NAG Library Routine Document

F11DUF

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

F11DUF solves a complex sparse non-Hermitian system of linear equations, represented in coordinate storage format, using a restarted generalized minimal residual (RGMRES), conjugate gradient squared (CGS), stabilized bi-conjugate gradient (Bi-CGSTAB), or transpose-free quasi-minimal residual (TFQMR) method, with block Jacobi or additive Schwarz preconditioning.

2 Specification

```
SUBROUTINE F11DUF (METHOD, N, NNZ, A, LA, IROW, ICOL, NB, ISTB, INDB,      &
                   LINDB, IPIVP, IPIVQ, ISTR, IDIAG, B, M, TOL, MAXITN,    &
                   X, RNORM, ITN, WORK, LWORK, IFAIL)

INTEGER                   N, NNZ, LA, IROW(LA), ICOL(LA), NB, ISTB(NB+1),  &
                          INDB(LINDB), LINDB, IPIVP(LINDB), IPIVQ(LINDB),  &
                          ISTR(LINDB+1), IDIAG(LINDB), M, MAXITN, ITN,     &
                          LWORK, IFAIL
REAL (KIND=nag_wp)        TOL, RNORM
COMPLEX (KIND=nag_wp)     A(LA), B(N), X(N), WORK(LWORK)
CHARACTER(*)              METHOD
```

3 Description

F11DUF solves a complex sparse non-Hermitian linear system of equations

$$Ax = b,$$

using a preconditioned RGMRES (see Saad and Schultz (1986)), CGS (see Sonneveld (1989)), Bi-CGSTAB(ℓ) (see Van der Vorst (1989) and Sleijpen and Fokkema (1993)), or TFQMR (see Freund and Nachtigal (1991) and Freund (1993)) method.

F11DUF uses the incomplete (possibly overlapping) block LU factorization determined by F11DTF as the preconditioning matrix. A call to F11DUF must always be preceded by a call to F11DTF. Alternative preconditioners for the same storage scheme are available by calling F11DQF or F11DSF.

The matrix A, and the preconditioning matrix M, are represented in coordinate storage (CS) format (see Section 2.1.1 in the F11 Chapter Introduction) in the arrays A, IROW and ICOL, as returned from F11DTF. The array A holds the nonzero entries in these matrices, while IROW and ICOL hold the corresponding row and column indices.

F11DUF is a Black Box routine which calls F11BRF, F11BSF and F11BTF. If you wish to use an alternative storage scheme, preconditioner, or termination criterion, or require additional diagnostic information, you should call these underlying routines directly.

4 References

Freund R W (1993) A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems *SIAM J. Sci. Comput.* **14** 470–482

Freund R W and Nachtigal N (1991) QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems *Numer. Math.* **60** 315–339

Saad Y and Schultz M (1986) GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869

Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32

Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52

Van der Vorst H (1989) Bi-CGSTAB, a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644

5 Parameters

1: METHOD – CHARACTER(*) *Input*

On entry: specifies the iterative method to be used.

METHOD = 'RGMRES'
Restarted generalized minimum residual method.

METHOD = 'CGS'
Conjugate gradient squared method.

METHOD = 'BICGSTAB'
Bi-conjugate gradient stabilized (ℓ) method.

METHOD = 'TFQMR'
Transpose-free quasi-minimal residual method.

Constraint: METHOD = 'RGMRES', 'CGS', 'BICGSTAB' or 'TFQMR'.

2: N – INTEGER *Input*
3: NNZ – INTEGER *Input*
4: A(LA) – COMPLEX (KIND=nag_wp) array *Input*
5: LA – INTEGER *Input*
6: IROW(LA) – INTEGER array *Input*
7: ICOL(LA) – INTEGER array *Input*
8: NB – INTEGER *Input*
9: ISTB(NB + 1) – INTEGER array *Input*
10: INDB(LINDB) – INTEGER array *Input*
11: LINDB – INTEGER *Input*
12: IPIVP(LINDB) – INTEGER array *Input*
13: IPIVQ(LINDB) – INTEGER array *Input*
14: ISTR(LINDB + 1) – INTEGER array *Input*
15: IDIAG(LINDB) – INTEGER array *Input*

On entry: the values returned in arrays IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG by a previous call to F11DTF.

The arrays ISTB, INDB and A together with the scalars N, NNZ, LA, NB and LINDB must be the same values that were supplied in the preceding call to F11DTF.

