NAG Toolbox for MATLAB

f01br

1 Purpose

f01br factorizes a real sparse matrix. The function 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 Syntax

```
[a, irn, icn, ikeep, w, idisp, ifail] = f01br(n, nz, a, irn, icn, abort, 'licn', licn, 'lirn', lirn, 'pivot', pivot, 'lblock', lblock, 'grow', grow)
```

3 Description

Given a real sparse matrix A, f01br 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 function 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 function 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 nonzeros can be accumulated.

The factorization is intended to be used by f04ax to solve sparse systems of linear equations $Ax = b$ or $A^{\mathrm{T}}x = b$. If several matrices of the same sparsity pattern are to be factorized, f01bs should be used for the second and subsequent matrices.

The method is fully described in Duff 1977.

A more recent algorithm for the same calculation is provided by f11me.

4 References

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

5 Parameters

5.1 Compulsory Input Parameters

1: **n – int32 scalar**

n, the order of the matrix A.

Constraint: **n** > 0.

2: **nz – int32 scalar**

the number of nonzero elements in the matrix A.

Constraint: **nz** > 0.

3: **a(licn) – double array**

a(i), for $i = 1, 2, \ldots,$ **nz**, must contain the nonzero elements of the sparse matrix A. They can be in any order since f01br will reorder them.

4: **irn(lirn) – int32 array**

irn(i), for $i = 1, 2, \ldots,$ **nz**, must contain the row index of the nonzero element stored in **a**(i).

5: **icn(licn) – int32 array**

icn(i), for $i = 1, 2, \ldots,$ **nz**, must contain, on entry, the column index of the nonzero element stored in **a**(i). **icn** contains, on exit, the column indices of the nonzero elements in the factorization. The array must **not** be changed by you between a call of f01br and subsequent calls of f01bs or f04ax.

6: **abort(4) – logical array**

If **abort**$(1) =$ **true**, f01br will exit immediately on detecting a structural singularity (one that depends on the pattern of nonzeros) and return **ifail** $= 1$; otherwise it will complete the factorization (see Section 8.3).

If **abort**$(2) =$ **true**, f01br 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**, f01br 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**, f01br exits immediately (with **ifail** $= 13$) if it finds duplicate elements in the input matrix.

If **abort**$(4) =$ **false**, f01br 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 x04ab), unless suppressed by the value of **ifail** on entry.

Suggested value:

abort$(1) =$ **true**;
abort$(2) =$ **true**;
abort$(3) =$ **false**;
abort$(4) =$ **true**.

5.2 Optional Input Parameters

1: **licn – int32 scalar**

Default: The dimension of the arrays **a**, **icn**. (An error is raised if these dimensions are not equal.)

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**.

2: **lirn – int32 scalar**

Default: The dimension of the array **irn**.

It need not be as large as **licn**; normally it will not need to be very much greater than **nz**.

Constraint: **lirn** $\geq$ **nz**.

3: **pivot – double scalar**

Should have a value in the range $0.0 \leq \textbf{pivot} \leq 0.9999$ and is used to control the choice of pivots. If $\textbf{pivot} < 0.0$, the value 0.0 is assumed, and if $\textbf{pivot} > 0.9999$, 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: $\textbf{pivot} = 0.1$ has been found to work well on test examples.

Default: 0.1

4: **lblock – logical scalar**

If $\textbf{lblock} = \textbf{true}$, the matrix is preordered into block lower triangular form before the *LU* factorization is performed; otherwise the entire matrix is factorized.

Suggested value: $\textbf{lblock} = \textbf{true}$ unless the matrix is known to be irreducible, or is singular and an upper bound on the rank is required.

Default: **true**

5: **grow – logical scalar**

If $\textbf{grow} = \textbf{true}$, then on exit $\mathbf{w}(1)$ 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 $\mathbf{w}(1)$ indicates that the *LU* factorization may be inaccurate and you should be wary of the results and perhaps increase the parameter **pivot** for subsequent runs (see Section 7).

Suggested value: $\textbf{grow} = \textbf{true}$.

Default: **true**

5.3 Input Parameters Omitted from the MATLAB Interface

iw

5.4 Output Parameters

1: **a(licn) – double array**

The nonzero elements in the *LU* factorization. The array must **not** be changed by you between a call of f01br and a call of f04ax.

2: **irn(lirn) – int32 array**

The array is overwritten and is not needed for subsequent calls of f01bs or f04ax.

