relevant components of BB and X are interchanged.

10. Pack the multiplier and pivot row (negative if an interchange occurred) into the I file, and repeat from 7 for the next nonzero in Y.

11. If any elements of Y were eliminated (often there will be none), increment NETAR and fill in NL, HPIVL for the new transformation.

12. Test the final diagonal of F. Request INVERT if smaller than EPS1 (EPS**(2/3)). This test should really be relative to the norm of A(JQ), but we assume that all columns have approximately unit norm.

AD-A051 536 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 9/2
MINOS SYSTEM MANUAL.(U)
DEC 77 M A SAUNDERS N00014-75-C-0865
UNCLASSIFIED SOL-77-31 ARO-12215.22-M NL

2 OF 2
AD
A051536

END
DATE
FILMED
4-78
DDC

SUBROUTINE MOVE(R1,R2,K,N1,N2)

Subroutines called: | **Called by:**

None | MPSIN

Purpose:

To copy array R2 into a specified part of array R1. This is required by MPSIN during input, once the dimensions of the problem are known. In particular, the bounds for the slacks have to be moved into their final position. To do this, a subroutine call is required in order to maintain alignment of REAL bounds within the REAL*8 array Z that MPSIN has to deal with.

Parameters:

REAL	R1(N1)	(Output) R1(K+1), ..., R(K+N2) will contain the contents of R2.
REAL	R2(N2)	(Input)
INTEGER	K	(Input) The required offset in R1.

Method:

Trivial, as follows:

```
      DO 10 J=1,N2
   10 R1(K+J) = R2(J)
```

```
      SUBROUTINE MPS( NCALL,
     1   LEN,NEN,NCOLL,MROWS,MCOLS,MELMS,M,MN,N,NE,NP1,NN,NN0,NS,MAXS,
     2   NCARD,HRTYPE,NAME1R,NAME2R,KEYNAM,
     3   HE,HA,A,BL,BU,HB,XN,NAME1C,NAME2C )
```

Subroutines called: | **Called by:**

HASH NMSRCH | MPSIN

Purpose:

To input constraint data in standard MPS format. This is the format used for specifying large-scale linear programming problems (the "CONVERT" data format of IBM's MPS/360 and MPSX/370).

Nonzero elements in the constraint matrix A are specified column-wise. All elements in a column must be specified on consective cards (there is no error check to detect split columns).

Three entries are required to input various sections of the data, as specified by the parameter NCALL, thus:

NCALL =	1	Input NAME and ROWS
	2	Input COLUMNS, RHS and RANGES
	3	Input BOUNDS and INITIAL BOUNDS

The RANGES and BOUNDS sections need not be present. Data should be terminated by an ENDATA card.

Parameters:

INTEGER	NCALL	(Input)	Defined above.
INTEGER	LEN	(Input)	Length of hash table for row names.
INTEGER	NEN	(Input)	No. of distinct entries in hash table.
INTEGER	NCOLL	(Input)	No. of collisions during searching of hash table.
INTEGER	MROWS	(Input)	Maximum no. of rows allowed for.
INTEGER	MCOLS	(Input)	Maximum no. of cols allowed for.
INTEGER	MELMS	(Input)	Maximum no. of matrix elements allowed for.
INTEGER	M	(Out,in)	The actual no. of rows found.
INTEGER	NCARD(6)	(Output)	No. of cards in each section of data.
INTEGER*2	HRTYPE(MROWS)	(Out,in)	Codes constraint types as follows:

-1	G	>=
0	E	=
1	L	<=
2	N	Non-binding

INTEGER*2	NAME1R(MROWS),NAME2R(MROWS)	(Out,in)	Left and right halves of row names.
INTEGER*2	NAME1C(MCOLS),NAME2C(MCOLS)	(Out,in)	Left and right halves of col names.
INTEGER	KEYNAM(LEN)	(Out,in)	The hash table pointers.
REAL*8	XN(NN0)	(Output)	After entry 3, returns initial values for nonlinear variables.

Method:

The required OBJ, RHS, RANGE and BOUND names are in the /MPSCOM/ arrays MOBJ(2), MRHS(2), MRNG(2), MBND(2). If these names are blank, the first names encountered in the MPS data will be used.

Several other parameters reside in labelled COMMON areas. In particular, the variables in /MPSLCI/ are not used by any other subroutine, but are in a COMMON area to ensure that they retain their values between entries to MPS.

NCALL = 1 Input NAME and ROWS.

1. Initialize various counters. Zero the hash table pointers.

2. (NAME)
   (a) Read first card from file INPUT. Error message if it is not the NAME card. Extract name from it anyway and proceed.

   (b) Skip if first card was the ROWS card. Otherwise, read second card. Error message if not the ROWS card. Proceed as if it were anyway.

3. (ROWS)
   Read row types and names until the COLUMNS card is found (or EOF).

   (a) Allow types L, G, E, N to occur in column 2 or column 3.

   (b) Make IOBJ point to the appropriate N row.

   (c) Enter each new row name into the hash table and check for duplicates.

   (d) Write row names out to the scratch file.

NCALL = 2 Input COLUMNS, RHS and RANGES.

1. (COLUMNS)
   Read data until RHS, BOUNDS or ENDATA cards are found. For each card:

   (a) If the column name is different from that on the previous card, increment the column counter N, save the new name in core and also write it out to the scratch file. Error message if the old column had no elements, unless it belongs to a nonlinear variable; for nonlinears, enter a dummy zero in row 1, so that Fortran loops can go round once without damage.

   (b) Extract 0, 1 or 2 row names and matrix coefficients. Look up any non-blank row name in the hash table. Error message if not found.

   (c) Count up total elements NE, ignored elements NAO, etc.

2. (RHS)
Very similar to CCLUMNS input, except for some juggling tc select one of possibly multiple RHS's. If no RHS elements are found, or if all are zero, insert a dummy RHS with zero in row 1. (This serves as a place holder for column no. KRHS.)

3. (RANGES)
   (a) Set default bounds on the slacks, according to the row types (stored in HSTYPE during entry 1).

   (b) Exit if there is nc range set.

   (c) Read cards until required RANGE name is found. Extract row names and range values in similar fashion to COLUMNS and RHS input.

   (d) Treat range values according to the row types.

NCALL = 3 Input BCUNDS and INITIAL BOUNDS.

1. Add a full identity matrix to the end of the constraint matrix (one nonzero per slack).

2. Set bounds on structurals tc their default values BSTRUC(1) and BSTRUC(2). These are usually 0.0 and 1.0E+30.

3. Skip if the ENDATA card has already been read. Otherwise, read data until the required bounds set is found, or the reserved name INITIAL is encountered.

4. Fcr normal bounds, read one bound type, cclumn name and value per card. Do a linear search to find which column it is, starting from the point where the previcus search finished. (It is likely that bounds will be roughly in natural order, so the linear search will usually terminate very quickly.)

5. Before reading any INITIAL bounds, set XN to the smallest bound (in absolute value) cn the ccrresponding nonlinear variables. Then read through all remaining data cards, until ENDATA is found. Even if the bounds are to be ignored, the cards must be read in case further problems occur later in the input file.

