Users' Guide for the Unsymmetric-pattern MultiFrontal Package (UMFPACK)

Timothy A. Davis

June 1993

Technical Report TR-93-020
Computer and Information Sciences Department
University of Florida
Gainesville, FL, 32611 USA

1 Introduction

The **Unsymmetric-Pattern MultiFrontal Package (UMFPACK)** is a set of subroutines designed to solve linear systems of the form $Ax = b$, where A is an n-by-n general unsymmetric sparse matrix, and x and b are n-by-1 vectors. It uses LU factorization, and performs pivoting for numerical purposes and to maintain sparsity.

UMFPACK is based on the *unsymmetric-pattern multifrontal method* [3]. The method relies on dense matrix kernels [4] to factorize *frontal matrices*, which are dense submatrices of the sparse matrix being factorized. In contrast to the classical multifrontal method [2, 8], frontal matrices are rectangular instead of square, and the assembly tree is replaced with a directed acyclic graph. As in the classical multifrontal method, advantage is taken of repetitive structure in the matrix by amalgamating nodes in the directed acyclic graph, giving it high performance on parallel-vector supercomputers.

UMFPACK is written in ANSI Fortran-77, with compiler directives for the Cray Fortran compiler (which are simply seen as comments by other compilers). Both double-precision and single-precision (`REAL`) versions are included. UMFPACK contains no common blocks.

There are eleven user-callable subroutines in each version of UMFPACK (single-precision and double-precision). All double-precision subroutines start with the three letters `UMD`, and all single-precision subroutines start with `UMS`. The `UM_` prefix will be used to describe both versions. Arguments described as `REAL/DOUBLE PRECISION` are `REAL` in the single-precision version, and `DOUBLE PRECISION` in the double-precision version. You do not need to call the last five subroutines in the following list unless you wish to dynamically allocate and deallocate memory from the workspace.

`UM_FAC` Computes the pivot ordering and both symbolic and numerical LU factors of an n-by-n general unsymmetric sparse matrix.

`UM_RFA` Computes the numerical LU factors of a matrix using the pivot ordering and symbolic LU factors computed by `UM_FAC`.

UM_SOL Solves a system using the LU factors computed by UM_FAC or UM_RFA.

UM_SMV Multiplies a sparse matrix A times a dense vector x.

UM_STA Computes summary statistics.

UM_INI Initializes allocatable memory. This subroutine must be called before calling any other UM_ subroutine.

UM_MCO Checks the validity of allocatable memory.

UM_MRK Marks the current state of allocatable memory.

UM_RST Restores allocatable memory to a previously marked state.

UM_GET Allocates memory from allocatable memory.

UM_FRE Deallocates memory from allocatable memory.

2 Arguments common to most UM_ subroutines

The user-callable subroutines in UMFPACK share a set of common arguments. Not all subroutines use the entire set. The arguments divide into five classes: workspace, input matrix (A), scale factors, LU factors, and diagnostics.

2.1 Workspace

The workspace consists of three arrays:

```
INTEGER IMEM (*), PMEM (7,2)
REAL/DOUBLE PRECISION XMEM (*)
```

UMFPACK makes use of an INTEGER workspace (IMEM) and a REAL/DOUBLE PRECISION workspace (XMEM). Both are one-dimensional arrays. The LU factors and all temporary data structures are allocated from these two workspaces. A single INTEGER array, PMEM(7,2), describes what portions of the IMEM and XMEM arrays are in use. The PMEM (*,1) column refers to IMEM, and the PMEM (*,2) column refers to XMEM. The name MEM refers to either IMEM or XMEM, as appropriate (depending on which column of PMEM is referred to).

PMEM (1,*) (head) First index of free memory.

PMEM (2,*) (tail) First index of allocated memory at the tail of MEM.

PMEM (3,*) (inithd) Initial value of head.

PMEM (4,*) (inittl) Initial value of tail.

PMEM (5,*) Maximum total usage of allocatable memory since the last call to UM_INI. The total usage is ((head-inithd)+(inittl-tail)), which includes external fragmentation.

PMEM (6,*) (actual) The actual current usage of memory, excluding external fragmentation. For example, if a data structure embedded in the middle of MEM (inithd..head-1) is freed, its space is not reclaimed. Thus, the total usage does not change. However, the actual usage is decremented by the size of the freed data structure.

PMEM (7,*) Maximum value of actual since the last call to UM_INI.

You should not modify PMEM directly. They are initialized by the UM_INI subroutine, and modified by the other UM_ subroutines. The first four entries describe what portions of memory have been allocated and which are free (the remaining three entries are for statistics only):

MEM (inithd..inittl-1) This is the only portion of MEM that will be used. The inithd and inittl indices are set by UM_INI, which also initializes the head and tail indices to inithd and inittl, respectively.

MEM (inithd..head-1) This portion of MEM holds data structures allocated from the head of MEM. It may also include external fragmentation.

MEM (head..tail-1) This portion of MEM is unused allocatable space.

MEM (tail..inittl-1) This portion of MEM holds data structures allocated from the tail of MEM. It may also include external fragmentation.

Most of the subroutines modify portions of XMEM, IMEM, and PMEM. Users may allocate and deallocate space from XMEM and IMEM using the UM_GET, UM_FRE, UM_MRK, and UM_RST subroutines.

2.2 Input matrix

The input matrix, A, consists of two arrays and nine scalars:

```
REAL/DOUBLE PRECISION AVALUE (NZ), DTOL
INTEGER AINDEX (2,NZ), N, NZ, NDUPL, NDROP, NINVLD
LOGICAL TRANS, DUPL, DROP
```

The matrix A is an N-by-N matrix in a simple triplet format. The entries are held in AINDEX (1..2, 1..NZ) and AVALUE (1..NZ). The order, N, must be greater than zero, and NZ must be greater than or equal to zero.

The entries are stored as follows. If TRANS is .FALSE., then the K-th entry in A (a_{ij}) has the value AVALUE (K), and is located in row i and column j, where i is AINDEX (1,K) and j is AINDEX (2,K). If TRANS is .TRUE., then the matrix is transposed: i is AINDEX (2,K) and j is AINDEX (1,K).

Invalid entries are those with row or column indices outside the range 1 to N. If the input matrix contains invalid entries, the matrix is not factorized. The output argument NINVLD is set to the number of invalid entries. The invalid entries are printed on the diagnostic output if PRLEV is two or more.

The subroutines tolerate duplicate entries, but **ONLY** if DUPL is set to .TRUE. (where applicable). The K-th entry of A is a duplicate if all four of the following conditions hold for some value of L:

1. AINDEX(1,L) .EQ. AINDEX(1,K)

2. AINDEX(2,L) .EQ. AINDEX(2,K)

3. L .LT. K

4. the L-th entry is not a duplicate.

Duplicate entries are removed, and their values are added to their non-duplicate representative (the L-th entry is the representative of the K-th entry in the conditions listed above). The output argument NDUPL is set to the number of duplicates found. If PRLEV is two or more, then a list of the duplicates is printed on the diagnostic output (see the IO argument). If DUPL is .FALSE., no error is reported if duplicate entries are given, but erroneous results may be returned. Set DUPL to .FALSE. only if you are **SURE** your input contains no duplicates. The performance is slightly higher if DUPL is .FALSE..

If DROP is .TRUE., then original entries in A (after removal of duplicates, and scaling, if applicable) that have absolute value less than or equal to DTOL are considered to be numerically zero. They are removed from A before factorization. The output argument NDROP is set to the number of non-duplicate entries dropped.