3: **icn(licn) – int32 array**

$\mathbf{icn}(i)$, for $i = 1, 2, \ldots, \mathbf{nz}$, must contain, on entry, the column index of the nonzero element stored in $\mathbf{a}(i)$. **icn** contains, on exit, the column indices of the nonzero elements in the factorization. The array must **not** be changed by you between a call of f01br and subsequent calls of f01bs or f04ax.

4: **ikeep(5 × n) – int32 array**

Indexing information about the factorization.

You must **not** change **ikeep** between a call of f01br and subsequent calls to f01bs or f04ax.

5: **w(n) – double array**

If $\textbf{grow} = \textbf{true}$, $\mathbf{w}(1)$ 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 $\textbf{grow} = \textbf{false}$, the array is not used.

6: **idisp(10) – int32 array**

Contains information about the factorization.

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 nonzeros in the factorization.) **idisp**(1) and **idisp**(2) must not be changed by you between a call of f01br and subsequent calls to f01bs or f04ax.

idisp(3) and **idisp**(4) monitor the adequacy of 'elbow room' in the arrays **irn** and **a** (and **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 you to increase the size of the corresponding array(s) for subsequent runs. If either is very low or zero, then you can perhaps save storage by reducing the size of the corresponding array(s).

idisp(5), when **lblock** = **false**, gives an upper bound on the rank of the matrix; when **lblock** = **true**, gives an upper bound on the sum of the ranks of the lower triangular blocks.

idisp(6) and **idisp**(7) give the minimum size of arrays **irn** and **a** (and **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.

You must **not** change **idisp** between a call of f01br and subsequent calls to f01bs or f04ax.

7: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

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. You are 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 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** ≤ 0.

ifail $= 9$

On entry, **nz** ≤ 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 $\mathbf{w}(1)$ 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 nonzeros in the factorized form (output as **idisp**(2)) and for f01br and its associates is

f01br: $5\tau^2/n$ units
f01bs: τ^2/n units
f04ax: 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 f01bs 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 nonzeros in A; for discretization of two-dimensional and three-dimensional partial differential equations it may be about $3n\log_2 n$ and $\frac{1}{2}n^{5/3}$, respectively.

The time taken by f01br to find the block lower triangular form (**lblock** = **true**) is typically $5 - -15\%$ of the time taken by the function 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 f01br will almost always be called for square nonsingular 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 f04ax 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 nonzeros (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 nonsingular by virtue of the particular values of the nonzeros, then we say that it is numerically singular and continue only if **abort**(2) = **false**, in which case an upper bound on the rank of the matrix is returned in **idisp**(5) when **lblock** = **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 you wish 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 you may rely on f01br 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 double variable `deta`) after a call of f01br:

```
deta = 1;
id = idisp(1);
```

```
for i = 1:n
  idg = id + ikeep(3*n+i);
  deta = deta*a(idg);
  if (ikeep(n+i) ~= i)     deta = -deta;
  end
  if (ikeep(2*n+i) ~= i)     deta = -deta;
  end
  id = id + ikeep(i);
end
```

9 Example

```
n = int32(6);
nz = int32(15);
a = zeros(150,1);
a(1:15) = [5; 2; -1; 2; 3; -2; 1; 1; -1; -1; 2; -3; -1; -1; 6];
irn = zeros(75,1,'int32');
irn(1:15) = [int32(1); int32(2); int32(2); int32(2); int32(3); int32(4);
...
              int32(4); int32(4); int32(5); int32(5); int32(5); int32(5);
...
             int32(6); int32(6); int32(6)];
icn = zeros(150,1,'int32');
icn(1:15) = [int32(1); int32(2); int32(3); int32(4); int32(3); int32(1);
...
              int32(4); int32(5); int32(1); int32(4); int32(5); int32(6);
...
             int32(1); int32(2); int32(6)];
abort = [true;
     true;
     false;
     true];
[aOut, irnOut, icnOut, ikeep, w, idisp, ifail] = f01br(n, nz, a, irn,
icn, abort)
```

```
aOut =
     array elided
irnOut =
     array elided
icnOut =
     array elided
ikeep =
           1
           1
           2
           3
           3
           2
          -1
          -3
           6
           2
           5
           4
           1
           3
           2
           6
           5
           4
           0
           0
           0
           1
           1
           1
           0
```

```
           0
           1
           1
           1
           1
w =
   18.0000
         0
    2.0000
    0.5000
         0
         0
idisp =
           5
          16
           0
           0
           6
          15
          19
           6
           3
           4
ifail =
           0
```