6. Count the no. of rows and cclumns of various type (NORMAL, FREE, FIXED, BOUNDED) and output as MATRIX STATISTICS. In this context, NORMAL means variables with bounds 0 to INFINITY or -INFINITY to 0.

```
      SUBROUTINE MPSIN( Z,MAXZ,MROWS,MCOLS,MELMS,
     1    M,M2,MN,MNN,N,NE,NP1,NR,NN,NNO,MAXS,NS,
     2    KHR,KHE,KHA,KAX,KBL,KBU,KHB,KXN,
     3    RHS,KPL,KPU,KPF,
     4    KNI,KNO,KRX,KCX,KXX,KEI,
     5    KGR,KYX,KY1,KG1,KGA,KGB )
```

Subroutines called: | **Called by:**

MOVE MPS | MINOS

Purpose:

To input constraint data, and to allocate all storage except that required by the LU file.

Parameters:

REAL*8 Z(MAXZ) (Output) The constraint matrix and bounds will be in various parts of Z, pointed to by the parameters KHE,KHA,KAX,KBL,KBU. Certain other arrays will also contain useful information, namely, those pointed to by KHB,KXN,KHS.

INTEGER KPL,KPU,KPF, ..., KGA,KGB
(Output) These are starting addresses in Z for various other arrays that will be required by DRIVER for the solving procedures.

Method:

Three calls are required to subroutine MPS. For the first, we don't know M or N. For the second we don't know N. For the third, the dimensions of the problem are known.

1. Allocate storage in Z for row types and row names, using the value MROWS which must be an over-estimate of the actual no. of rows in the data about to be read.

2. Find the nearest prime number larger than MROWS. This will be LEN, the length of the hash table for holding row names. Allocate that amount of storage in Z.

4. Allocate more storage in Z for arrays HE,HA,A (from the bottom up starting at end of hash table KEYNAM), and for BL,BU,NAME1C,NAME2C (from top down). At this stage, BL and BU are bounds for the slacks only, to be defined during RANGES input. Also, MCOLS must be an over-estimate of the no. of structurals.

5. Call MPS with entry 2, to input COLUMNS, RHS and RANGES.

6. Now that M and N are known, we can compact storage. First, set up new (and final) starting addresses for existing arrays and for some new ones HB,XN,HS that are required during BOUNDS input. Then copy existing arrays into their new positions in Z, starting from the

bottom. In particular, the existing bounds on slacks must go into the end of the new N-dimensional arrays BL and BU. Word alignment can be maintained only with the help of subroutine MOVE.

7. Call MPS with entry 3, to input BOUNDS (on structurals) and INITIAL BOUNDS (on nonlinear variables).

8. Allocate starting addresses in Z for all other arrays required by DRIVER.

```
      SUBROUTINE NEWPIC(EPS, T, ETA, XLAMDA, U, FU, GU,
     *    XMIN, FMIN, GMIN, XW, FW, GW, A, B, OLDF,
     *    B1, SCXBD, E, D, RR, SS, GTEST1, GTEST2, TOL,
     *    ILOC, ITEST)
```

Subroutines called:

None

Called by:

SEARCH

Purpose:

To find a step length during a linesearch. For full documentation see Gill, Murray, Picken, Barber and Wright, NPL Algorithms Library, Ref. No. E4/16/F, DNAC, National Physical Laboratory, Teddington. (Crown Copyright Reserved.)

```
      SUBROUTINE NMSRCH( N,ID1,ID2,NAME1,NAME2,
     1   NCARD,NOTFND,MAXMSG,J1,J2,JMARK,JFOUND )
```

Subroutines called: | **Called by:**

None | INSERT LOADN MPS

Purpose:

To search arrays ID1 and ID2 for the name defined by NAME1 and NAME2. Names in these variables are in 2A4 format.

Parameters:

INTEGER	N	(Input)	No. of names to be searched.
INTEGER	ID1(N)	(Input)	Left half of names to be searched.
INTEGER	ID2(N)	(Input)	Right half of names to be searched.
INTEGER	NAME1,NAME2	(Input)	Left and right halves of name being looked up.
INTEGER	NCARD	(Input)	Card no. from which NAME1,NAME2 came.
INTEGER	NOTFND	(In,out)	Accumulates the no. of times the input name was not found.
INTEGER	MAXMSG	(Input)	Limit on no. of error messages allowed.
INTEGER	J1	(Input)	Marks start of names to be searched.
INTEGER	J2	(Input)	Marks end of names to be searched.
INTEGER	JMARK	(In,out)	On input, marks where the previous search ended. On output, marks end of current search.
INTEGER	JFOUND	(Output)	Will be 0 if the name was not found. Otherwise, points to position of name in ID1,2.

Method:

Nothing fancy -- just a linear search, split into 2 halves (JMARK to J2, then J1 to JMARK), on the assumption that the names being looked up are likely to be in roughly the same order as the list of names in ID1 and ID2.

```
      SUBROUTINE PACKIU( M,M2,NP,MAXL,MAXU,IPIV,
     1    HPIVL,HPIVP,HPIVR,NL,NU,HL,HU,DL,DU,P,Y,IBGN,K )
```

Subroutines called: | **Called by:**

None | FACTOR

Purpose:

To pack up significant nonzeros in L, U and P during INVERT. The K-th column of the basis has just been processed by FACTOR and it is a spike.

Parameters:

INTEGER	IPIV	(Input) Pivot row for the spike.
INTEGER*2	HPIVL(M2)	List of pivot rows for the columns of L. IPIV will be added to the list.
INTEGER*2	HPIVP(M2)	List of pivot rows for P (previous spike rows). IPIV will be added.
INTEGER*2	HPIVR(M2)	(Input) The full row permutation being used by FACTOR.
REAL*8	Y(M)	(Input) The transformed spike, containing the nonzeros to be packed.
INTEGER	IBGN	(Input) Marks the beginning of the current bump.
INTEGER	K	Position of this spike in the basis.

Method:

1. Increase NSPIKE by 1; add IPIV to array HPIVP. Use HPIVP to pack up nonzeros in all spike rows. Reset such elements Y(i) to zero, so they will not be packed up again for U. Save pivot element, PIV, for storing in the L file.

2. Use HPIVR to pack up elements in U. These are in rows HPIVR(i), i=IBGN, ..., K.

3. There cannot be any elements to add to L if this is the last column in the basis (K=M). Otherwise, pack remaining elements into L. These are in rows HPIVR(i), i=K+1, ..., M. Note: The transformation is going to be used as if the nonzeros, starting from the pivot row, were
   1.0, Y(i1)/PIV, Y(i2)/PIV, ...
   because the pivot element goes into P rather than L. However, to avoid the divisions we store the elements directly as
   PIV, Y(i1), Y(i2), ...
   and communicate the fact to FTRANL, BTRANL by setting the pivot index HL(KL) to be negative, viz. -IPIV.

```
      SUBROUTINE PRICE( M,MN,N,NE,NP1,NN,NNO,NS,MAXS,
     1   MA,HE,HS,HP,A,BL,BU,C,X,PI,GRD,G )
```

Subroutines called: **Called by:**

None DRIVER

Purpose:

To select one or more nonbasic variables with favorable reduced gradients, for addition to the superbasic set. The class of candidates depends on certain parameters as follows:

1. (BANDAID) If NREJ>0, give preference to a list of variables in array JREJ that were rejected from the basis by the last INVERT.
2. (MULTIPLE PRICE) If NMULPR>0, scan all nonbasic columns and select up to NMULPR candidates.
3. (PARTIAL PRICE) Otherwise, scan just the next partition of A, as well as all slacks.

Parameters:

REAL*8	PI(M)	(Input) The pricing vector.
REAL*8	GRD(MN)	Not used.
INTEGER*2	HB(MN)	(In,out) If NMULPR>0, selected column numbers will be added to this array, starting at HB(NS+1).
REAL*8	G(MAXS)	(In,out) If NMULPR>0, the corresponding reduced gradients will be added to this array, starting at G(NS+1).