If desired, AINDEX and AVALUE (and any other user array) can be stored in IMEM and XMEM, respectively. The SDEMO and DDEMO programs do this. See the UM_INI, UM_GET, UM_FRE, UM_MRK, and UM_RST subroutines for more details.

Note that no subroutine modifies the input matrix, A (specifically, the AVALUE, AINDEX, N, NZ, TRANS, DUPL, DROP, and DTOL arguments), even if entries are dropped, scaled, and duplicates are removed. The subroutines make a copy of A into an internal format before performing these operations. The copy is placed in memory allocated from IMEM and XMEM. Only the NDUPL, NDROP, and NINVLD output arguments are modified.

2.3 Scale factors

The input matrix can be scaled before factorization by providing two one-dimensional arrays:

```
REAL/DOUBLE PRECISION ASR (N), ASC (N)
LOGICAL SCALE
```

No user-callable subroutine is provided in UMFPACK for computing the scale factors (although the DDEMO and SDEMO programs compute them). The subroutines UM_FAC, UM_SOL, and UM_RFA use the scale factors (and do not modify them). The scaled matrix is $D_r A D_c$, where D_r and D_c are diagonal N-by-N matrices, such that the I-th diagonal of D_r is ASR (I), and the J-th diagonal of D_c is ASC (J). The scale factors are used only if the SCALE flag is .TRUE.. The same system ($Ax = b$ or $A^T x = b$) is still solved, regardless of the scale factors.

2.4 LU factors

The following array describes the location of the factors in allocatable memory:

```
INTEGER LU (2,2)
```

Table 1: Contents of IMEM (LU(1,1)..LU(2,1)-1)

size	contents
1	N
1	BLOCKS
1	NZOFF
1	NZDIA
1	NFRONT
1	LUPP, location of LU pattern pointers
N	permutation array (P)
N	permutation array (Q)
2*BLOCKS+2	block triangularization information
N+1	pointers for off-diagonal blocks (if BLOCKS > 1)
NZOFF	pattern of off-diagonal blocks (if BLOCKS > 1)
(remainder)	LU pattern and assembly directed acyclic graph
NFRONT	LU pattern pointers (one for each frontal matrix)

Table 2: Contents of XMEM (LU(1,2)..LU(2,2)-1)

size	contents
NZOFF	numerical values of off-diagonal blocks (if BLOCKS > 1)
(remainder)	numerical values of LU factors

The LU factors are stored at the head of allocatable memory in XMEM and IMEM. The INTEGER part of the LU factors (including pattern, permutation vectors, and various scalar information) is stored in IMEM (LU(1,1)..LU(2,1)-1). The REAL/DOUBLE PRECISION part of the LU factors is stored in XMEM (LU(1,2)..LU(2,2)-1). The subroutines UM_FAC and UM_RFA place these at the head of available memory. That is, LU (1,1) and LU (1,2) will be equal to the values of PMEM (1,1) and PMEM (1,2), respectively, when UM_FAC was called.

The LU factors are completely relocatable in XMEM and IMEM. To move the factors, simply copy the contents to another portion of XMEM and IMEM, and update the LU array to reflect the new location. All indices stored in IMEM are relative to the LU (1,1) and LU (1,2) values, and need not be modified when the LU factors are relocated.

The LU array is modified by UM_FAC and UM_RFA, and used (but not modified) by UM_SOL and UM_STA.

The factorization is $PD_rAD_cQ = LU$, where P and Q are row and column permutations due to permutations to block-upper-triangular form and pivoting during factorization, D_r is the row scaling, D_c is the column scaling, L is lower triangular with a unit diagonal, and U is upper triangular.

The structure of the LU factors in IMEM and XMEM is shown in Table 1 and Table 2, respectively. See the UM_STA subroutine for more details (Section 3.5).

Table 3: Output file format for matrices

contents	description
`N N`	line 1. The matrix is `N`-by-`N`.
`I J X`	one line per entry: row `I`, column `J`, value `X`
`0 0 0`	last line

The permutations are stored as two one-dimensional arrays, `PERMR` and `PERMC`. If `ROW` and `COL` refer to a row or column in the original matrix A, then `ABS (PERMR (ROW))` gives the row index of where `ROW` appears in PA, and `ABS (PERMC (COL))` gives the column index of where `COL` appears in AQ. If a row (or column) was not selected as a pivot row (or column) due to numerical problems, then `PERMR (ROW)` (or `PERMC (COL)`) is negative.

2.5 Diagnostics

Error reporting is controlled by the following arguments:

```
INTEGER PRLEV, IO, ERROR
```

The input argument `PRLEV` controls the printing of error messages on the Fortran unit used for diagnostic output. The unit used for diagnostics is given by the `IO` input argument. It is assumed to be already opened. If `PRLEV` is zero, then no diagnostics are printed (the diagnostic output unit is not used). If `PRLEV` is nonzero, then terse error message(s) are printed if an error occurs. If `PRLEV` is two or more, then more diagnostics are printed. In particular, duplicate and invalid entries in A are printed.

If `PRLEV` is less than zero, then the patterns of several matrices are written to files using `ABS(PRLEV)` as the Fortran output unit. The original matrix A is written to the file `IJORIG`, the permutation of A to block-upper-triangular form is written to the file `IJBLOK`, and the LU factors (excluding the diagonal of L) are written to the file `IJLU`. The format of the files is compatible with the *Sparse Matrix Manipulation System (SMMS)* [1], and is shown in Table 3.

The output argument, `ERROR`, is zero if no error occurred, and nonzero otherwise.

3 Description of each user-callable subroutine

Each user-callable subroutine is described in this section. The argument lists of each subroutine contain arguments in one of four classifications:

Input The argument is read, but not written to.

Modified The argument is both read and written.

Output The argument is only written. The previous value of the argument is not used.

Unused The argument is not used by the subroutine. It is reserved for possible future releases.

3.1 UM_FAC

The UM_FAC subroutine computes an LU factorization of a general unsymmetric sparse matrix A. The matrix can be optionally pre-ordered into a block-upper-triangular form, using two Harwell MA28 subroutines [7]. Pivoting within each diagonal block is performed during factorization to maintain sparsity and numerical stability. The input matrix can be optionally pre-scaled before factorization.

3.1.1 Argument list

```
SUBROUTINE UM_FAC (XMEM, IMEM, PMEM, AVALUE, AINDEX, N, NZ, TRANS, DUPL, DROP,
        DTOL, NDUPL, NDROP, NINVLD, ASR, ASC, SCALE, LU, PRLEV, IO, ERROR,
        BLOCK, RELPT, ABSPT, GRO, LOSRCH, HISRCH, SYMSRC, NB, EXTRA, NPIV,
        PMIN, IGAR, XGAR)
INTEGER IMEM (*), PMEM (7,2), AINDEX (2,NZ), N, NZ, NDUPL, NDROP, NINVLD,
        LU (2,2), PRLEV, IO, ERROR, LOSRCH, HISRCH, NB, EXTRA, NPIV, IGAR, XGAR
LOGICAL TRANS, DUPL, DROP, SCALE, BLOCK, SYMSRC
REAL/DOUBLE PRECISION XMEM (*), AVALUE (NZ), DTOL, ASR (N), ASC (N), RELPT,
        ABSPT, GRO, PMIN
```

3.1.2 Input

AVALUE, AINDEX, N, NZ, TRANS, DUPL, DROP, DTOL Input matrix to factorize.

ASR, ASC, SCALE Scale factors.

PRLEV, IO Diagnostic printing.

BLOCK If .TRUE., then A is pre-permuted to block-upper-triangular form. Uses the Harwell MC13E and MC21B subroutines.

