NAG Library Routine Document

F11JBF

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

F11JBF solves a system of linear equations involving the incomplete Cholesky preconditioning matrix generated by F11JAF.

2 Specification

```
SUBROUTINE F11JBF (N, A, LA, IROW, ICOL, IPIV, ISTR, CHECK, Y, X, IFAIL)

INTEGER            N, LA, IROW(LA), ICOL(LA), IPIV(N), ISTR(N+1), IFAIL
REAL (KIND=nag_wp) A(LA), Y(N), X(N)
CHARACTER(1)       CHECK
```

3 Description

F11JBF solves a system of linear equations

$$Mx = y$$

involving the preconditioning matrix $M = PLDL^{\mathrm{T}}P^{\mathrm{T}}$, corresponding to an incomplete Cholesky decomposition of a sparse symmetric matrix stored in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the F11 Chapter Introduction), as generated by F11JAF.

In the above decomposition L is a lower triangular sparse matrix with unit diagonal, D is a diagonal matrix and P is a permutation matrix. L and D are supplied to F11JBF through the matrix

$$C = L + D^{-1} - I$$

which is a lower triangular N by N sparse matrix, stored in SCS format, as returned by F11JAF. The permutation matrix P is returned from F11JAF via the array IPIV.

It is envisaged that a common use of F11JBF will be to carry out the preconditioning step required in the application of F11GEF to sparse symmetric linear systems. F11JBF is used for this purpose by the Black Box routine F11JCF.

F11JBF may also be used in combination with F11JAF to solve a sparse symmetric positive definite system of linear equations directly (see Section 9.4 in F11JAF). This use of F11JBF is demonstrated in Section 10.

4 References

None.

5 Parameters

1: N – INTEGER *Input*

On entry: n, the order of the matrix M. This **must** be the same value as was supplied in the preceding call to F11JAF.

Constraint: $N \geq 1$.

2: A(LA) – REAL (KIND=nag_wp) array *Input*

On entry: the values returned in the array A by a previous call to F11JAF.

3: LA – INTEGER *Input*

On entry: the dimension of the arrays A, IROW and ICOL as declared in the (sub)program from which F11JBF is called. This **must** be the same value returned by the preceding call to F11JAF.

4: IROW(LA) – INTEGER array *Input*
5: ICOL(LA) – INTEGER array *Input*
6: IPIV(N) – INTEGER array *Input*
7: ISTR($N+1$) – INTEGER array *Input*

On entry: the values returned in arrays IROW, ICOL, IPIV and ISTR by a previous call to F11JAF.

8: CHECK – CHARACTER(1) *Input*

On entry: specifies whether or not the input data should be checked.

CHECK = 'C'
    Checks are carried out on the values of N, IROW, ICOL, IPIV and ISTR.

CHECK = 'N'
    No checks are carried out.

See also Section 9.2.

Constraint: CHECK = 'C' or 'N'.

9: Y(N) – REAL (KIND=nag_wp) array *Input*

On entry: the right-hand side vector y.

10: X(N) – REAL (KIND=nag_wp) array *Output*

On exit: the solution vector x.

11: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

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 $= 1$

On entry, CHECK $\neq$ 'C' or 'N'.

IFAIL $= 2$

On entry, $N < 1$.

IFAIL $= 3$

On entry, the SCS representation of the preconditioning matrix M is invalid. Further details are given in the error message. Check that the call to F11JBF has been preceded by a valid call to F11JAF and that the arrays A, IROW, ICOL, IPIV and ISTR have not been corrupted between the two calls.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

The computed solution x is the exact solution of a perturbed system of equations $(M + \delta M)x = y$, where

$$|\delta M| \leq c(n)\epsilon P|L||D||L^{\mathrm{T}}|P^{\mathrm{T}},$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

8 Parallelism and Performance

Not applicable.

9 Further Comments

9.1 Timing

The time taken for a call to F11JBF is proportional to the value of NNZC returned from F11JAF.