Method:

This is one of the more intricately (i.e., obscurely) coded routines, because the various options were added in several stages. However, the important point is that in all cases, the variables J1, J2, J3 determine which columns are to be priced (see the loop beginning DO 700 JJ=J1,J3). The possible values are:

	J1	J2	J3
BANDAID	1	NREJ	NREJ
MULTIPLE PRICE	1	NSTRUC	J2+M
PARTIAL PRICE	J1	J1+NPRC	J2+M

where

NREJ	is the no. of rejected variables remaining;
NSTRUC=KRHS-1	marks the end of structurals;
NPRC=NSTRUC/NPARPR	is the no. of columns in each partition of A.
J1	for PARTIAL PRICE is 1+(a multiple of NPRC) or, equivalently, (previous J2)+1 (mod NSTRUC).
J3=J2+M	for the last two cases in order to include the M slacks.

1. If PARTIAL PRICE, move pointers J1,J2 to bracket the next partition of A.

2. Set TOLD, the tolerance to be used in measuring whether a reduced gradient DJ is significant. If the problem is infeasible, TOLD is always TOLDJ1*PINORM. (Scaling by PINORM is not vital here, except perhaps if a very large weight is used for the composite objective.) If the whole of A is going to be scanned during this PRICE (i.e. if NPARPR=1 or MULTIPLE PRICE is in effect), we might as well allow through anything that is going to be regarded as significant before optimality is declared. Only if NPARPR>1 is TOLD going to be decreased dynamically, by setting it to TOLDJ which is reduced systematically as described in 7 below.

3. The first three lines in loop 700 define J, the next column to be priced. Skip basics and superbasics immediately. Skip fixed nonbasics next. Then treat a column specially if it is a slack (DJ = - appropriate PI value). Otherwise, the reduced gradient is

   DJ = grad(J) - PI*A(J)

   where grad(J) is 0 or C(J) for linear and nonlinear variables if the solution is feasible, otherwise 0 for all variables.

4. The required sign for a DJ depends on the state of the variable. The quantity D is defined to be either DJ or -DJ or abs(DJ), such that a positive value for D means a favorable DJ.

5. If MULTIPLE PRICE, compare DJ with the list of values in G. If necessary, add J and DJ to HB and G, such that the values in G remain in descending order of magnitude. (This section was coded by Richard Asmuth.)

6. Update DJMAX, DJQ, JQ to record the best DJ found so far.

7. On exit from loop 700, determine if any suitable column was found (NONOPT>0). If so, exit. Otherwise, the following cases arise:

BANDAID	Revert to MULTIPLE or PARTIAL PRICE.
MULTIPLE PRICE	Set NEWSB = number of columns added to HB, G; if necessary, delete any small entries from the end of these lists.
PARTIAL PRICE	If not all of A has been scanned, repeat from 1. Otherwise, test if the largest DJ encountered in all partitions (DJSUP) is smaller than the level TOLDJ3*PINORM, which is taken to be insignificant. If so, exit. Otherwise, reduce TOLDJ to a tenth of DJSUP and repeat from 1. (At least one column must get picked up next time. It would be better to exit with the column corresponding to DJSUP directly. However, the reduction of TOLDJ occurs only once or twice during a run.)

SUBROUTINE PUNCH(M,N,KRHS,HS,BL,BU,Y,ID1,ID2)

Subroutines called:	Called by:
None	DRIVER

Purpose:

To output a basis description to file IPNCH, in a format compatible with some of the commercial LP systems. Each card image is of the form

KEY NAME1 NAME2 VALUE

where KEY may be LL, UL, XL, XU or SB. (Type SB is a generalization of the standard format, necessary for description of nonbasic solutions.)

The file produced by PUNCH may be used as input to INSERT.

Parameters:

REAL*8	Y(M)	(Input) Contains solution values for all variables (structurals, RHS and logicals).
INTEGER	ID1(M)	Work vector to hold left part of row names.
INTEGER	ID2(M)	Work vector to hold right part of row names.

Method:

1. Read row names from the scratch file into ID1, ID2.

2. Process each structural in turn.

   (a) Read column name from scratch file, to be used as NAME1.

   (b) (Nonbasics or superbasics) Skip nonbasics that are at a zero bound or are fixed. Otherwise, use the state vector HS to index the local array KEY to get the correct indicator (LL, UL or SB). Output with NAME2 blank.

   (c) (Basics) Scan the state vector to find the next logical that is not basic (it may be nonbasic or superbasic). The corresponding row name will be NAME2. Select the correct indicator (XL or XU) and output.

   Note: XL means the logical (not the row) is at its lower bound.

3. Output any superbasic logicals, using indicator SB and NAME2 blank. It is important that these be output last.

4. Print message to indicate successful PUNCH.

```
      SUBROUTINE P3( M,M1,NBELEM,NBUMP,HSPIKE,HC,HR,HI,HJ,
     1   HSC,HSR,HCLIST,HRLIST,NSPIKE )
```

Subroutines called: **Called by:**

None P4

Purpose:

To permute the rows and columns of a square matrix B so that it is close to being lower triangular, apart from a few columns ("spikes") which have nonzeros above the diagonal.

Parameters:

INTEGER	M	(Input) Dimension of B. Will be the size of just one of the bumps in the whole basis.
INTEGER	M1	(Input) M+1.
INTEGER	NBELEM	(Input) No. of nonzeros in B.
INTEGER	NBUMP	(Input) The number of this bump, used only for storing into array HSPIKE.
INTEGER*2	HSPIKE(M)	(Output) HSPIKE(j) will be -NBUMP if column j is a spike, +NBUMP otherwise.
INTEGER*2	HC(M)	(Output) The assigned column order.
INTEGER*2	HR(M)	(Output) The assigned row order.
INTEGER*2	HI(M)	(Input) Column counts.
INTEGER*2	HJ(M)	Work vector to hold row counts.
INTEGER*2	HSC(M1)	(Input) HSC(j) points to the start of column j in HCLIST.
INTEGER*2	HSR(M1)	Work vector to point to start of rows in HRLIST.
INTEGER*2	HCLIST(NBELEM)	(Input) Column list.
INTEGER*2	HRLIST(NBELEM)	Work vector to hold row list.
INTEGER	NSPIKE	(Output) No. of spikes found.

Method:

The bulk of P3 was coded directly from the description of the Preassigned Pivot Procedure by Hellerman and Rarick, Math. Programming 1, 2, November 1971. It has been modified as follows:

1. The backward and forward triangle sections have been deleted, since they are now handled by P4. Hence, this version of P3 is intended for use on an irreducible square matrix.

2. The method used for computing the Tally Function is more efficient and follows that due to R. Peacock of CDC.

```
      SUBROUTINE P4( M,M1,N,NE,NP1,NZ2,HSPIKE,HA,HE,HS,HB,HC,HR,HI, J,
     1   NUMB,HP,B,W1,W2,HRN,NBUMP,NSPIKE )
```

Subroutines called: **Called by:**

BUMPS MKLIST P3 TRNSVL INVERT

Purpose:

To permute the rows and columns of the basis matrix B into the bump and spike ordering of Hellerman and Rarick. The permuted B will be block lower triangular.