RELPT Relative numerical pivot tolerance. If zero, then no relative numerical test is made. If greater than zero, a pivot $a_{ij}^{[k]}$ must satisfy the threshold partial pivoting test:

$$|a_{ij}^{[k]}| \geq \texttt{RELPT} \cdot \max_{k \leq s \leq n} |a_{sj}^{[k]}|$$

where the notation $a_{ij}^{[k]}$ refers to an entry in the partially factorized matrix just prior to step k. The RELPT argument performs the same function as the U argument in MA28. Range: 0 to 1.0, typical value: 0.001 to 0.1.

ABSPT Absolute numerical pivot tolerance. A pivot $a_{ij}^{[k]}$ must satisfy

$$|a_{ij}^{[k]}| > \texttt{ABSPT}$$

Entries with absolute values less than or equal to ABSPT are essentially considered to be numerically zero, although no entries are dropped during factorization. Setting DROP to .TRUE. only drops entries *before* factorization starts. Range: ≥ 0, typical value: 0 to DTOL.

GRO How much a frontal matrix can grow due to amalgamation. A value of 1.0 means that no fill-in due to amalgamation will occur. Some amalgamation is necessary for efficient use of the Level-3 BLAS. A value of N will result in unlimited growth, and the matrix A will be treated as a single, N-by-N dense matrix. Range: 1.0 to N, typical value: 2.0 to 3.0.

LOSRCH The UM_FAC subroutine does not maintain the true degree of each row and column of the matrix being factorized (the degree is simply the number of entries in the row or column). Instead, it keeps track of upper and lower bounds that are easier to compute (see [3] for details). The LOSRCH argument is the number of columns of lowest lower bound degree to examine during pivot search. If greater than zero, then 3·N extra temporary INTEGER space is allocated from IMEM. Range: 0 to N, typical value: 0.

HISRCH The number of columns of lowest upper bound degree to examine during pivot search. The LOSRCH and HISRCH arguments perform a similar function as the NSRCH argument in MA28, except that setting HISRCH and/or LOSRCH to N is not efficient in UM_FAC. Range: 0 to N, typical value: 4.

SYMSRC If .TRUE., then pivots on the diagonal of A are preferred over pivots off the diagonal. If A is pre-permuted to block-upper-triangular form, then the diagonal of the permuted matrix is preferred. If .FALSE., then no preference is made. Setting SYMSRC to .TRUE. is useful for matrices that are diagonally dominant, since fill-in is sometimes less if symmetry is preserved. Typical value: .FALSE.

NB The block size for the numerical factorization of the dense frontal matrices. It controls the trade-off between Level-2 and Level-3 BLAS. A value of one will (effectively) result in no Level-3 BLAS being used during the factorization of frontal matrix pivot blocks (actually, the Level-3 GEMM subroutine is still used, but it could be replaced with a call to the Level-2 GER subroutine if NB is one). The Level-3 GEMM subroutine is used to update the contribution blocks, regardless of the value of NB. See [3] for details. The best value of NB depends on the computer being used. Range: 1 to N, typical value: 16 to 64.

EXTRA How much extra space to allocate for additional elbow-room in the tuple lists. A tuple list is a list of the frontal matrices that affect a single row or column. The tuple lists grow and shrink during factorization (see [3] for details). Up to EXTRA·2·N memory in IMEM is used, although this upper bound is rarely reached in practice. A value of zero will not be very efficient. Range: 0 to N, typical value: 5.

3.1.3 Modified

PMEM Status of allocatable memory.

3.1.4 Output

XMEM, IMEM Used for temporary workspace, and to store the LU factors on exit.

NDUPL Number of duplicate entries in A.

NDROP Number of (non-duplicate) entries dropped from A.

NINVLD Number of invalid entries in A.

LU Location of the LU factors in XMEM and IMEM.

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

101 UM_FAC: matrix order <= 0!, N
N must be greater than zero.

102 UM_FAC: number of nonzeros < 0!, NZ
NZ must be greater than or equal to zero.

103 UM_FAC: invalid entries in input matrix!
Check input matrix for entries with row or column indices outside the range 1 to N. Get a listing of them by setting PRLEV to two.

104 UM_FAC: memory inconsistency!
Call UM_INI before calling any other UM_ subroutine.

105 UM_FAC: out of memory! (IMEM)
Increase the size of IMEM, decrease memory requirements, or decrease fill-in. Memory requirements can be reduced by decreasing EXTRA and/or setting LOSRCH to zero. Fill-in can usually be decreased by dropping small entries, decreasing GRO, increasing LOSRCH, increasing HISRCH, decreasing RELPT, and/or decreasing ABSPT. Fill-in can sometimes be decreased by setting BLOCK to .TRUE., if the matrix is reducible to a block-upper-triangular form. Small entries can be dropped by setting DROP to .TRUE. and increasing DTOL. Sometimes factorizing A^T can lead to less fill-in than factorizing A (setting TRANS to .TRUE. in both UM_FAC and UM_SOL will solve $Ax = b$; try this in the DEMO program in Section 5). None of these suggestions for reducing fill-in are guaranteed to work, since the pivot search is a heuristic.

106 UM_FAC: out of memory! (XMEM)
Increase the size of XMEM, or decrease fill-in (see the comments for error 105).

108 UM_FAC: MAXINT too small! (UM_max), MAXINT
Change MAXINT in the UM_MAX subroutine.

NPIV Number of numerically acceptable pivots found during factorization. If NPIV is equal to N, then the matrix U contains a zero-free diagonal (all entries on the diagonal have absolute value greater than ABSPT, to be precise).

PMIN Minimum absolute value on the diagonal of U, excluding entries with absolute value less than ABSPT. If NPIV is less than N, then PMIN is the minimum absolute value of the acceptable pivots.

IGAR Number of garbage collections performed on IMEM. Garbage collections are performed when the available memory is exhausted. Memory is compacted to remove all external fragmentation. If IGAR is excessively high, it can degrade performance. Try increasing the size of IMEM if that occurs (or try reducing fill-in or memory requirements using the suggestions listed under error 105 above).

XGAR Number of garbage collections performed on XMEM (if XGAR is too high, it can be reduced by using the suggestions listed under error 106 above).

3.2 UM_RFA

The UM_RFA subroutine factorizes a matrix using the same pattern and pivot ordering as another matrix previously factorized by UM_FAC. No variations are made in the pivot order computed by UM_FAC. The entries in AINDEX and AVALUE must be within the pattern of the LU factors. That is, if $(L+U)_{ij}$ is nonzero, then the entry $(PAQ)_{ij}$ can be present in AINDEX and AVALUE (where P and Q are the permutations determined by UM_FAC). The argument NINVLD is set to the number of entries that violate this condition.

3.2.1 Argument list

```
SUBROUTINE UM_RFA (XMEM, IMEM, PMEM, AVALUE, AINDEX, N, NZ, TRANS, DUPL, DROP,
        DTOL, NDUPL, NDROP, NINVLD, ASR, ASC, SCALE, LU, PRLEV, IO, ERROR,
        DEALLO, ABSPT, NB, NPIV, PMIN, XGAR)
INTEGER IMEM (*), PMEM (7,2), AINDEX (2,NZ), N, NZ, NDUPL, NDROP, NINVLD,
        LU (2,2), PRLEV, IO, ERROR, NB, NPIV, XGAR
LOGICAL TRANS, DUPL, DROP, SCALE, DEALLO
REAL/DOUBLE PRECISION XMEM (*), AVALUE (NZ), ASR (N), ASC (N), DTOL, ABSPT,
        PMIN
```

3.2.2 Input

AVALUE, AINDEX, N, NZ, TRANS, DUPL, DROP, DTOL Input matrix to factorize.