On entry: the values returned in arrays IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG by a previous call to F11DTF.

The arrays ISTB, INDB and the scalars NB and LINDB must be the same values that were supplied in the preceding call to F11DTF.

16: B(N) – COMPLEX (KIND=nag_wp) array *Input*

On entry: the right-hand side vector b.

17: M – INTEGER *Input*

On entry: if METHOD = 'RGMRES', M is the dimension of the restart subspace.

If METHOD = 'BICGSTAB', M is the order ℓ of the polynomial Bi-CGSTAB method.

Otherwise, M is not referenced.

Constraints:

if METHOD = 'RGMRES', $0 < \mathrm{M} \leq \min(\mathrm{N}, 50)$;
if METHOD = 'BICGSTAB', $0 < \mathrm{M} \leq \min(\mathrm{N}, 10)$.

18: TOL – REAL (KIND=nag_wp) *Input*

On entry: the required tolerance. Let x_k denote the approximate solution at iteration k, and r_k the corresponding residual. The algorithm is considered to have converged at iteration k if

$$\|r_k\|_\infty \leq \tau \times \left(\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty\right).$$

If $\mathrm{TOL} \leq 0.0$, $\tau = \max \sqrt{\epsilon}, \sqrt{n}\epsilon$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max(\mathrm{TOL}, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: $\mathrm{TOL} < 1.0$.

19: MAXITN – INTEGER *Input*

On entry: the maximum number of iterations allowed.

Constraint: $\mathrm{MAXITN} \geq 1$.

20: X(N) – COMPLEX (KIND=nag_wp) array *Input/Output*

On entry: an initial approximation to the solution vector x.

On exit: an improved approximation to the solution vector x.

21: RNORM – REAL (KIND=nag_wp) *Output*

On exit: the final value of the residual norm $\|r_k\|_\infty$, where k is the output value of ITN.

22: ITN – INTEGER *Output*

On exit: the number of iterations carried out.

23: WORK(LWORK) – COMPLEX (KIND=nag_wp) array *Workspace*
24: LWORK – INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which F11DUF is called.

Constraints:

if METHOD = 'RGMRES', $\mathrm{LWORK} \geq 6 \times \mathrm{N} + \mathrm{M} \times (\mathrm{M} + \mathrm{N} + 5) + 121$;
if METHOD = 'CGS', $\mathrm{LWORK} \geq 10 \times \mathrm{N} + 120$;
if METHOD = 'BICGSTAB', $\mathrm{LWORK} \geq 2 \times \mathrm{N} \times \mathrm{M} + 8 \times \mathrm{N} + \mathrm{M} \times (\mathrm{M} + 2) + 120$;
if METHOD = 'TFQMR', $\mathrm{LWORK} \geq 13 \times \mathrm{N} + 120$.

25: 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, for $b = \langle value \rangle$, ISTB$(b + 1) = \langle value \rangle$ and ISTB$(b) = \langle value \rangle$.
Constraint: ISTB$(b + 1) >$ ISTB(b), for $b = 1, 2, \ldots,$ NB.

On entry, INDB$(\langle value \rangle) = \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: $1 \leq$ INDB$(m) \leq$ N, for $m = 1, 2, \ldots,$ ISTB(NB $+ 1) - 1$

On entry, ISTB$(1) = \langle value \rangle$.
Constraint: ISTB$(1) \geq 1$.

On entry, LA $= \langle value \rangle$ and NNZ $= \langle value \rangle$.
Constraint: LA $\geq 2 \times$ NNZ.

On entry, LINDB $= \langle value \rangle$, ISTB(NB $+ 1) - 1 = \langle value \rangle$ and NB $= \langle value \rangle$.
Constraint: LINDB $\geq$ ISTB(NB $+ 1) - 1$.

On entry, LWORK $= \langle value \rangle$.
Constraint: LWORK $\geq \langle value \rangle$.

On entry, M $= \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: if METHOD = 'RGMRES', $1 \leq$ M $\leq$ min(N, $\langle value \rangle$).
If METHOD = 'BICGSTAB', $1 \leq$ M $\leq$ min(N, $\langle value \rangle$).

On entry, MAXITN $= \langle value \rangle$.
Constraint: MAXITN ≥ 1.

On entry, METHOD $= \langle value \rangle$.
Constraint: METHOD = 'RGMRES', 'CGS' or 'BICGSTAB'.

On entry, N $= \langle value \rangle$.
Constraint: N ≥ 1.