Parameters:

INTEGER	NZ2	(Input) Twice the no. of nonzeros in B.
INTEGER*2	HSPIKE(M)	(Output) Codes the bump and spike structure in the following way. Suppose B turns out to have 2 bumps of dimension 4, each with 2 spikes. HSPIKE might then be 0 0 0 1 1 -1 -1 0 0 2 2 -2 -2 0 0 0 which would mean that the last two columns in each bump are spikes, the forward and backward triangles are both of size 3, and the two bumps are separated by 2 triangle columns.
INTEGER*2	HB(M)	On input, contains column nos. of basic vars. INVERT sets this list up backwards, so slacks appear first.
INTEGER*2	HC(M)	(Output) Will contain the column nos. in permuted order, ready for FACTOR.
INTEGER*2	HR(M)	(Output) Will specify the preassigned pivot rows for the reordered columns.
INTEGER*2	HI(M),HJ(M),NUMB(M),HP(M),B(M),W1(M),W2(M)	All work vectors.
INTEGER*2	HRN(NZ2)	Work vector, will be used for column and row lists by P3.
INTEGER	NBUMP	(Output) Final no. of bumps.
INTEGER	NSPIKE	(Output) Final no. of spikes in all bumps.

Method:

1. Set up column list in order specified by HB. This order is arbitrary except for safety the slacks are listed first.

2. Call TRNSVL to find a maximal transversal. HP will contain the resulting column permutation.

3. If B is structurally singular (NREJ.NE.0), an indirect method is used to find which columns in HB were selected and which of those must be replaced by suitable slacks.

4. Reorder HB and make a new column list.

5. Call BUMPS to find a symmetric permutation HP that makes B block upper triangular.

6. Since we want B to be block lower triangular, we now process the output from BUMPS in reverse order, in particular the array B(NBLK) which points to the beginning of each bump within array HE.

7. For each bump, pointers P1,P2 mark the beginning and end going backwards, and Q0,Q1 mark the beginning and end going forwards when results are stored finally in HC and HR.

8. Bumps of dimension 1 or 2 are treated directly. Otherwise, the column list for a bump is set up by taking the row nos. in each column in the bump and using them to index the array NUMB which is output by BUMPS. If the relevant value of NUMB lies between P1 and P2 then the element lies inside the bump.

9. Call P3 and accumulate results forwards in HSPIKE, HC, HR.

SUBROUTINE RESETR(NN,NR,NS,R,COND)

Subroutines called:

None

Called by:

DRIVER INVERT RGITN

Purpose:

Resets R to the identity matrix (R upper triangular, stored by columns). Called when the first feasible point is found, or as one of the recovery actions if the linesearch fails.

Parameters:

REAL*8 COND (Output) Reset to 1.0.

```
      SUBROUTINE RGITN( M,M2,MN,MNN,N,NE,NP1,NF,NR,NN,NN0,MAXS,NS,
     1    MAXL,MAXU,
     2    HSPIKE,HA,HB,HE,HS,HPIVL,HPIVU,HPIVF,NL,NU,A,BL,BU,R,PI,X,
     3    GRD,Y,Y1,XN,C,G1,G,CNEW,
     4    HL,DL,HU,DU,F )
```

Subroutines called:

ADDCOL ALIGN BTRANU CALCG
CG CHUZQ CHUZR CCMDFR
DELCCL DOT1 FTRANI FUNGRD
GETGRD RESETR RTRSOL R1PROD
SEARCH SETPI SQUEEZ UNPACK

Called by:

DRIVER

Purpose:

To perform an iteration of the reduced-gradient method.

Parameters:

All defined previously.

Method:

NPHS is either 3 or 4 on entry to RGITN. The only difference is that when NPHS=3, a list of nonbasic variables (set up by PRICE) must first be added to the superbasic set.

Thereafter (for both phases) the routine performs one optimization step on the current basics and superbasics. In general it may be entered with NPHS=4 for several consecutive iterations, during which the number of superbasics will either remain constant or decrease.

1. (Initialize) Set up various logical variables to determine the current state of the problem (FEASBL, INFSBL, LINEAR, NONLIN, VARMET). For example, LINEAR is true if the current solution is infeasible, or if there are no nonlinear variables; the linesearch and quasi-Newton or conjugate-gradient steps can then be skipped.

2. (Add superbasics) If NPHS=3, the nonbasic variables listed in HB(j), j=M+NS+1, ..., M+NS+NEWSB have been set up by PRICE to become superbasic. The first variable in the list has the largest reduced gradient, DJQ. If DJQ is not sufficiently larger than RGNORM (the norm of the reduced-gradient vector for the existing superbasics), ignore the list and proceed as if NPHS=4. Otherwise, add the new superbasics one by one and increment NS accordingly. If necessary, add new columns to R. Take failure exit if NS has already reached MAXS. Set VARMET = FALSE if NS exceeds MAXR (Hessian dimension).

3. (Compute search direction)
   (a) Generate a search direction s for the superbasics, storing s in Y(M+j), j=1, ..., NS.
   If LINEAR, s is the negative reduced gradient vector, -G.
   If VARMET, R'Rs = -G.

If CONJGR, s will already have been set up by the previous entry to RGITN, except if RESTRT = TRUE.

(b) Solve By = -Rs to get corresponding search direction y for basics. Store y in Y(j), j=1, ..., M.

4. (CHUZR) Find which variable reaches one of its bounds first when X is changed to X + THETA*Y (THETA >= 0). Set JP accordingly. Problem is unbounded if LINEAR and JP=0.

5. (Linesearch) Skip if LINEAR or if THETA is essentially zero. Otherwise, call SEARCH to determine a new step length THETA. Declare problem unbounded if necessary. If SEARCH fails on one or more consecutive iterations, try the following in turn:

(a) Reset the Hessian (R = I) and try again.
(b) Call INVERT and try again.
(c) Switch to NPHS=3 to ask for PRICE (more superbasics) and try again.
(d) Repeat (c) up to 4 times.
(e) Give up for the day. Try again tomorrow.

6. (Find new reduced gradient) Skip if LINEAR or THETA=0. Otherwise, compute new PI vector and new reduced-gradient vector GNEW. If VARMET, compute Y = GNEW - G, etc. ready for quasi- Newton update to Hessian.

7. (Basis change) If 1 <= JP <= M, a basic variable reached its bound.
(a) If VARMET, do quasi-Newton update to R.
(b) Solve B'y = e(JP).
(c) Call CHUZQ to find the KQ-th superbasic to replace basic. This will be column no. JQ1.
(d) If VARMET, call R1PSCD to modify R to account for basis change.
(e) Modify PI using y from (b) above.
(f) Solve L*Y1 = A(JQ1) to prepare for update of LU factors of basis.
(g) Go to 8(b).

8. (Delete one superbasic) If JP > M, a superbasic variable reached its bound. Set KQ = JQ - M.
(a) If VARMET, do quasi-Newton update to R.
(b) Call DELCOL to delete the KQ-th superbasic from X, HS and perhaps R.
(c) Recompute reduced-gradient G and norm RGNORM.
(d) Go to 10.

9. (Unconstrained step) If JP=0, a minimum was found along X + THETA*Y before any variable hit a bound.
(a) If CONJGR, call CG to compute search direction for superbasics for next iteration.
(b) If VARMET, do quasi-Newton update to R.
(c) Set G = GNEW.

10. (Estimate condition no. of reduced Hessian) Compute DMAX and DMIN, the largest and smallest diagonal elements of R. Set COND = (DMAX/DMIN)**2.

11. (Convergence test for current subspace) Determine whether to stay in phase 4 (another iteration with the same superbasics) or to ask

for phase 3 (PRICE). See Murtagh and Saunders (1978), section 3.4.

SUBROUTINE RTRSCI(R,Y,NR,N)

Subroutines called:

None

Called by:

RGITN

Purpose:

To solve the system R'Ry = z.