ASR, ASC, SCALE Scale factors.

PRLEV, IO Diagnostic printing.

DEALLO If .TRUE., then UM_RFA deallocates the old numerical LU factors in XMEM before computing new factors. If .FALSE., then the old LU factors are not deallocated. Memory will not be reclaimed unless the LU factors are adjacent to the free memory space in XMEM (which is where UM_FAC places them). Typical value: .TRUE..

ABSPT, NB Same usage as UM_FAC.

3.2.3 Modified

XMEM, IMEM, PMEM Holds the LU factors on input, as computed by UM_FAC (or a previous call to UM_RFA). Used as workspace, and on output holds the new LU factorization.

LU Holds the location of the old LU factors on input, and the location of the new LU factors on output (in XMEM and IMEM).

3.2.4 Output

NDUPL, NDROP, NINVLD, NPIV, PMIN, XGAR Same usage as UM_FAC.

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

201 `UM_RFA: different matrix order!`
`N` must be the same as given to `UM_FAC`.

202 `UM_RFA: number of nonzeros < 0!, NZ`
`NZ` must be greater than or equal to zero.

203 `UM_RFA: invalid entries in input matrix!`
Check input matrix for entries with row or column indices outside the range 1 to `N`. Any entry not within the LU pattern of the matrix factorized by `UM_FAC` is also invalid. Get a listing of them by setting `PRLEV` to two.

204 `UM_RFA: memory inconsistency!`
Call `UM_INI` before calling any other `UM_` subroutine.

205 `UM_RFA: out of memory! (IMEM)`
Increase the size of `IMEM`, or factorize the matrix with `UM_FAC` while reducing fill-in using the suggestions given for errors 105 and 106.

206 `UM_RFA: out of memory! (XMEM)`
Increase the size of `XMEM`, or factorize the matrix with `UM_FAC` while reducing fill-in using the suggestions given for errors 105 and 106.

207 `UM_RFA: invalid LU factors!`
Probably caused by not properly passing the factors computed by `UM_FAC` to `UM_RFA`.

3.3 UM_SOL

Given LU factors computed by UM_FAC or UM_RFA, scale factors, and the right-hand-side, b, UM_SOL computes the solution, x. If TRANS is .TRUE., then $A^T x = b$ is solved, otherwise $Ax = b$ is solved.

The computed solution, X, may overwrite the right-hand-side, B, simply by passing the right-hand-side as both B and X. This subroutine handles all permutation and scaling, so that b and x are in terms of the original triplet form of the matrix, A, and not in terms of the scaled permuted matrix. The array W is used as workspace, and must not overlap with B or X (the SDEMO and DDEMO programs allocate B, X, and W out of XMEM).

If U_{kk} is zero (actually, if its absolute value is less than or equal to ABSPT), then X (COL) is set to B (ROW) * ASR (ROW) * ASC (COL) or simply B (ROW) if no scaling is used, where k = ABS (PERMR (ROW)) = ABS (PERMC (COL)). That is, the k-th row of $LU = PD_r AD_c Q$ is replaced with the k-th row of the identity matrix. Returns NPIV, the number of nonzero entries on the diagonal of U (as determined by ABSPT).

3.3.1 Argument list

```
SUBROUTINE UM_SOL (XMEM, IMEM, PMEM, ASR, ASC, SCALE, LU, PRLEV, IO, ERROR,
        B, TRANS, ABSPT, VL, X, NPIV, W)
INTEGER IMEM (*), PMEM (7,2), LU (2,2), PRLEV, IO, ERROR, VL, NPIV
LOGICAL SCALE, TRANS
REAL/DOUBLE PRECISION XMEM (*), ASR (*), ASC (*), ABSPT, B (*), X (*), W (*)
```

3.3.2 Input

XMEM, IMEM, PMEM Holds the LU factors computed by UM_FAC or UM_RFA.

ASR, ASC, SCALE Scale factors. ASR and ASC are each of size N. The value of N is not given by an input argument. It is stored in the LU factors (see Table 1).

LU Location of the LU factors in XMEM and IMEM.

PRLEV, IO Diagnostic printing.

B Right-hand side of $Ax = b$ or $A^T x = b$. Can partially or completely overlap with X, in which case B is overwritten with X. The size of B is N.

TRANS If .TRUE., then $A^T x = b$ is solved, otherwise $Ax = b$ is solved.

ABSPT If $|U_{kk}| \leq$ABSPT, then the k-th row of the identity matrix is used instead of the k-th row of L and U during forward and backward solves (L and U are unchanged). Range: ≥ 0, typical value: same as that given to UM_FAC or UM_RFA.

VL Desired vector length for backward solve (or forward solve if TRANS is .TRUE.). If the number of pivots in a frontal matrix is less than VL, then stride-one access is abandoned in favor of longer, non-stride-one loops. Range: 1 to N, typical value: 8 to 64

3.3.3 Modified

W Used as temporary workspace. Cannot overlap B or X. The size of W is 2*N.

3.3.4 Output

`ERROR` No error if zero. Nonzero if error occurred. Error codes returned:

304 `UM_SOL: memory inconsistency!`
Call `UM_INI` before calling any other `UM_` subroutine.

307 `UM_SOL: invalid LU factors!`
Probably caused by not properly passing the factors computed by `UM_FAC` or `UM_RFA` to `UM_SOL`.

`X` Computed solution. Can partially or completely overlap with `B`, in which case `B` is overwritten with `X`. The size of `X` is `N`.

`NPIV` Number of entries on then diagonal of U with absolute value greater than `ABSPT`.

3.4 UM_SMV

The UM_SMV subroutine computes $y = Ax + y$ or $y = A^T x + y$, where A is a triplet-form sparse N-by-N matrix, and x is a dense N-by-1 column vector. Duplicate entries in A are tolerated. If TRANS is .TRUE. then $A^T x + y$ is computed, otherwise $Ax + y$ is computed. No scaling of A is used, nor are permutations used, even if permutations have been made by UM_FAC. Y should not overlap X.

3.4.1 Argument list

```
SUBROUTINE UM_SMV (AVALUE, AINDEX, N, NZ, NINVLD, PRLEV, IO, ERROR, TRANS, X,
        Y)
INTEGER AINDEX (2,NZ), N, NZ, NINVLD, PRLEV, IO, ERROR
LOGICAL TRANS
REAL/DOUBLE PRECISION AVALUE (NZ), X (N), Y (N)
```

3.4.2 Input

AVALUE, AINDEX, N, NZ Input matrix.

PRLEV, IO Diagnostic printing.

TRANS If TRANS is .TRUE. then do $A^T x$, else do Ax.

X Dense column vector, x, to pre-multiply by A or A^T.

3.4.3 Modified

Y Result of $Ax + y$ or $A^T x + y$.

3.4.4 Output

NINVLD Number of invalid entries in A. Invalid entries are ignored, and the matrix product is still computed.

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

401 UM_SMV: matrix order <= 0!, N
N must be greater than zero.

402 UM_SMV: number of nonzeros < 0!, NZ
NZ must be greater than or equal to zero.

403 UM_SMV: invalid entries in input matrix!
Check input matrix for entries with row or column indices outside the range 1 to N. Get a listing of them by setting PRLEV to two.

3.5 UM_STA

The UM_STA subroutine computes the statistics described below. It cannot be called until UM_FAC or UM_RFA have produced valid LU factors.