On entry, NB $= \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: $1 \leq$ NB $\leq$ N.

On entry, NNZ $= \langle value \rangle$.
Constraint: NNZ ≥ 1.

On entry, NNZ $= \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: NNZ $\leq$ N^2.

On entry, TOL $= \langle value \rangle$.
Constraint: TOL < 1.0.

IFAIL $= 2$

On entry, element $\langle value \rangle$ of A was out of order.
Check that A, IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG have not been corrupted between calls to F11DTF and F11DUF.

On entry, ICOL$(\langle value \rangle) = \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: $1 \leq$ ICOL$(i) \leq$ N, for $i = 1, 2, \ldots,$ NNZ.
Check that A, IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG have not been corrupted between calls to F11DTF and F11DUF.

On entry, IROW$(\langle value \rangle) = \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: $1 \leq$ IROW$(i) \leq$ N, for $i = 1, 2, \ldots,$ NNZ.
Check that A, IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG have not been corrupted between calls to F11DTF and F11DUF.

On entry, location ⟨*value*⟩ of $(\mathrm{IROW}, \mathrm{ICOL})$ was a duplicate.
Check that A, IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG have not been corrupted between calls to F11DTF and F11DUF.

IFAIL $= 3$

The CS representation of the preconditioner is invalid.
Check that A, IROW, ICOL, IPIVP, IPIVQ, ISTR and IDIAG have not been corrupted between calls to F11DTF and F11DUF.

IFAIL $= 4$

The required accuracy could not be obtained. However, a reasonable accuracy may have been achieved. You should check the output value of RNORM for acceptability. This error code usually implies that your problem has been fully and satisfactorily solved to within or close to the accuracy available on your system. Further iterations are unlikely to improve on this situation.

IFAIL $= 5$

The solution has not converged after ⟨*value*⟩ iterations.

IFAIL $= 6$

Algorithmic breakdown. A solution is returned, although it is possible that it is completely inaccurate.

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

On successful termination, the final residual $r_k = b - Ax_k$, where $k = \mathrm{ITN}$, satisfies the termination criterion

$$\|r_k\|_\infty \leq \tau \times \left(\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty\right).$$

The value of the final residual norm is returned in RNORM.

8 Parallelism and Performance

F11DUF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

F11DUF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by F11DUF for each iteration is roughly proportional to the value of NNZC returned from the preceding call to F11DTF.

The number of iterations required to achieve a prescribed accuracy cannot be easily determined *a priori*, as it can depend dramatically on the conditioning and spectrum of the preconditioned coefficient matrix $\bar{A} = M^{-1}A$.

10 Example

This example program reads in a sparse matrix A and a vector b. It calls F11DTF, with the array LFILL $= 0$ and the array DTOL $= 0.0$, to compute an overlapping incomplete LU factorization. This is then used as an additive Schwarz preconditioner on a call to F11DUF which uses the RGMRES method to solve $Ax = b$.

