NAG Fortran Library Routine Document

F01BRF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

F01BRF factorizes a real sparse matrix. The routine either forms the LU factorization of a permutation of the entire matrix, or, optionally, first permutes the matrix to block lower triangular form and then only factorizes the diagonal blocks.

2 Specification

```
 SUBROUTINE F01BRF(N, NZ, A, LICN, IRN, LIRN, ICN, PIVOT, IKEEP, IW, W,
1                  LBLOCK, GROW, ABORT, IDISP, IFAIL)
 INTEGER           N, NZ, LICN, IRN(LIRN), LIRN, ICN(LICN), IKEEP(5*N),
1                  IW(8*N), IDISP(10), IFAIL
 real              A(LICN), PIVOT, W(N)
 LOGICAL           LBLOCK, GROW, ABORT(4)
```

3 Description

Given a real sparse matrix A, this routine may be used to obtain the LU factorization of a permutation of A,

$$PAQ = LU$$

where P and Q are permutation matrices, L is unit lower triangular and U is upper triangular. The routine uses a sparse variant of Gaussian elimination, and the pivotal strategy is designed to compromise between maintaining sparsity and controlling loss of accuracy through round-off.

Optionally the routine first permutes the matrix into block lower triangular form and then only factorizes the diagonal blocks. For some matrices this gives a considerable saving in storage and execution time.

Extensive data checks are made; duplicated non-zeros can be accumulated.

The factorization is intended to be used by F04AXF to solve sparse systems of linear equations $Ax = b$ or $A^T x = b$. If several matrices of the same sparsity pattern are to be factorized, F01BSF should be used for the second and subsequent matrices.

The method is fully described in Duff (1977).

4 References

Duff I S (1977) MA28 – a set of Fortran subroutines for sparse unsymmetric linear equations *AERE Report R8730* HMSO

5 Parameters

1: N – INTEGER *Input*

On entry: n, the order of the matrix A.

Constraint: N > 0.

2: NZ – INTEGER *Input*

On entry: the number of non-zero elements in the matrix A.

Constraint: NZ > 0.

3: A(LICN) – ***real*** array *Input/Output*

On entry: A(i), for $i = 1, 2, \ldots,$ NZ must contain the non-zero elements of the sparse matrix A. They can be in any order since the routine will reorder them.

On exit: the non-zero elements in the LU factorization. The array must **not** be changed by the user between a call of this routine and a call of F04AXF.

4: LICN – INTEGER *Input*

On entry: the dimension of the arrays A and ICN as declared in the (sub)program from which F01BRF is called. Since the factorization is returned in A and ICN, LICN should be large enough to accommodate this and should ordinarily be 2 to 4 times as large as NZ.

Constraint: LICN $\geq$ NZ.

5: IRN(LIRN) – INTEGER array *Input/Output*

On entry: IRN(i), for $i = 1, 2, \ldots,$ NZ must contain the row index of the non-zero element stored in A(i).

On exit: the array is overwritten and is not needed for subsequent calls of F01BSF or F04AXF.

6: LIRN – INTEGER *Input*

On entry: the dimension of the array IRN as declared in the (sub)program from which F01BRF is called. It need not be as large as LICN; normally it will not need to be very much greater than NZ.

Constraint: LIRN $\geq$ NZ.

7: ICN(LICN) – INTEGER array *Input/Output*

On entry: ICN(i), for $i = 1, 2, \ldots,$ NZ must contain the column index of the non-zero element stored in A(i).

On exit: the column indices of the non-zero elements in the factorization. The array must **not** be changed by the user between a call of this routine and subsequent calls of F01BSF or F04AXF.

8: PIVOT – ***real*** *Input*

On entry: PIVOT should have a value in the range 0.0 $\leq$ PIVOT $\leq$ 0.9999 and is used to control the choice of pivots. If PIVOT $<$ 0.0, the value 0.0 is assumed, and if PIVOT $>$ 0.9999, the value 0.9999 is assumed. When searching a row for a pivot, any element is excluded which is less than PIVOT times the largest of those elements in the row available as pivots. Thus decreasing PIVOT biases the algorithm to maintaining sparsity at the expense of stability.

Suggested value: PIVOT = 0.1 has been found to work well on test examples.

9: IKEEP(5*N) – INTEGER array *Output*

On exit: indexing information about the factorization. The array must **not** be changed by the user between a call of this routine and calls of F01BSF or F04AXF.

10: IW(8*N) – INTEGER array *Workspace*

11: W(N) – ***real*** array *Output*

On exit: if GROW = .TRUE., W(1) contains an estimate (an upper bound) of the increase in size of elements encountered during the factorization (see GROW); the rest of the array is used as workspace.