3.5.1 Argument list

```
SUBROUTINE UM_STA (XMEM, IMEM, PMEM, AVALUE, AINDEX, N, NZ, ASR, ASC, SCALE,
        LU, PRLEV, IO, ERROR, NZ2, NZDIA, NZOFF, SGLTNS, BLOCKS, LNZ, UNZ,
        LUNZ, NFRONT, LUOPS, SYMCNT)
INTEGER IMEM (*), PMEM (7,2), AINDEX (2,NZ), N, NZ, LU (2,2), PRLEV, IO, ERROR,
        NZ2, NZDIA, NZOFF, SGLTNS, BLOCKS, LNZ, UNZ, LUNZ, NFRONT, SYMCNT
LOGICAL SCALE
REAL/DOUBLE PRECISION XMEM (*), AVALUE (NZ), ASR (N), ASC (N), LUOPS
```

3.5.2 Input

XMEM, IMEM, PMEM Holds the LU factors computed by UM_FAC or UM_RFA.

N A and LU are N-by-N matrices.

LU Location of LU factors in XMEM and IMEM.

PRLEV, IO Diagnostic printing.

3.5.3 Output

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

501 UM_STA: different matrix order!
N must be the same as given to UM_FAC.

504 UM_STA: memory inconsistency!
Call UM_INI before calling any other UM_ subroutine.

507 UM_STA: invalid LU factors!
Probably caused by not properly passing the factors computed by UM_FAC or UM_RFA to UM_STA.

NZ2 Number of nonzeros in A, after dropping entries with absolute value less than or equal to DTOL.

NZDIA Number of nonzeros in the block-diagonal portion of A.

NZOFF Number of nonzeros in the off-diagonal blocks (NZ2 = NZDIA + NZOFF).

SGLTNS Number of 1-by-1 blocks in the block-upper-triangular form of A.

BLOCKS Number of diagonal blocks in the block-upper-triangular form of A.

LNZ Number of nonzeros in the strictly lower triangular part of L (excluding diagonal).

UNZ Number of nonzeros in the strictly upper triangular part of U (excluding diagonal).

LUNZ Number of nonzeros in $L + U$ (LUNZ = LNZ + UNZ + N + NZOFF).

NFRONT Number of frontal matrices.

LUOPS Theoretical number of floating-point operations performed during LU factorization. Note: this count does not take into consideration the floating-point operations skipped in the BLAS because of numerically zero entries in the frontal matrices, nor does it include the extra floating-point additions performed by the assembly phase. It is typically higher than the actual number of floating-point operations performed, but may be lower. UM_FAC can skip more floating-point operations than UM_RFA because UM_FAC intersperses the numerical factorization of a large frontal matrix with its amalgamation. UM_RFA can perform extra (wasted) work because it factorizes frontal matrices after amalgamation has completed. Although very uncommon, cases have been observed on non-vector computers where UM_RFA actually takes more time than UM_FAC because of this effect.

SYMCNT Number of pivots selected on the diagonal of the original matrix A (or on the diagonal of the block-upper-triangular form of A if BLOCK was .TRUE. in UM_FAC). If SYMCNT is equal to N, then only symmetric pivot permutations were used ($P = Q^T$).

3.5.4 Unused

AVALUE, AINDEX, NZ, ASR, ASC, SCALE May be used as input arguments in future releases.

3.6 UM_INI

UM_INI initializes the head and tail pointers of XMEM and IMEM. Subsequent memory allocations will be from XMEM (XHEAD..XTAIL-1) and IMEM (IHEAD..ITAIL-1). The free, allocatable memory is always a single contiguous region in XMEM and IMEM. UM_INI must be called before calling any other UM_ subroutine.

3.6.1 Argument list

```
SUBROUTINE UM_INI (XMEM, IMEM, PMEM, PRLEV, IO, ERROR, IHEAD, ITAIL, XHEAD,
        XTAIL)
INTEGER IMEM (ITAIL-1), PMEM (7,2), PRLEV, IO, ERROR, IHEAD, ITAIL, XHEAD,
        XTAIL
REAL/DOUBLE PRECISION XMEM (XTAIL-1)
```

3.6.2 Input

PRLEV, IO Diagnostic printing.

IHEAD Initial value of head pointer to free memory in IMEM.

ITAIL Initial value of tail pointer to free memory in IMEM.

XHEAD Initial value of head pointer to free memory in XMEM.

XTAIL Initial value of tail pointer to free memory in XMEM.

3.6.3 Output

PMEM Set to the initial state of free memory.

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

601 UM_INI: invalid inputs!
IHEAD, ITAIL, XHEAD, and/or XTAIL are invalid.

3.6.4 Unused

XMEM, IMEM May be used as arguments in future releases. XMEM should be at least of size XTAIL-1, and IMEM should be at least of size ITAIL-1.

3.7 UM_MCO

UM_MCO checks the validity of PMEM, and returns an error if it is invalid.

3.7.1 Argument list

```
SUBROUTINE UM_MCO (XMEM, IMEM, PMEM, PRLEV, IO, ERROR)
INTEGER IMEM (*), PMEM (7,2), PRLEV, IO, ERROR
REAL/DOUBLE PRECISION XMEM (*)
```

3.7.2 Input

PMEM Current state of memory allocation.

PRLEV, IO Diagnostic printing.

3.7.3 Output

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

602 UM_MCO: memory inconsistent!
PMEM is invalid. Call UM_INI to re-initialize.

3.7.4 Unused

XMEM, IMEM May be used as arguments in future releases.

3.8 UM_MRK

The UM_MRK subroutine saves the current head and tail pointers of both XMEM and IMEM. The mark can be used for a later restoration by UM_RST.

3.8.1 Argument list

```
SUBROUTINE UM_MRK (XMEM, IMEM, PMEM, PRLEV, IO, ERROR, MARK)
INTEGER IMEM (*), PMEM (7,2), PRLEV, IO, ERROR, MARK (2,2)
REAL/DOUBLE PRECISION XMEM (*)
```

3.8.2 Input

PMEM Current state of memory allocation.

3.8.3 Output

ERROR Set to zero. No error can occur.

MARK Current state of head and tail pointers of IMEM and XMEM.

3.8.4 Unused

XMEM, IMEM, PRLEV, IO May be used as arguments in future releases.

3.9 UM_RST

The UM_RST subroutine restores the head and tail pointers in either XMEM or IMEM. Assumes no external fragmentation exists after restoring the pointers (caller has performed his own garbage collection).

3.9.1 Argument list

```
SUBROUTINE UM_RST (XMEM, IMEM, PMEM, PRLEV, IO, ERROR, MARK, M)
INTEGER IMEM (*), PMEM (7,2), PRLEV, IO, ERROR, MARK (2,2), M
REAL/DOUBLE PRECISION XMEM (*)
```

3.9.2 Input

PRLEV, IO Diagnostic printing.

MARK The mark made by a previous call to UM_MRK.

M If M is one, then restore IMEM, otherwise restore XMEM.

3.9.3 Modified

PMEM Current state of memory allocation.

3.9.4 Output

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

603 UM_RST: invalid inputs!
MARK is invalid.

3.9.5 Unused

XMEM, IMEM May be used as arguments in future releases.

3.10 UM_GET

The UM_GET subroutine allocates memory of size S from IMEM if M is one, or from XMEM otherwise. Returns an index, P, into IMEM or XMEM. The allocated space is MEM (P..P+S-1). Returns P = 0 if not enough memory is available. Allocates from the head of memory (low indices) if HT is one, and from the tail otherwise. If allocating from the head, the smaller free space is MEM (HEAD+S..TAIL-1). If allocating from the tail, the smaller free space is MEM (HEAD..TAIL-1-S). The head and tail pointers are updated in PMEM to reflect the new size of the free memory space.