Parameters:

INTEGER	N	(Input) Current dimension of R.
REAL*8	Y(N)	Initially holds z. Overwritten by y.

Method:

Forward substitution followed by back-substitution. Both are arranged to run through the columns of R, in order to access memory sequentially. This improves execution speed on some machines, e.g. those with a cache memory.

SUBROUTINE R1ADD(R,Y,NR,N,TOLZ)

Subroutines called:

None

Called by:

R1MOD R1PROD

Purpose:

To modify R such that (new R'R) = (old R'R) + yy'.

Parameters:

INTEGER	N	(Input) Current dimension of R.
REAL*8	Y(N)	On input, contains vector y. Gets overwritten.
REAL*8	TOLZ	(Input) Tolerance for skipping transformations.

Method:

A single forward sweep of plane rotations. For example, Y(1) is eliminated first, changing the remainder of Y and the first row of R. Maintains nonsingularity of R. Underflow may occur occassionally without fatal consequence.

SUBROUTINE R1MOD(R,Y,NR,N,TOLZ,SIGMA,K)

Subroutines called: **Called by:**

R1ADD R1SUB COMDFP

Purpose:

To modify R such that (new R'R) = (old R'R) + SIGMA*yy'.

Parameters:

INTEGER	N	(Input) Current dimension of R.
REAL*8	Y(N)	On input, contains vector y. Gets overwritten.
REAL*8	TOLZ	(Input) Tolerance for skipping transformations.
REAL*8	SIGMA	(Input) A scalar, may be positive or negative.
INTEGER	K	(Output) Error flag, set to -K if modification failed.

Method:

1. Change y to abs(SIGMA)*y. Exit if norm(y) <= TOLZ.

2. Call R1ADD or R1SUB according to the sign of SIGMA.

SUBROUTINE R1PRCD(R,V,Y,NR,N,TOLZ,TOLD,JQ,KADD)

Subroutines called: **Called by:**

R1ADD RGITN

Purpose:

To modify R such that (new R'R) = (I + vw')*(old R'R)*(I + wv'), for the special case where v is general but w is the JQ-th unit vector, e(JQ).

Parameters:

INTEGER	N	(Input) Current dimension of R.
REAL*8	Y(N)	On input, contains vector y. Gets overwritten.
REAL*8	TOLZ	(Input) Tolerance for skipping transformations.
REAL*8	TOLD	(Input) Minimum allowable size of the JQ-th diagonal of R.
INTEGER	KADD	(Output) Error flag, set to -KADD if the JQ-th diagonal of R had to be raised to TOLD.

Method:

1. JQ=1 is a special case. Simply add R(1,1)*v' to the first row of R and exit.

2. Otherwise, compute y = Rw = Re(JQ). i.e. pull out the JQ-th column of R.

3. Perform a partial backward sweep of plane rotations (as in P1SUB) reducing the vector (y 0) to (0 S) by rotations in the planes (j,N+1), j=JQ,JQ-1, ..., 1. This creates a row r below R.

4. Compute y = r + Sv.

5. Give y to R1ADD to complete the modification.

6. Test the JQ-th diagonal of R, which could be zero or very small.

Postscript: This routine has since been modified to perform a partial backward sweep and then just a partial forward sweep (instead of the call to R1ADD). See Murtagh and Saunders (1978).

SUBROUTINE R1SUB(R,Y,NR,N,TOLZ,FAIL)

Subroutines called:

None

Called by:

R1MOD

Purpose:

To modify R such that (new R'R) = (old R'R) - yy'.

Parameters:

INTEGER	N	(Input) Current dimension of R.
REAL*8	Y(N)	On input, contains vector y. Gets overwritten.
REAL*8	TOLZ	(Input) Tolerance for skipping transformations.
LOGICAL	FAIL	(Output) Will be TRUE if y is such that (new R'R) would not be positive definite; in this case, R will be unaltered.

Method:

1. Solve R'p = y. Compute D = 1 - p'p.

2. Exit with FAIL = TRUE if D <= TOLZ. Otherwise set S = SQRT(D).

3. Perform backward sweep of plane rotations, reducing the vector (p S) to (0 1) by rotations in the planes (j,N+1), j=N,N-1, ..., 1.

SUBROUTINE SAVEB(M,N,NNO,NS,MS,HS,HB,X,XN,ISTATE)

Subroutines called:

None

Called by:

DRIVER

Purpose:

To output the current state of each variable in the form of a bit map (one character per variable), along with column numbers and values for superbasic variables. Output is to file INEWB.

Parameters:

INTEGER ISTATE(3) (Input) A 12-character string describing the current state of the solution (see subroutine STATE).

Method:

1. Output one card image containing Problem name, Iteration number, ISTATE, Phase, Objective value (or Sum of infeasibilities, if Phase=1).

2. Output one card image containing OBJ name, RHS name, RANGE name, BOUND name, No. of constraints (M), No. of variables (N), No. of superbasics (NS).

3. Output the state vector HS using FORMAT(80I1).

4. Output the column number HB(J) and value X(J) for each superbasic, one pair per card image, using FORMAT(I8, 1PE24.14). Terminate the list with column number 0.

5. Rewind file INEWB.

6. Print message 'BASIS MAP SAVED ON FILE <INEWB>'.

```
      SUBROUTINE SEARCH( NS,MS,NN,X,P,GRD,HB,XN,C,X1,G1,
     1   ETA,PNORM,STEPMX,ALPHA,F,UNBDD,IFAIL )
```

Subroutines called: DOT FUNGRD GETGRD NEWPTC

Called by: RGITN

Purpose:

To find an approximation to the minimum (with respect to ALPHA) of the function f(X + ALPHA*P) defined by FUNGRD. Driver for NEWPTC, derived directly from subroutine LNSRCH by Gill, Murray, Picken, Barber and Wright, NPL Algorithms Library, Ref. No. E4/16/F, DNAC, National Physical Laboratory, Teddington.

Parameters:

REAL*8	X(MS)	On input, contains initial values of the basic and superbasic variables. On output, contains X + ALPHA*P.
REAL*8	P(MS)	(Input) Contains the search direction for basics and superbasics.
REAL*8	X1(MS)	Work vector used to hold temporary X + ALPHA*P.
REAL*8	G1(NN)	Work vector to hold gradient for nonlinears at X1.
REAL*8	ETA	(Input) The linesearch accuracy tolerance.
REAL*8	PNORM	(Input) Norm of P.
REAL*8	STEPMX	(Input) Upper limit on ALPHA, defined by the bounds on X.
REAL*8	ALPHA	(Output) Returns step length.
REAL*8	F	(Output) Returns best objective value.
LOGICAL	UNBDD	(Output) Will be TRUE if ALPHA turns out to be very large (1.0E+8 currently used).
INTEGER	IFAIL	(Output) Will be 0 if search was successful, positive otherwise.

```
      SUBROUTINE SETPI( M,M2,NE,NP1,NF,MAXL,MAXU,NORM,TOLZ,
     1   HSPIKE,HA,HE,HPIVI,HPIVU,HPIVP,NL,NU,A,PI,Y,
     2   HL,DL,HU,DU,F )
```

Subroutines called:

BTRANL FTRANU

Called by:

ADDCOL DRIVER RGITN

Purpose:

To solve the system B(transpose)*PI = Y, given the vector Y.