9.2 Use of CHECK

It is expected that a common use of F11JBF will be to carry out the preconditioning step required in the application of F11GEF to sparse symmetric linear systems. In this situation F11JBF is likely to be called many times with the same matrix M. In the interests of both reliability and efficiency, you are recommended to set CHECK = 'C' for the first of such calls, and to set CHECK = 'N' for all subsequent calls.

10 Example

This example reads in a symmetric positive definite sparse matrix A and a vector y. It then calls F11JAF, with LFILL $= -1$ and DTOL = 0.0, to compute the **complete** Cholesky decomposition of A:

$$A = PLDL^{\mathrm{T}}P^{\mathrm{T}}.$$

Then it calls F11JBF to solve the system

$$PLDL^{\mathrm{T}}P^{\mathrm{T}}x = y.$$

It then repeats the exercise for the same matrix permuted with the bandwidth-reducing Reverse Cuthill–McKee permutation, calculated with F11YEF.

10.1 Program Text

```
    Program f11jbfe

!     F11JBF Example Program Text

!     Mark 25 Release. NAG Copyright 2014.

!     .. Use Statements ..
      Use nag_library, Only: f11jaf, f11jbf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: dscale, dtol
      Integer                              :: i, ifail, la, lfill, liwork, n,  &
                                              nnz, nnzc, npivm
      Character (1)                        :: check, mic, pstrat
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: a(:), x(:), y(:)
      Integer, Allocatable                 :: icol(:), ipiv(:), irow(:),       &
                                              istr(:), iwork(:), perm_fwd(:),  &
                                              perm_inv(:)
!     .. Executable Statements ..
      Write (nout,*) 'F11JBF Example Program Results'
!     Skip heading in data file
      Read (nin,*)

!     Read order of matrix and number of non-zero entries

      Read (nin,*) n
      Read (nin,*) nnz

      la = 3*nnz
      liwork = 2*la + 7*n + 1

      Allocate (a(la),x(n),y(n),icol(la),ipiv(n),irow(la),istr(n+1), &
        iwork(liwork),perm_fwd(n),perm_inv(n))

!     Read the matrix A


      Do i = 1, nnz
        Read (nin,*) a(i), irow(i), icol(i)
      End Do

!     Read the vector y

      Read (nin,*) y(1:n)

!     Calculate Cholesky factorization

      lfill = -1
      dtol = 0.0E0_nag_wp
      mic = 'N'
      dscale = 0.0E0_nag_wp
      pstrat = 'M'

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call f11jaf(n,nnz,a,la,irow,icol,lfill,dtol,mic,dscale,pstrat,ipiv,istr, &
        nnzc,npivm,iwork,liwork,ifail)

!     Check the output value of NPIVM
      If (npivm/=0) Then
```

```
          Write (nout,99998) 'Factorization is not complete', npivm
        Else
!         Solve P L D L^T P^T x = y
          check = 'C'
          ifail = 0
          Call f11jbf(n,a,la,irow,icol,ipiv,istr,check,y,x,ifail)
!         Output results
          Write (nout,*) ' Solution of linear system'
          Write (nout,99999) x(1:n)
        End If

!       Compute reverse Cuthill-McKee permutation for bandwidth reduction
        Call do_rcm(irow,icol,a,y,istr,perm_fwd,perm_inv,iwork)

        ifail = 0
        Call f11jaf(n,nnz,a,la,irow,icol,lfill,dtol,mic,dscale,pstrat,ipiv,istr, &
          nnzc,npivm,iwork,liwork,ifail)

!       Check the output value of NPIVM
        If (npivm/=0) Then
          Write (nout,99998) 'Factorization is not complete', npivm
        Else
!         Solve P L D L^T P^T x = y
          ifail = 0
          Call f11jbf(n,a,la,irow,icol,ipiv,istr,check,y,x,ifail)
!         Output results
          Write (nout,*) ' Solution of linear system with Reverse Cuthill-McKee'
          Write (nout,99999)(x(perm_inv(i)),i=1,n)
        End If

99999 Format (1X,E16.4)
99998 Format (1X,A,I20)
      Contains
        Subroutine do_rcm(irow,icol,a,y,istr,perm_fwd,perm_inv,iwork)

!         .. Use Statements ..
          Use nag_library, Only: f11yef, f11zaf, f11zbf
!         .. Parameters ..
          Logical, Parameter                     ::                           &
                                    lopts(5) = (/.False.,.False.,.True.,.True., &
                                                 .True./)
!         .. Array Arguments ..
          Real (Kind=nag_wp), Intent (Inout)     :: a(la), y(n)
          Integer, Intent (Inout)                :: icol(la), irow(la), istr(n+1), &
                                                    iwork(*)
          Integer, Intent (Out)                  :: perm_fwd(n), perm_inv(n)
!         .. Local Scalars ..
          Integer                                :: i, ifail, j, nnz_cs, nnz_scs
!         .. Local Arrays ..
          Real (Kind=nag_wp), Allocatable        :: rwork(:)
          Integer                                :: info(4), mask(1)
!         .. Intrinsic Procedures ..
          Intrinsic                              :: size
!         .. Executable Statements ..

!         SCS to CS, must add the upper triangle entries.
          j = nnz + 1
          Do i = 1, nnz
            If (irow(i)>icol(i)) Then
!             strictly lower triangle, add the transposed
              a(j) = a(i)
              irow(j) = icol(i)
              icol(j) = irow(i)
              j = j + 1
            End If
          End Do
          nnz_cs = j - 1

!         Reorder, CS to CCS, icolzp in istr
          ifail = 0
          Call f11zaf(n,nnz_cs,a,icol,irow,'F','F',istr,iwork,ifail)
```

```
!       Calculate reverse Cuthill-McKee
        ifail = 0
        Call f11yef(n,nnz_cs,istr,irow,lopts,mask,perm_fwd,info,ifail)

!       compute inverse perm, in perm_inv(1:n)
        Do i = 1, n
          perm_inv(perm_fwd(i)) = i
        End Do

!       Apply permutation on column/row indices
        icol(1:nnz_cs) = perm_inv(icol(1:nnz_cs))
        irow(1:nnz_cs) = perm_inv(irow(1:nnz_cs))

!       restrict to lower triangle, SCS format
!       copying entries upwards
        j = 1
        Do i = 1, nnz_cs
          If (irow(i)>=icol(i)) Then
!           non-upper triangle, bubble up
            a(j) = a(i)
            icol(j) = icol(i)
            irow(j) = irow(i)
            j = j + 1
          End If
        End Do
        nnz_scs = j - 1

!       sort
        ifail = 0
        Call f11zbf(n,nnz_scs,a,irow,icol,'S','K',istr,iwork,ifail)

!       permute rhs vector
        Allocate (rwork(size(perm_fwd)))
        rwork(:) = y(perm_fwd(:))
        y(:) = rwork(:)
        Deallocate (rwork)
      End Subroutine do_rcm
    End Program f11jbfe
```