3.10.1 Argument list

```
SUBROUTINE UM_GET (XMEM, IMEM, PMEM, PRLEV, IO, ERROR, M, S, HT, P)
INTEGER IMEM (*), PMEM (7,2), PRLEV, IO, ERROR, M, S, HT, P
REAL/DOUBLE PRECISION XMEM (*)
```

3.10.2 Input

PRLEV, IO Diagnostic printing.

M If M is one, allocate from IMEM, otherwise, allocate from XMEM.

S Size of memory to allocate. Must be greater than or equal to zero.

HT If HT is one, then allocate from the head of MEM (low indices), otherwise allocate from the tail of MEM (high indices).

3.10.3 Modified

PMEM Current state of memory allocation.

3.10.4 Output

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

604 UM_GET: invalid inputs!
S must be greater than or equal to zero.

P Index into XMEM or IMEM of first allocated entry. MEM (P..P+S-1) is the allocated space. Returns P = 0 if not enough space is available.

3.10.5 Unused

XMEM, IMEM May be used as arguments in future releases.

3.11 UM_FRE

The UM_FRE subroutine deallocates memory of size S from IMEM if M is one, or from XMEM otherwise. The deallocated space is MEM (P..P+S-1). The memory allocation mechanism does not actually reclaim the space passed to it by UM_FRE, unless the space happens to be adjacent to the current free memory space. Otherwise, to actually reclaim the space you must use the UM_INI or UM_RST subroutines (and your own garbage collection to reclaim external fragmentation, if necessary).

3.11.1 Argument list

```
SUBROUTINE UM_FRE (XMEM, IMEM, PMEM, PRLEV, IO, ERROR, M, S, P)
INTEGER IMEM (*), PMEM (7,2), PRLEV, IO, ERROR, M, S, P
REAL/DOUBLE PRECISION XMEM (*)
```

3.11.2 Input

PRLEV, IO Diagnostic printing.

M If M is one, deallocate from IMEM, otherwise, deallocate from XMEM.

S Size of memory to deallocate. Must be greater than or equal to zero.

P Index into XMEM or IMEM of first entry of space to deallocate. MEM (P..P+S-1) is the space to deallocate.

3.11.3 Modified

PMEM Current state of memory allocation.

3.11.4 Output

ERROR No error if zero. Nonzero if error occurred. Error codes returned:

605 UM_FRE: invalid inputs!
Region being deallocated must be in allocated memory space (P and/or S are invalid).

3.11.5 Unused

XMEM, IMEM May be used as arguments in future releases.

4 How to install UMFPACK

This section describes how to obtain and install UMFPACK.

4.1 Getting the files

To obtain UMFPACK from NETLIB, send electronic mail to `netlib@ornl.gov` with the message:

```
send umfpack.shar from misc
```

You will be sent one or more messages that together form the `umfpack.shar` file. The file is a text archive of all source files in the UMFPACK distribution. On a UNIX system, follow the instructions in the message(s) to extract the files. On a non-UNIX system, manually edit the `umfpack.shar` file (remove all the `X` characters in the first column; each file is bracketed by a line containing "`CUT_HERE`").

You can also get UMFPACK by anonymous ftp to `ftp.cis.ufl.edu`. If you do not have a UNIX system, this is probably simpler than obtaining it from NETLIB, since you do not have to manually edit a text archive file. A sample ftp session is shown below. Commands that you enter are shown in a box. If your system gets confused by the welcome message, type a dash (`-`) as the first character in your response to the `Password:` prompt.

```
% ftp ftp.cis.ufl.edu
Connected to snoopy-le1.cis.ufl.edu.
220 snoopy FTP server (Version 2.0WU(10) Sun Apr 25 14:08:59 EDT 1993) ready.
Remote system type is UNIX.
Using binary mode to transfer files.
Name (ftp.cis.ufl.edu:user):  anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: type your complete e-mail address here
230-
230- Welcome to the University of Florida Computer & Information Sciences
230- FTP archive.  Problems to root@cis.ufl.edu.
230-
230 Guest login ok, access restrictions apply.
ftp> cd pub/umfpack
250 CWD command successful.
ftp> get README -
```

The last command will print the contents of the `README` file on your screen. Follow its instructions to obtain UMFPACK.

UMFPACK (Versions 1.0s and 1.0d) consists of 93 ANSI Fortran-77 files (23 of which are for the three test programs), plus three documentation/installation files. Also included in the distribution are six input files for the test programs, for which the UMFPACK Copyright does not apply. Two of the files are sparse matrices from the Harwell/Boeing Sparse Matrix Collection (Release 1) [5, 6]. They are included by permission.

The standard distribution consists of a total of 102 files, described in Table 4. User-callable subroutines are shown in a box. Each user-callable subroutine is followed by its primary non-user-callable subroutines.

4.2 Obtaining the required Harwell and BLAS subroutines

UMFPACK uses the following BLAS subroutines: `DCOPY`, `DGEMM`, `DGEMV`, `DSCAL`, `DSWAP`, `IDAMAX`, `SCOPY`, `SGEMM`, `SGEMV`, `SSCAL`, `SSWAP`, and `ISAMAX`. These subroutines also call the BLAS utility subroutines `XERBLA` and `LSAME`. UMFPACK also uses the Harwell `MC13E` and `MC21B` subroutines to permute a matrix into block-upper-triangular form. These BLAS and MA28 subroutines are thus not covered by the UMFPACK copyright and do not come with UMFPACK. They must be obtained separately. Ideally, optimized BLAS subroutines should already be installed on your system. If not, you can get Fortran versions from NETLIB. To obtain the required BLAS and MA28 subroutines, send electronic mail to `netlib@ornl.gov` with the message:

```
send dscal.f dgemm.f dswap.f dgemv.f dcopy.f idamax.f from blas
send sscal.f sgemm.f sswap.f sgemv.f scopy.f isamax.f from blas
send mc13e.f mc21b.f from harwell
```

You will receive the following files: `dcopy.f`, `dgemm.f`, `dgemv.f`, `dscal.f`, `dswap.f`, `idamax.f`, `scopy.f`, `sgemm.f`, `sgemv.f`, `sscal.f`, `sswap.f`, `isamax.f`, `xerbla.f`, `lsame.f`, `disclaimer`, `mc13e.f`, and `mc21b.f`. Read the Harwell `disclaimer` file and follow its instructions.

4.3 Installing UMFPACK

You should now have all 102 files in UMFPACK, the 14 BLAS files (if you need them), and the three Harwell files, for a total of 119 files.

The machine-dependent files `umdmax.f` and `umsmax.f` contain subroutines that return the maximum positive `INTEGER` value that can be represented on your computer. The default is $2^{31} - 1$, which assumes a 32-bit `INTEGER`. Edit these two files if the value on your computer is less than the default.

The remaining discussion assumes you are using a UNIX system. Place all 119 source files in a single directory. Edit the `makefile` to configure it to your system. The `makefile` includes examples for a Cray, Sun-4, and a generic UNIX system. Typing `make` will compile the entire UMFPACK distribution and run the test programs. Individual `make` commands are shown in Table 5.