If GROW = .FALSE., the array is not used.

12: LBLOCK – LOGICAL *Input*

On entry: if LBLOCK = .TRUE., the matrix is pre-ordered into block lower triangular form before the *LU* factorization is performed; otherwise the entire matrix is factorized.

Suggested value: LBLOCK = .TRUE. unless the matrix is known to be irreducible.

13: GROW – LOGICAL *Input*

On entry: if GROW = .TRUE., then on exit W(1) contains an estimate (an upper bound) of the increase in size of elements encountered during the factorization. If the matrix is well-scaled (see Section 8.2), then a high value for W(1) indicates that the *LU* factorization may be inaccurate and the user should be wary of the results and perhaps increase the parameter PIVOT for subsequent runs (see Section 7).

Suggested value: GROW = .TRUE..

14: ABORT(4) – LOGICAL array *Input*

On entry: if ABORT(1) = .TRUE., the routine will exit immediately on detecting a structural singularity (one that depends on the pattern of non-zeros) and return IFAIL = 1; otherwise it will complete the factorization (see Section 8.3).

If ABORT(2) = .TRUE., the routine will exit immediately on detecting a numerical singularity (one that depends on the numerical values) and return IFAIL = 2; otherwise it will complete the factorization (see Section 8.3).

If ABORT(3) = .TRUE., the routine will exit immediately (with IFAIL = 5) when the arrays A and ICN are filled up by the previously factorized, active and unfactorized parts of the matrix; otherwise it continues so that better guidance on necessary array sizes can be given in IDISP(6) and IDISP(7), and will exit with IFAIL in the range 4 to 6. Note that there is always an immediate error exit if the array IRN is too small.

If ABORT(4) = .TRUE., the routine exits immediately (with IFAIL = 13) if it finds duplicate elements in the input matrix. If ABORT(4) = .FALSE., the routine proceeds using a value equal to the sum of the duplicate elements. In either case details of each duplicate element are output on the current advisory message unit (see X04ABF), unless suppressed by the value of IFAIL on entry.

Suggested values:

ABORT(1) = .TRUE.
ABORT(2) = .TRUE.
ABORT(3) = .FALSE.
ABORT(4) = .TRUE.

15: IDISP(10) – INTEGER array *Output*

On exit: IDISP is used to communicate information about the factorization to the user and also between a call of F01BRF and subsequent calls to F01BSF or F04AXF.

IDISP(1) and IDISP(2) indicate the position in arrays A and ICN of the first and last elements in the *LU* factorization of the diagonal blocks. (IDISP(2) gives the number of non-zeros in the factorization.) IDISP(1) and IDISP(2) must not be changed by the user between a call of F01BRF and subsequent calls to F01BSF or F04AXF.

IDISP(3) and IDISP(4) monitor the adequacy of 'elbow room' in the arrays IRN and A/ICN respectively, by giving the number of times that the data in these arrays has been compressed during the factorization to release more storage. If either IDISP(3) or IDISP(4) is quite large (say greater than 10), it will probably pay the user to increase the size of the corresponding array(s) for subsequent runs. If either is very low or zero, then the user can perhaps save storage by reducing the size of the corresponding array(s).

IDISP(5) gives an upper bound on the rank of the matrix.

IDISP(6) and IDISP(7) give the minimum size of arrays IRN and A/ICN respectively which would enable a successful run on an identical matrix (but some ‘elbow-room’ should be allowed – see Section 8).

IDISP(8) to (10) are only used if LBLOCK = .TRUE..

IDISP(8) gives the structural rank of the matrix.

IDISP(9) gives the number of diagonal blocks.

IDISP(10) gives the size of the largest diagonal block.

16: IFAIL – INTEGER *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01).

Before entry, IFAIL must be set to a value with the decimal expansion cba, where each of the decimal digits c, b and a must have a value of 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e., hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6 Error Indicators and Warnings

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL = -2

Successful factorization of a numerically singular matrix (which may also be structurally singular) (see Section 8.3).

IFAIL = -1

Successful factorization of a structurally singular matrix (see Section 8.3).

IFAIL = 1

The matrix is structurally singular and the factorization has been abandoned (ABORT(1) was .TRUE. on entry).

IFAIL = 2

The matrix is numerically singular and the factorization has been abandoned (ABORT(2) was .TRUE. on entry).

IFAIL = 3

LIRN is too small: there is not enough space in the array IRN to continue the factorization. The user is recommended to try again with LIRN (and the length of IRN) equal to at least IDISP(6) + N/2.