10.2 Program Data

```
F11JBF Example Program Data
  9                   N
 23                   NNZ
  4.   1    1
 -1.   2    1
  6.   2    2
  1.   3    2
  2.   3    3
  3.   4    4
  2.   5    1
  4.   5    5
  1.   6    3
  2.   6    4
  6.   6    6
 -4.   7    2
  1.   7    5
 -1.   7    6
  6.   7    7
 -1.   8    4
 -1.   8    6
  3.   8    8
  1.   9    1
  1.   9    5
 -1.   9    6
```

```
  1.   9    8
  4.   9    9          A(I), IROW(I), ICOL(I), I=1,...,NNZ
  4.10 -2.94  1.41
  2.53  4.35  1.29
  5.01  0.52  4.57     Y(I), I=1,...,N
```

10.3 Program Results

```
 F11JBF Example Program Results
  Solution of linear system
       0.7000E+00
       0.1600E+00
       0.5200E+00
       0.7700E+00
       0.2800E+00
       0.2100E+00
       0.9300E+00
       0.2000E+00
       0.9000E+00
  Solution of linear system with Reverse Cuthill-McKee
       0.7000E+00
       0.1600E+00
       0.5200E+00
       0.7700E+00
       0.2800E+00
       0.2100E+00
       0.9300E+00
       0.2000E+00
       0.9000E+00
```