Parameters:

INTEGER	NORM	(Input) 0 if norm(PI) is not required.
REAL*8	TOLZ	(Input) If norm(PI) is computed, any PI(i) smaller than TOLZ will be reset to zero.
REAL*8	PI(M)	(Output) The required solution. (The actual parameter may not always be the pricing vector for the simplex method.)
REAL*8	Y(M)	(Input) The given rhs vector. Not overwritten.

Method:

1. Set PI = Y and call FTRANU, BTRANL.

2. If NORM is nonzero, compute SUM = sum of PI(i)**2, resetting small components to zero and counting significant components. Set PINORM = square root of the average value of nonzero PI(i)**2. This is an attempt to dampen the effect of a few very large components.

```
      SUBROUTINE SETX( M,M2,N,NE,NP1,NF,NS,MS,MAXL,MAXU,
     1   HA,HB,HE,HS,HPIVL,HPIVU,HPIVF,NL,NU,A,BL,BU,X,Y,Y1,
     2   HL,DI,HU,DU,F,IRESET )
```

Subroutines called:

BTRANU FTRANL

Called by:

DRIVER INVERT

Purpose:

(Optionally) to compute values for the basic variables, using the current basis factorization and values for superbasics. Also, to perform a row check on X, to ensure that the general constraints Ax = b are satisfied to a certain precision.

Parameters:

REAL*8	X(MS)	Superbasic values are input in X(j), j=M+1, ..., MS. Values for basic variables are either input in X(j), j=1, ..., M, or are to be computed, depending on the value of IRESET.
REAL*8	Y(M)	Work vector.
REAL*8	Y1(M)	Work vector.
INTEGER	IRESET	(Input) 0 if basic X(j) are input, >0 if they are to be recomputed.

Method:

1. Set Y = 0.

2. For each nonbasic column A(K), accumulate -A(K)*B into Y, where B is the appropriate upper or lower bound, depending on the state of the associated variable. Can be skipped if B is zero (as it usually will be).

3. For each superbasic column A(K), accumulate -A(K)*X(J) into Y, where X(J) is the appropriate value of the superbasic variable.

4. (Recompute X) If IRESET>0, solve B*X = Y for the basic vars.

5. (Iterative refinement) If IRESET>0, set Y1 = Y - B*X and solve the system B*DX = Y1 for a correction to X. Store DX in Y1. Compute DXNORM = the largest component of DX, excluding corrections to logicals.

6. (Row check) Set Y = Y - B*X. At the same time, compute XNORM = the largest component of X, excluding logicals; also compute Y1(i) = norm of the i-th row of B.

7. If iterative refinement was done, an estimate of the condition number of B is DXNORM/(XNORM*EPS).

8. Compute the largest normalized row residual, RMAX = max Y(i)/Y1(i). Set the error flag IERR=6 if RMAX/XNORM is larger than the row-check tolerance, TOLROW.

```
      SUBROUTINE SOLN( NCALL,M,N,NE,NP1,NN,NNO,NS,MS,
     1    HA,HB,HE,HS,A,EI,BU,XN,C,X,PI,Y,HST,ISTATE,ONDISK )
```

Subroutines called:

SCLPRT UNPACK

Called by:

DRIVER

Purpose:

To load solution and state values for all variables into arrays Y and HST, and to output the solution in approximately MPS format.

Parameters:

INTEGER	NCALL	(Input) NCALL=1 if Y and HST are to be set up; NCALL=2 if solution is to be output.
REAL*8	Y(N)	Holds solution values as follows: Y(1), ..., Y(KRHS-1) Structurals Y(KRHS) RHS, always -1.0 Y(KRHS+1), ..., Y(N) Logicals (not rows)
INTEGER*2	HST(N)	Holds state values for structurals, RHS and logicals, as follows:

HST	State	Meaning
0	LL	Variable is at its lower limit
1	UL	Variable is at its upper limit
2	SBS	Superbasic
3	BS	Basic
4	EQ	Fixed
5	FR	Free (but nonbasic)
6	--	Below lower bound
7	++	Above upper bound

INTEGER	ISTATE(3)	(Input) Contains a string describing the state of the solution.
LOGICAL	ONDISK	(Input) TRUE if solution is to be output to file ISOLN.

Method:

1. (NCALL=1)
   (a) Copy the state vector HS into array HST.

   (b) Run through the nonbasics, setting Y to the appropriate bound values and changing LL or UL states to EQ or FR.

   (c) Run through the basics and superbasics, setting Y to the appropriate X values and changing BS or SBS states to -- or ++ for any infeasible values.

   (d) Exit if problem is linear or infeasible, or if user has not requested one last call to CALCFG. Otherwise, overwrite any nonlinear solution values with the values in XN, in case the user altered XN.

2. (NCALL>1)
   (a) Set LPR to the the output unit (file 6 or file ISOLN).

   (b) Expand the RHS vector into X.

   (c) Output solution heading (NAME, OBJ value, STATE, PHASE, etc.).

   (d) Output ROWS heading.

   (e) For each row, convert state and solution values for logicals into those for rows (Rows value = RHS value - Logical value); read the Row Name from file ISCR; call SOLPRT to output one line of the solution.

   (f) Output COLUMNS heading.

   (g) For each column J, compute the reduced gradient DJ by the same method as used in PRICE; include the gradient value C(J) if solution is feasible and variable J is nonlinear; change any negligible values to zero; read the Column Name from file ISCR; call SOLPRT to output one line of the solution.

SUBROUTINE SOLPST(J,ID,IA,X,C,B1,B2,D,K,BPLUS,ONDISK)

Subroutines called:

None

Called by:

SOLN

Purpose:

To output one line of the solution. Used for both the ROWS and the COLUMNS sections.

Parameters: (All are input parameters)

INTEGER	J	The variable number. J=1 for the first structural; J=N for the last row.
INTEGER	ID(2)	Name of the row or column.
INTEGER	IA	An index defining the state of the variable.
REAL*8	X	The value of the variable.
REAL*8	C	Slack activity (for rows) or objective gradient (for columns).
REAL	B1	Lower bound on row or column.
REAL	B2	Upper bound on row or column.
REAL*8	D	Pi value (for rows) or reduced gradient (for columns).
INTEGER	K	The column no. used by MPS/360. K=1 for the first row; K=N for the last structural (the RHS).
REAL	BPLUS	"Plus infinity" for bounds.
LOGICAL	ONDISK	TRUE if the solution is being output to file ISOLN, rather than the printer.

Method:

The only logic here is in determining which format statement to use, depending on the size of the two bound values, B1 and B2. If ONDISK is TRUE, output occurs directly to file ISOLN, using 1PE16.6 for all REAL quantities. Otherwise, one of four possible format statements is used. The word "NONE" is used for all infinite bounds; otherwise, REALS are output with format F16.5.

SUBROUTINE SPECS(L,ISPECS,ISCR,IVERSN,NSPEC)

Subroutines called: | Called by:

None | MINOS

Purpose:

To make a first pass at processing the SPECS file. Each card of the file is assumed to contain one of the following:

```
KEYWORD     IDENTIFIER
KEYWORD     INTEGER
KEYWORD     REAL
```

in relatively free format. The aim here is to extract such items as certain character strings and to output them to the scratch file in a fixed format that is suitable for re-reading using standard I/O routines. This is one way of processing data in free format, in a way that is machine-independent.

It is assumed that the first and last cards in the SPECS file contain the KEYWORDS "BEGIN" and "END" respectively.

Parameters:

INTEGER	L(80)	Workspace to hold a card image.
INTEGER	ISPECS	(Input) Unit no. for SPECS file.
INTEGER	ISCR	(Input) Unit no. for scratch file.
INTEGER	IVERSN	(Input) Coded version and date of MINOS.
INTEGER	NSPEC	(Output) Will be negative if an EOF occurred on file ISPECS before a "BEGIN" card was found. Otherwise, returns the no. of card images that contained useful information.