IFAIL = 4

LICN is much too small: there is much too little space in the arrays A and ICN to continue the factorization.

IFAIL = 5

LICN is too small: there is not enough space in the arrays A and ICN to store the factorization. If ABORT(3) was .FALSE. on entry, the factorization has been completed but some of the LU factors have been discarded to create space; IDISP(7) then gives the minimum value of LICN (i.e., the minimum length of A and ICN) required for a successful factorization of the same matrix.

IFAIL = 6

LICN and LIRN are both too small: effectively this is a combination of IFAIL = 3 and IFAIL = 5 (with ABORT(3) = .FALSE.).

IFAIL = 7

LICN is too small: there is not enough space in the arrays A and ICN for the permutation to block triangular form.

IFAIL = 8

On entry, N $\leq$ 0.

IFAIL = 9

On entry, NZ $\leq$ 0.

IFAIL = 10

On entry, LICN < NZ.

IFAIL = 11

On entry, LIRN < NZ.

IFAIL = 12

On entry, an element of the input matrix has a row or column index (i.e., an element of IRN or ICN) outside the range 1 to N.

IFAIL = 13

Duplicate elements have been found in the input matrix and the factorization has been abandoned (ABORT(4) = .TRUE. on entry).

7 Accuracy

The factorization obtained is exact for a perturbed matrix whose (i,j)th element differs from a_{ij} by less than $3\epsilon\rho m_{ij}$ where ϵ is the ***machine precision***, ρ is the growth value returned in W(1) if GROW = .TRUE., and m_{ij} the number of Gaussian elimination operations applied to element (i,j). The value of m_{ij} is not greater than n and is usually much less. Small ρ-values therefore guarantee accurate results, but unfortunately large ρ-values may give a very pessimistic indication of accuracy.

8 Further Comments

8.1 Timing

The time required may be estimated very roughly from the number τ of non-zeros in the factorized form (output as IDISP(2)) and for this routine and its associates is

F01BRF:	$5\tau^2/n$ units
F01BSF:	τ^2/n units
F04AXF:	2τ units

where our unit is the time for the inner loop of a full matrix code (e.g., solving a full set of equations takes about $\frac{1}{3}n^3$ units). Note that the faster F01BSF time makes it well worthwhile to use this for a sequence of problems with the same pattern.

It should be appreciated that τ varies widely from problem to problem. For network problems it may be little greater than NZ, the number of non-zeros in A; for discretisation of 2-dimensional and 3-dimensional partial differential equations it may be about $3n\log_2 n$ and $\frac{1}{2}n^{5/3}$, respectively.

The time taken to find the block lower triangular form (LBLOCK = .TRUE.) is typically 5–15% of the time taken by the routine when it is not found (LBLOCK = .FALSE.). If the matrix is irreducible (IDISP(9) $= 1$ after a call with LBLOCK = .TRUE.) then this time is wasted. Otherwise, particularly if the largest block is small (IDISP(10) $\ll n$), the consequent savings are likely to be greater.

The time taken to estimate growth (GROW = .TRUE.) is typically under 20% of the overall time.

The overall time may be substantially increased if there is inadequate ‘elbow-room’ in the arrays A, IRN and ICN. When the sizes of the arrays are minimal (IDISP(6) and IDISP(7)) it can execute as much as three times slower. Values of IDISP(3) and IDISP(4) greater than about 10 indicate that it may be worthwhile to increase array sizes.

8.2 Scaling

The use of a relative pivot tolerance PIVOT essentially presupposes that the matrix is well-scaled, i.e., that the matrix elements are broadly comparable in size. Practical problems are often naturally well-scaled but particular care is needed for problems containing mixed types of variables (for example millimetres and neutron fluxes).

8.3 Singular and Rectangular Systems

It is envisaged that this routine will almost always be called for square non-singular matrices and that singularity indicates an error condition. However, even if the matrix is singular it is possible to complete the factorization. It is even possible for F04AXF to solve a set of equations whose matrix is singular provided the set is consistent.

Two forms of singularity are possible. If the matrix would be singular for any values of the non-zeros (e.g., if it has a whole row of zeros), then we say it is structurally singular, and continue only if ABORT(1) = .FALSE.. If the matrix is non-singular by virtue of the particular values of the non-zeros, then we say that it is numerically singular and continue only if ABORT(2) = .FALSE..

Rectangular matrices may be treated by setting N to the larger of the number of rows and numbers of columns and setting ABORT(1) = .FALSE..

Note: the **soft failure** option should be used (last digit of IFAIL $= 1$) if the user wishes to factorize singular matrices with ABORT(1) or ABORT(2) set to .FALSE..