If UMFPACK is installed correctly, the output of the `DEMO` program will be as shown in Section 5. The output of the `SDEMO` and `DDEMO` programs should contain no error messages (lines with a "!"). The first 52 lines of the `ddemo.out` file should look something like the following:

```
nmeth =   64
prlev =   1
--------------------------------------------------------------------------------
Matrix: ibm32
title: 1UNSYMMETRIC PATTERN ON LEAFLET ADVERTISING IBM 1971 CONFERENCE
key: IBM32
Lines: tot:             11 ptr:              3 ind:              8
       val:              0 rhs:              0
type: PUA nrow:             32 ncol:             32
```

Table 4: Source files in the Unsymmetric-pattern MultiFrontal Package

file name	description
UMD, double precision version (35 files):	
umdfac.f	factorize A in LU
umdfa0.f	initialize and factorize
umdblo.f	permute to block-upper-triangular form
umdcof.f	count entries in off-diagonal blocks
umdsma.f	store A into block-upper-triangular form
umdsva.f	store A into block-upper-triangular form, only 1 block found
umdsnb.f	store A, not permuting to block-upper-triangular form
umdfa2.f	primary factorization subroutine
umdigr.f	INTEGER garbage collection
umdxgr.f	DOUBLE-PRECISION garbage collection
umdpof.f	permute off-diagonal entries into pivot order
umdrfa.f	re-factorize A into LU
umdrf0.f	initialize and re-factorize
umdcar.f	count entries in A for conversion to arrowhead format
umdsar.f	store A into arrowhead format
umdrf2.f	primary re-factorization subroutine
umdxg2.f	DOUBLE-PRECISION garbage collection
umdsol.f	solve $Ax = b$, given LU factors.
umdsl1.f	solve $Ax = b$
umdsl2.f	solve $A^T x = b$
umdsmv.f	sparse matrix times a dense vector
umdsta.f	compute statistics from LU factors
umdst1.f	compute statistics
umdini.f	initialize allocatable memory
umdmco.f	check validity of allocatable memory
umdmrk.f	mark current state of allocatable memory
umdrst.f	restore to previously marked state of allocatable memory
umdget.f	get a region of allocatable memory
umdfre.f	free a region of allocatable memory
utility subroutines for double precision version:	
umdmax.f	return maximum INTEGER value (MACHINE DEPENDENT)
umddup.f	convert input A, remove duplicates, drop, and scale
umdlug.f	retrieve scalar information from LU factors
umdsho.f	print A to file IJORIG
umdshb.f	print block-upper-triangular form of A to file IJBLOK
umdshl.f	print LU factors to file IJLU

Table 4. Source files (continued).

file name	description
UMS, single precision version (35 files):	
	same set of file names as UMD, except with ums prefix instead of umd
DDEMO, double precision demo program (11 files):	
ddemo1.f	main demo program
ddemo2.f	read input matrix
ddemo3.f	compute scale factors
ddemo4.f	factorize a matrix, solve, re-factorize, and solve
ddemo5.f	compute b, then use LU factors to solve $Ax = b$ for x
ddemo6.f	portable random number generator
ddemo7.f	read or generate a sparse matrix
ddemo8.f	generate a band matrix, with a dense band
ddemo9.f	read a matrix in the SMMS format [1]
ddemoa.f	read a matrix in the Harwell/Boeing format [5, 6]
ddemob.f	create random duplicate entries in A (to test duplicate removal)
SDEMO, single precision demo program (11 files):	
	same set of file names as DDEMO, except with s prefix instead of d
DEMO, simple double-precision demo program (1 file):	
demo.f	factorize and solve a 5-by-5 matrix (see Section 5)
Input data for DDEMO and SDEMO (5 files):	
in	input file
ibm32	32-by-32 matrix from then Harwell/Boeing collection [5, 6]
will199	199-by-199 matrix from then Harwell/Boeing collection
ten	singular 10-by-10 matrix in triplet form
four	4-by-4 matrix in triplet form
five	symmetric 5-by-5 matrix in triplet form
Documentation and installation files (3 files)	
notice	copyright notice
manual.tex	this manual
makefile	for installing on UNIX systems

Table 5: `make` options for compiling and testing UMFPACK

command	action
`make libumd.a`	compile `UMD` subroutines and place in library `libumd.a`
`make libums.a`	compile `UMS` subroutines and place in library `libums.a`
`make ddemo`	compile the `DDEMO` program
`make sdemo`	compile the `SDEMO` program
`make demo`	compile the `DEMO` program
`make dtest`	compile and run `DEMO` and `DDEMO`
`make stest`	compile and run `SDEMO`
`make test`	compile and run all test programs
`make manual.ps`	generate Postscript version of this manual
`make clean`	remove object files and compiler listings
`make purge`	remove all but the distribution files

```
nz:            126 nrhs:              0
ptrfmt:     (16I5)           rowfmt:     (16I5)
valfmt:                      rhsfmt:
sym:   F skew:   0.
==================== SET:   0 0 0  2.000  0.100   0   4 0 ====================
-------------------- UMDfac:
fac mem size:  X:     1999874            I:      799748
fac mem usage: X:         712        671 I:        1615       1615
fac mem for LU X:         271            I:         329
fac garbage:   X:           0            I:           0
A*x=b      0.3588E+01   0.6328E-14   0.8665E-15   8
A^T*x=b    0.4738E+01   0.4885E-14   0.1594E-14   8
fac A:  n       32 nz       126 nz2       126 nzdia       126 nzoff         0
fac A:  blocks        1 singletons        0
fac A:  ninvld         0 ndrop         0 ndupl         0
fac LU: lnz       113 unz       126 lunz       271 nfront       19
fac LU: offdiag pivots       19 theor. flop count     0.1237000000E+04
fac LU: pmin   0.1044E+00 pivot failures        0
-------------------- UMDrfa:
rfa mem size:  X:     1999874            I:      799748
rfa mem usage: X:         597        556 I:         804        804
rfa garbage:   X:           0
A*x=b      0.3588E+01   0.2665E-14   0.1238E-14   8
A^T*x=b    0.4738E+01   0.4441E-14   0.1781E-14   8
-------------------- UMDfac:
fac mem size:  X:         691            I:        1615
fac mem usage: X:         684        671 I:        1615       1615
fac mem for LU X:         271            I:         329
fac garbage:   X:           1            I:           0
A*x=b      0.3588E+01   0.6328E-14   0.8665E-15   8
A^T*x=b    0.4738E+01   0.4885E-14   0.1594E-14   8
fac A:  n       32 nz       126 nz2       126 nzdia       126 nzoff         0
```

```
fac A:  blocks        1 singletons        0
fac A:  ninvld         0 ndrop         0 ndupl         0
fac LU: lnz       113 unz       126 lunz       271 nfront       19
fac LU: offdiag pivots       19 theor. flop count     0.1237000000E+04
fac LU: pmin   0.1044E+00 pivot failures        0
-------------------- UMDrfa:
rfa mem size:  X:         576            I:         804
rfa mem usage: X:         529        496 I:         804        804
rfa garbage:   X:           1
A*x=b      0.3588E+01   0.2665E-14   0.1238E-14   8
A^T*x=b    0.4738E+01   0.4441E-14   0.1781E-14   8

 ... output truncated ...
```

The lines with `A*x=b` or `A^T*x=b` show the norm of A,

$$||A||_\infty$$

the relative solution error,

$$\frac{||x_{true} - x_{computed}||_\infty}{||x_{true}||_\infty}$$

and the relative residual,

$$\frac{||Ax_{computed} - b||_\infty}{||A||_\infty}$$

in that order. The last two numbers should be fairly small, except for the singular 10-by-10 matrix in the file `ten`. See the `DDEMO` source code for details on the rest of the output.

If you do not have LaTeX, you may obtain a postscript version of this manual via anonymous ftp to `ftp.cis.ufl.edu` as the compressed postscript file `cis/tech-reports/tr93/tr93-020.ps.Z`. You may request a printed version of TR-93-020 at the following address: Technical Reports, CSE E301, Computer and Information Sciences Department, University of Florida, Gainesville, FL 32611-2024, USA.