Method:

Since the scratch file is also used for row and column names, we are stuck with the same Logical Record Length, 8 characters. Four logical records will be output to the scratch file for each meaningful SPECS card. These will be the following:

```
KEYWORD          (3 characters + 5 blanks)
IDENTIFIER       (8A1, to be re-read as 2A4)
INTEGER          (8A1, to be re-read as I8)
REAL             (8A1, to be re-read as E8.1)
```

Depending on the KEYWORD, any of the IDENTIFIER, INTEGER or REAL may be 8 blanks.

1. Read cards from file ISPECS until one is found that contains "BEG" as the first non-blank characters. Exit with IPECS = -1 if End of File occurs first.

2. (BEGIN found) Decode IVERSN and output heading.

3. Process each card as follows.

   (a) Read and list the card image.

   (b) Scan to the first non-blank character. Skip card if completely blank, or if first character is "*", which indicates a comment.

   (c) Exit if the KEYWORD is "END".

   (d) Take as KEYWORD the 3 characters starting from, and including, the first non-blank character.

   (e) Scan to the next blank or "=".

   (f) Scan past all blanks or "="s.

   (g) The next character determines whether the second item on the card is an identifier or a number. If it is one of a list of characters contained in the local array LDIGIT, then extract up to 8 such characters and treat either as an INTEGER, or (if it contains a "." or "E" or "D") as a REAL. Otherwise, extract exactly 8 characters to be output as IDENTIFIER.

   (h) If an INTEGER or REAL has not yet been found, return to (e).

   (i) Output the 4 items KEYWORD, IDENTIFIER, INTEGER and REAL to the scratch file.

4. If a SPECS file was found, close and rewind file ISCR.

```
      SUBROUTINE SPECS2( NSPEC,MROWS,MCOLS,MELMS,
     1    M,M2,MN,MNN,N,NE,NF1,NF,NN,NN0,MAXS,NS )
```

Subroutines called: **Called by:**

None MINOS

Purpose:

To make a second pass at decoding information from the SPECS file. The scratch file now contains a sequence of KEYWORDS and VALUES in fixed format. It remains to determine which KEYWORDS they are.

Parameters:

INTEGER NSPEC (Input) The no. of KEYWORDS to be read from the scratch file.

The remaining parameters are output. They are the various INTEGER variables defining the dimensions of the problem about to be solved.

Method:

1. Set most parameters to their default values. For the others, it is important to know whether the user attempted to define values, or not. This is determined by setting the parameters to illegal values, e.g. 0 or -1, and then testing for such values after all KEYWORDS have been processed.

2. For each KEYWORD, read the variables KEY, ID, NUM, DNUM using format(A3/, 2A4/, I8/, E8.1).

3. Determine which keyword is contained in KEY by searching a list of legal keywords. The use of ID, NUM and DNUM depends on KEY. In most cases, only one of ID, NUM, or DNUM is used.

4. When all keywords have been processed, assign default values to those that were not specified.

5. Print a summary of the parameters about to be used, and exit.

```
      SUBROUTINE SQUEEZ( M,N,NB,NN,NNO,NS,MS,
     1    HB,HS,BL,BU,R,X,XN,G,GRD,TOLX,PINORM,RGNORM,DELETR )
```

Subroutines called:

DELCOL

Called by:

RGITN

Purpose:

To remove any superbasics that are very close to a bound and do not have a significant reduced gradient driving them away from that bound. (Will be called only after constrained steps in RGITN.)

Parameters:

REAL*8	TOLX	(Input) The measure of closeness to a bound.
REAL*8	PINORM	(Input) Norm of PI, to scale the tolerance on reduced-gradient components.
LOGICAL	DELETR	(Input) To be passed to DELCOL.

Method:

Superbasics are split up depending on whether their reduced gradient is small or not. Coding could be simplified if the purpose stated above is followed directly.

If a variable is to be squeezed, its state, HS(K), is made nonbasic at the appropriate bound. Arrays G and GRD are compacted, and DELCOL does the same for HE, X and R.

Throughout, superbasics are processed in reverse order to reduce work slightly if more than one gets squeezed.

SUBROUTINE STATE(K,ISTATE)

Subroutines called: | **Called by:**

None | DRIVER

Purpose:

To load appropriate character strings into the array ISTATE, for output with a saved bit map or the final solution.

Parameters:

INTEGER	K	(Input) Specifies the current state of the problem.
INTEGER	ISTATE(3)	(Output) Will contain one of the following strings, 4 characters per word:

K=0 PROCEEDING
1 OPTIMAL SOLN
2 INFEASIBLE
3 UNBOUNDED
4 EXCESS ITNS
5 ERROR CONDN

Method:

Trivial.

SUBROUTINE TRNSVI(N,N1,NZ,IP,IRN,IPIVRW,PC,RV,PREORD,ACP,OUT,NREJ)

Subroutines called: **Called by:**

None F4

Purpose:

Given the row numbers of the nonzeros in each column of a square sparse matrix A, this routine finds a column permutation P such that AP has a nonzero diagonal. If this is not possible, a permutation is found such that the no. of consecutive nonzero diagonals in AP is as large as possible.

Parameters:

INTEGER	N	(Input)	Dimension of the matrix.
INTEGER	N1	(Input)	N+1.
INTEGER	NZ	(Input)	No. of nonzeros in the matrix.
INTEGER*2	IP(N1)	(Input)	Points to the first nonzero in each column.
INTEGER*2	IRN(NZ)	(Input)	List of row numbers for nonzeros.
INTEGER*2	IPIVRW(N)	(Output)	Will contain the required permutation. Considering the matrix as an N*N array of elements A(I,J), the elements A(I,IPIVRW(I)) will be nonzero (except if IPIVRW(I) = 0; see NREJ below).
INTEGER*2	PC(N)	Work vector.	
INTEGER*2	RV(N)	Work vector.	
INTEGER*2	PREORD(N)	Work vector.	
INTEGER*2	ACP(N)	Work vector.	
INTEGER*2	OUT(N)	Work vector.	
INTEGER	NREJ	(Output)	Will be nonzero if the matrix is structurally singular (i.e. singular for all possible values of the nonzeros). Certain elements of IPIVRW will then be zero and it is possible therefore to determine which columns of the matrix must be replaced to obtain a nonsingular matrix.

Method:

This routine is derived directly from Harwell subroutine MC16B by I.S. Duff (dated Dec 1975), which may since have been replaced by MC21A.

```
      SUBROUTINE UNPACK( M,NE,NP1,HA,HE,A,Y,JQ )
```

Subroutines called:

None

Called by:

ADDCOL DRIVER FACTOR LPITN
RGITN SOLN

Purpose:

To expand the packed column A(JQ) into the vector Y.

Method:

The parameters and method are obvious from the code. Note that negative row idices HA(J) may occur in special applications (e.g., MINOS/GRG), but only their absolute value is relevant here.

```
      SUBROUTINE UNPACK( M,NE,NP1,HA,HE,A,Y,JQ )
      INTEGER*2  HA(NE),HE(NP1)
      REAL       A(NE)
      REAL*8     Y(M)
C
C     UNPACK THE JQ-TH COLUMN OF  A
C
      DO 10 I=1,M
   10 Y(I) = 0.0
C
      J1 = HE(JQ)+1
      J2 = HE(JQ+1)
C
      DO 20 I=J1,J2
         IR = HA(I)
         IR = IABS(IR)
         Y(IR) = A(I)
   20 CONTINUE
C
      RETURN
C     END OF UNPACK
      END
```

11. PERFORMANCE

The proving ground for MINOS has been the U.S. energy-economic models of Alan Manne. These are the models ETA (Manne, 1976) and ETA-MACRO (Manne, 1977), of which several hundred cases have been run during 1975-77.