8.4 Duplicated Non-zeros

The matrix A may consist of a sum of contributions from different sub-systems (for example finite elements). In such cases the user may rely on this routine to perform assembly, since duplicated elements are summed.

8.5 Determinant

The following code may be used to compute the determinant of A (as the ***real*** variable DET) after a call of F01BRF:

```
         DET = 1.0
         ID = IDISP(1)
         DO 10 I = 1, N
            IDG = ID + IKEEP(3*N+I)
            DET = DET*A(IDG)
            IF (IKEEP(N+I).NE.I)DET = -DET
            IF (IKEEP(2*N+I).NE.I)DET = -DET
            ID = ID + IKEEP(I)
      10 CONTINUE
```

9 Example

To factorize the real sparse matrix:

$$\begin{pmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ -2 & 0 & 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 & 2 & -3 \\ -1 & -1 & 0 & 0 & 0 & 6 \end{pmatrix}.$$

This example program simply prints out some information about the factorization as returned by F01BRF in W(1) and IDISP. Normally the call of F01BRF would be followed by a call of F04AXF (see Example for F04AXF).

9.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     F01BRF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, NZMAX, LICN, LIRN
      PARAMETER        (NMAX=20,NZMAX=50,LICN=3*NZMAX,LIRN=3*NZMAX/2)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             U
      INTEGER          I, IFAIL, N, NZ
      LOGICAL          GROW, LBLOCK
*     .. Local Arrays ..
      real             A(LICN), W(NMAX)
      INTEGER          ICN(LICN), IDISP(10), IKEEP(NMAX,5), IRN(LIRN),
     +                 IW(NMAX,8)
      LOGICAL          ABORT(4)
*     .. External Subroutines ..
      EXTERNAL         F01BRF, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'F01BRF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N, NZ
      CALL X04ABF(1,NOUT)
      IF (N.GT.0 .AND. N.LE.NMAX .AND. NZ.GT.0 .AND. NZ.LE.NZMAX) THEN
         READ (NIN,*) (A(I),IRN(I),ICN(I),I=1,NZ)
         U = 0.1e0
         LBLOCK = .TRUE.
         GROW = .TRUE.
         ABORT(1) = .TRUE.
         ABORT(2) = .TRUE.
         ABORT(3) = .FALSE.
         ABORT(4) = .TRUE.
         IFAIL = 110
*
         CALL F01BRF(N,NZ,A,LICN,IRN,LIRN,ICN,U,IKEEP,IW,W,LBLOCK,GROW,
     +               ABORT,IDISP,IFAIL)
```

```
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999)
     +     'Number of non-zeros in decomposition         =', IDISP(2)
         WRITE (NOUT,99999)
     +     'Minimum size of array IRN                    =', IDISP(6)
         WRITE (NOUT,99999)
     +     'Minimum size of arrays A and ICN             =', IDISP(7)
         WRITE (NOUT,99999)
     +     'Number of compresses on IRN (IDISP(3))       =', IDISP(3)
         WRITE (NOUT,99999)
     +     'Number of compresses on A and ICN (IDISP(4)) =', IDISP(4)
         IF (GROW) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99998) 'Value of W(1) =', W(1)
         END IF
         IF (LBLOCK) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Structural rank                 =',
     +        IDISP(8)
            WRITE (NOUT,99999) 'Number of diagonal blocks      =',
     +        IDISP(9)
            WRITE (NOUT,99999) 'Size of largest diagonal block =',
     +        IDISP(10)
         END IF
      ELSE
         WRITE (NOUT,99999) 'N or NZ is out of range:  N = ', N,
     +     '  NZ = ', NZ
      END IF
      STOP
*
99999 FORMAT (1X,A,I5,A,I5)
99998 FORMAT (1X,A,F8.4)
      END
```

9.2 Program Data

```
F01BRF Example Program Data
  6 15
   5.0  1  1   2.0  2  2  -1.0  2  3   2.0  2  4   3.0  3  3
  -2.0  4  1   1.0  4  4   1.0  4  5  -1.0  5  1  -1.0  5  4
   2.0  5  5  -3.0  5  6  -1.0  6  1  -1.0  6  2   6.0  6  6
```

9.3 Program Results

```
 F01BRF Example Program Results

 Number of non-zeros in decomposition         =   16
 Minimum size of array IRN                    =   15
 Minimum size of arrays A and ICN             =   19
 Number of compresses on IRN (IDISP(3))       =    0
 Number of compresses on A and ICN (IDISP(4)) =    0

 Value of W(1) = 18.0000

 Structural rank                =    6
 Number of diagonal blocks      =    3
 Size of largest diagonal block =    4
```