5 Example of use

UMFPACK comes with three test programs: `SDEMO` and `DDEMO`, which test most of the features of UMFPACK, and a simple program `DEMO`, shown below (also in the `demo.f` file). The `DEMO` program uses typical parameter settings for the `UM_` subroutines (except that the `PRLEV` setting is typically zero or one, not -2). The `DEMO` program does not use scale factors (`XMEM` is passed in place of `ASR` and `ASC`; they are not accessed by the `UM_` subroutines since `SCALE` is `.FALSE.`).

```
C  simple demo program for the Unsymmetric-pattern MultiFrontal Package
C
C       Factor and solve a 5-by-5 system:
C
C       [ 2  3  0  0  0 ]     [  8 ]                    [ 1 ]
C       [ 3  0  4  0  6 ]     [ 45 ]                    [ 2 ]
C       [ 0 -1 -3  2  0 ] x = [ -3 ].   Solution is x = [ 3 ]
```

```
C       [ 0  0  1  0  0 ]     [  3 ]                    [ 4 ]
C       [ 0  4  2  0  1 ]     [ 19 ]                    [ 5 ]
C
C  prints input matrix A to the file IJORIG, A permuted to block-upper-
C  triangular form to IJBLOK, and LU factors to IJLU.

        PROGRAM DEMO

        INTEGER XS, IS
        PARAMETER (XS = 300, IS = 300)
        INTEGER IMEM (IS), PMEM (7,2), AINDEX (2,12), LU (2,2), ERROR,
     $     NDUPL, NDROP, NINVLD, NPIV, IGAR(5), XGAR(5), I
        DOUBLE PRECISION XMEM (XS), B (5), X (5), W (10), AVALUE (12),
     $     PMIN
        DATA AINDEX /1,1,   1,2,   2,1,   2,3,   2,5,   3,2,    3,3,
     $     3,4,   4,3,   5,2,   5,3,   5,5/
        DATA AVALUE /2.0D0, 3.0D0, 3.0D0, 4.0D0, 6.0D0, -1.0D0, -3.0D0,
     $     2.0D0, 1.0D0, 4.0D0, 2.0D0, 1.0D0/
        DATA B /8.0D0, 45.0D0, -3.0D0, 3.0D0, 19.0D0/

C       initialize pmem
        CALL UMDINI (XMEM, IMEM, PMEM, -2, 6, ERROR, 1, IS+1, 1, XS+1)

C       factorize A into LU (no scaling used)
        CALL UMDFAC (XMEM, IMEM, PMEM, AVALUE, AINDEX, 5, 12,
     $     .FALSE., .TRUE., .FALSE., 0.0D0, NDUPL, NDROP, NINVLD,
     $     XMEM, XMEM, .FALSE., LU, -2, 6, ERROR, .TRUE., 0.1D0,
     $     0.0D0, 2.0D0, 0, 4, .FALSE., 32, 5, NPIV, PMIN, IGAR, XGAR)
        IF (ERROR .NE. 0) STOP

C       solve A*x = b
        CALL UMDSOL (XMEM, IMEM, PMEM, XMEM, XMEM, .FALSE., LU, -2, 6,
     $     ERROR, B, .FALSE., 0.0D0, 8, X, NPIV, W)

C       print solution
        WRITE (6, 10) (X (I), I = 1,5)
10      FORMAT (' Solution: ', 5F12.5)
        STOP
        END
```

The output of the `demo` program is:

```
 Solution:      1.00000     2.00000     3.00000     4.00000     5.00000
```

The program also generates three output files, since the `PRLEV` argument is negative. The original matrix is printed to the file `IJORIG`:

```
  5   5
  1   1    2.0000000000000
  1   2    3.0000000000000
  2   1    3.0000000000000
```

```
  2   3    4.0000000000000
  2   5    6.0000000000000
  3   2   -1.0000000000000
  3   3   -3.0000000000000
  3   4    2.0000000000000
  4   3    1.0000000000000
  5   2    4.0000000000000
  5   3    2.0000000000000
  5   5    1.0000000000000
0 0 0
```

The matrix is permuted to block-upper-triangular form and printed to the file `IJBLOK`:

```
  5   5
  1   1    2.0000000000000
  4   2    3.0000000000000
  2   2    4.0000000000000
  3   3    6.0000000000000
  2   3    1.0000000000000
  4   4    2.0000000000000
  3   4    3.0000000000000
  5   5    1.0000000000000
  1   2   -1.0000000000000
  1   5   -3.0000000000000
  2   5    2.0000000000000
  3   5    4.0000000000000
0 0 0
```

Finally, the LU factors are printed to the file `IJLU` (not including the permutation arrays):

```
  5   5
  5   5    1.0000000000000
  3   5    2.0000000000000
  4   5    4.0000000000000
  3   2    1.3333333333333
  4   2  0.
  4   3   -1.1250000000000
  4   4    7.1250000000000
  3   3   -2.6666666666667
  3   4    1.0000000000000
  2   2    3.0000000000000
  2   4  0.
  2   3    2.0000000000000
  1   2   -1.0000000000000
  1   5   -3.0000000000000
  1   1    2.0000000000000
0 0 0
```

6 Copyright Notice

Sciences Department, University of Florida, USA. email: davis@cis.ufl.edu.

7 Final comments

As soon as you receive a copy of UMFPACK, please send email to me at `davis@cis.ufl.edu`, so I can put you on a mailing list for news and updates. Please include your postal address as well.

While I would appreciate hearing any bug reports and comments, I cannot promise that I can fix any specific bugs. I would also appreciate receiving copies of publications that refer to the package.

If you find this software to have a much higher performance that the software you were previously using, I would be very interested in getting copies of your sparse matrices. This software development requires large sparse matrices from a variety of disciplines (random sparse matrices are not useful). If you would like to assist in the further development of this software, please send me your matrices. Doing so will improve the performance of future versions of this software for your application. Of particular interest are large unsymmetric sparse matrices (say, 5000-by-5000 or larger) with unsymmetric nonzero pattern. The current release of the Harwell/Boeing Sparse Matrix Collection is weak in this area [5, 6].

8 Acknowledgments

This project is in collaboration with Iain Duff (Rutherford Appleton Laboratory, England, and CERFACS, Toulouse, France). Portions of this work were supported by a post-doctoral grant from CERFACS, September 1989 to December 1990. Support for this project also provided by the National Science Foundation (ASC-9111263, DMS-9223088), and by Cray Research, Inc. (with thanks to Steve Zitney) and Florida State University through the allocation of supercomputer resources.

References

[1] F. L. Alvarado. Manipulation and visualization of sparse matrices. *ORSA J. on Computing*, 2:186–207, 1990.

[2] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. Supercomputer Appl.*, 3(3):41–59, 1989.

[3] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report TR-93-018, CIS Dept., Univ. of Florida (anonymous ftp to ftp.cis.ufl.edu:cis/tech-reports/tr93/tr93-018.ps.Z), Gainesville, FL (submitted to the SIAM Journal on Matrix Analysis and Applications), 1993.

[4] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level-3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990.

[5] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15:1–14, 1989.

[6] I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (release 1). Technical Report RAL-92-086, Rutherford Appleton Laboratory, Didcot, Oxon, England, Dec. 1992.

[7] I. S. Duff and J. K. Reid. Some design features of a sparse matrix code. *ACM Trans. Math. Softw.*, 5(1):18–35, 1979.

[8] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Statist. Comput.*, 5(3):633–641, 1984.