In ETA-MACRO, the constraint matrix is approximately 430 x 750 with 2800 nonzero coefficients. The objective function involves 80 variables nonlinearly. It is of the form

$$f(K,E,N,I,EC) = \sum_{t=1}^{16} \delta_t \log C_t$$

where

$$C_t = y_t + (ac_t + bd_t)^{1/\rho} - I_t - EC_t ,$$

$$c_t = (K_t - k_t)^{\alpha\rho} (L_t - l_t)^{(1-\alpha)\rho} ,$$

$$d_t = (E_t - e_t)^{\beta\rho} (N_t - n_t)^{(1-\beta)\rho} ,$$

and the unknowns are the five sets of variables K_t, E_t, N_t, I_t, EC_t. Clearly $\log C_t$ is a non-separable function of these variables.

On an IBM System 370 the problem can be run in a 384K region. This core requirement is made up approximately as follows:

Program code	184K
REAL*8 Z(20000)	160K
I/O buffers	40K
	384K bytes.

From a cold start, using CRASH OPTION 1 with no special initialization of the nonlinear variables, the run time on an IBM 370/168 is typically about 100 seconds for 1100 iterations. (The first 500 iterations are primal simplex iterations obtaining an initial feasible solution.) Restarts with different bound sets require a few hundred further iterations.

Several factors contribute to the efficiency of these runs. The constraints are very sparse, and the LU factorization of each basis typically has 4 bumps and only 8 spikes. The number of superbasic variables, and hence the dimension of the reduced Hessian approximation, remains at around

30, which is well below the theoretical upper bound of 80. Finally, although a relatively accurate search is used (ETASCH = 0.01), the line-search procedure of Gill et al (1976) in this case averages only 2 function and gradient evaluations per iteration.

12. ADDENDA TO MINOS USER'S GUIDE

The User's Guide accurately describes MINOS Version 3.1. Some of the changes incorporated in MINOS Version 3.3 will be apparent to the user, as follows:

1. Two additional parameters may be specified in the SPECS file, by cards of the form

   DJ TOLERANCE 1.0E-5
   FEASIBILITY TOL 1.0E-4

   These reset the variables TOLDJ3 and TOLX respectively (see section 5).

2. The default value for the feasibility tolerance TOLX is now $\max(\sqrt{EPS}, 10^{-5})$.

3. Subroutines ALIGN and SQUEEZ have been deleted. The keyword ALIGNMENT TOLERANCE no longer has the effect described on p. 40 of the User's Guide.

4. Variables that are specified to be superbasic in
   - an INITIAL bounds set in the MPS file
   - an OLD BASIS FILE

   will be initialized at the specified values regardless of feasibility. (Exceptions: "infinite" values are replaced by zero; if the SUPERBASICS LIMIT has been reached, variables will be made nonbasic at the nearest bound.) This facilitates initialization at intentionally infeasible points, and exact reconstruction of solutions from previous runs. The following parts of the User's Guide should be altered accordingly:

   p. 29, 8(a).
   p. 62, 8.

5. MPS FILE is a synonym for INPUT FILE.

REFERENCES

R.H. Bartels (1971), "A Stabilization of the Simplex Method", Numerische Mathematik 16, 414-434.

R.H. Bartels and G.H. Golub (1969), "The Simplex Method of Linear Programming Using LU Decomposition", Comm. ACM 12, 266-268.

R.P. Brent (1973), "Reducing the Retrieval Time of Scatter Storage Techniques", Comm. ACM 16, 105-109.

I.S. Duff (1975), An algorithm for obtaining a maximum transversal for a sparse matrix, Subroutine MC21A in the Harwell Subroutine Library, Harwell, England.

I.S. Duff and J.K. Reid (1976), "An implementation of Tarjan's algorithm for the block triangularization of a matrix", AERE Report CSS 29, Harwell, England.

J.J.H. Forrest and J.A. Tomlin (1972), "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method", Mathematical Programming 2, 263-278.

P.E. Gill, W. Murray, S.M. Picken, H.M. Barber and M.H. Wright (1976), Subroutines LNSRCH and NEWPTC, Ref. No. E4/16/0/Fortran/02/76, NPL Algorithms Library, DNAC, National Physical Laboratory, Teddington. (Crown Copyright Reserved.)

T.S. Hedges (1975), An assembly language main program to acquire core at run-time on IBM Systems 360 and 370. (Unpublished. Documented in-line.)

E. Hellerman and D.C. Rarick (1971), "Reinversion with the Preassigned Pivot Procedure", Mathematical Programming 1, 195-216.

A. Jain, L.S. Lasdon and M.A. Saunders (1976), "An In-core Nonlinear Mathematical Programming System for Large Sparse Nonlinear Programs", presented at ORSA/TIMS joint national meeting, Miami, Florida.

A.S. Manne (1976), "ETA: A Model for Energy Technology Assessment", Bell Journal of Economics, Autumn 1976, 381-406.

A.S. Manne (1977), ETA-MACRO: A Model of Energy-Economy Interactions", in C.J. Hitch (ed.), Modeling Energy-Economy Interactions, Resources for the Future, Washington, D.C., 1977. Also appears as ch. 9 in R. Pindyck (ed.), Advances in the Economics of Energy and Resources, vol. 2: The Production and Pricing of Energy Resources, JAI Press, Greenwich, Connecticut, 1978.

B.A. Murtagh and M.A. Saunders (1977), MINOS User's Guide, Report SOL 77-9, Department of Operations Research, Stanford University, Stanford, California.

B.A. Murtagh and M.A. Saunders (1978), "Large-scale linearly constrained optimization", Mathematical Programming 0, 000-000. (Revised version of the following report by the same authors: "Nonlinear Programming for Large, Sparse Systems", Report SOL 76-15, Department of Operations Research, Stanford University, Stanford, California.)

M.A. Saunders (1976), "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating", in: J.R. Bunch and D.J. Rose, eds., Sparse Matrix Computations (Academic Press, New York and London), 213-226.

M.A. Saunders (1977), "Use of MINOS on the Burroughs B6700", SCIN 59, Applied Mathematics Division, DSIR, Wellington.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 77-31	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MINOS System Manual		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M.A. Saunders		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0865 DAAG-29-74-C-0034
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research--SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS Operations Research Program Office of Naval Research Arlington, VA 22217 Mathematics Division U.S. Army Research Office Box CM, Duke Station Durham, N.C. 27706		12. REPORT DATE December 1977 13. NUMBER OF PAGES 136 15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for any purposes of the United States Government.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Linear Programming	Mathematical Software
Nonlinear Programming	Large-Scale Optimization
MINOS	Linearly Constrained Optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

MINOS is a Fortran system for solving large-scale linearly constrained optimization problems. The System Manual gives an overview of the system, the programming conventions used, data structures, tolerances, and error conditions. Details are given of a practical implementation of the method of Bartels and Golub for maintaining a sparse LU factorization. The reduced-gradient approach for handling a nonlinear objective function has been described elsewhere by Murtagh and Saunders; further implementation details are included here. The System Manual should facilitate interfacing of MINOS with other optimization software.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)