10.1 Program Text

```
!   F11DUF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Program f11dufe

!     .. Use Statements ..
      Use nag_library, Only: f11dtf, f11duf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: dtolg, rnorm, tol
      Integer                          :: i, ifail, itn, k, la, lfillg, lindb, &
                                          liwork, lwork, m, maxitn, mb, n, nb, &
                                          nnz, nnzc, nover
      Character (8)                    :: method
      Character (1)                    :: milug, pstrag
!     .. Local Arrays ..
      Complex (Kind=nag_wp), Allocatable :: a(:), b(:), work(:), x(:)
      Real (Kind=nag_wp), Allocatable  :: dtol(:)
      Integer, Allocatable             :: icol(:), idiag(:), indb(:),          &
                                          ipivp(:), ipivq(:), irow(:),         &
                                          istb(:), istr(:), iwork(:),          &
                                          lfill(:), npivm(:)
      Character (1), Allocatable       :: milu(:), pstrat(:)
!     .. Executable Statements ..
      Continue

!     Print example header
      Write (nout,*) 'F11DUF Example Program Results'
      Write (nout,*)

!     Skip heading in data file
      Read (nin,*)

!     Get the square matrix size
      Read (nin,*) n

!     Allocate arrays with lengths based on mesh.
      liwork = 9*n + 3
      Allocate (b(n),x(n),iwork(liwork))

!     Get the number of non zero (nnz) matrix entries
      Read (nin,*) nnz
      la = 20*nnz
      Allocate (a(la),irow(la),icol(la))

      lindb = 3*n
      Allocate (idiag(lindb),indb(lindb),ipivp(lindb),ipivq(lindb), &
        istr(lindb+1))

!     Read in matrix A
      Read (nin,*)(a(i),irow(i),icol(i),i=1,nnz)

!     Read in RHS
      Read (nin,*) b(1:n)

!     Read algorithmic parameters
      Read (nin,*) method
      Read (nin,*) lfillg, dtolg
      Read (nin,*) pstrag
      Read (nin,*) milug
      Read (nin,*) m, tol, maxitn
      Read (nin,*) nb, nover

!     Allocate arrays with length based on number of blocks.
      Allocate (dtol(nb),istb(nb+1),lfill(nb),npivm(nb),milu(nb),pstrat(nb))

!     Set up initial approximate solution x
      x(1:n) = (0.0_nag_wp,0.0_nag_wp)

!     Define diagonal block indices.
!     In this example use blocks of MB consecutive rows and initialise
!     assuming no overlap.
      mb = (n+nb-1)/nb
      Do k = 1, nb
        istb(k) = (k-1)*mb + 1
      End Do
      istb(nb+1) = n + 1
      Do i = 1, n
        indb(i) = i
      End Do

!     Modify INDB and ISTB to account for overlap.
      Call f11dufe_overlap(n,nnz,la,irow,icol,nb,istb,indb,lindb,nover,iwork)
      If (iwork(1)==-999) Then
        Write (nout,*) '** LINDB too small, LINDB = ', lindb, '.'
        Go To 100
      End If

!     Set algorithmic parameters for each block from global values
      lfill(1:nb) = lfillg
      dtol(1:nb) = dtolg
      pstrat(1:nb) = pstrag
      milu(1:nb) = milug

!     Set size of real workspace
      lwork = 6*n + m*(m+n+5) + 121
      Allocate (work(lwork))

!     Calculate factorization
      ifail = 0
      Call f11dtf(n,nnz,a,la,irow,icol,nb,istb,indb,lindb,lfill,dtol,pstrat, &
        milu,ipivp,ipivq,istr,idiag,nnzc,npivm,iwork,liwork,ifail)

!     Solve Ax = b using F11DUF
      ifail = 0
      Call f11duf(method,n,nnz,a,la,irow,icol,nb,istb,indb,lindb,ipivp,ipivq, &
        istr,idiag,b,m,tol,maxitn,x,rnorm,itn,work,lwork,ifail)

      Write (nout,99999) itn
      Write (nout,99998) rnorm
      Write (nout,*)

!     Output x
      Write (nout,*) 'Solution vector   X'
      Write (nout,*) '-------------------'
      Write (nout,99997) x(1:n)
```

```
100   Continue

99999 Format (1X,' Converged in',I10,' iterations')
99998 Format (1X,' Final residual norm =',1P,D16.3)
99997 Format (1X,'(',F8.4,',',F8.4,')')

    Contains
      Subroutine f11dufe_overlap(n,nnz,la,irow,icol,nb,istb,indb,lindb,nover, &
        iwork)

!       Purpose
!       =======
!       This routine takes a set of row indices INDB defining the diagonal
!       blocks to be used in F11DTF to define a block Jacobi or additive Schwarz
!       preconditioner, and expands them to allow for NOVER levels of overlap.
!       The pointer array ISTB is also updated accordingly, so that the returned
!       values of ISTB and INDB can be passed to F11DTF to define overlapping
!       diagonal blocks.
!       ------------------------------------------------------------------------

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (In)             :: la, lindb, n, nb, nnz, nover
!       .. Array Arguments ..
        Integer, Intent (In)             :: icol(la), irow(la)
        Integer, Intent (Inout)          :: indb(lindb), istb(nb+1)
        Integer, Intent (Out)            :: iwork(3*n+1)
!       .. Local Scalars ..
        Integer                          :: i, ik, ind, iover, k, l, n21,      &
                                            nadd, row
!       .. Executable Statements ..
        Continue

!       Find the number of non-zero elements in each row of the matrix A, and
!       and start address of each row. Store the start addresses in
!       IWORK(N+1,...,2*N+1).

        iwork(1:n) = 0
        Do k = 1, nnz
          iwork(irow(k)) = iwork(irow(k)) + 1
        End Do
        iwork(n+1) = 1
        Do i = 1, n
          iwork(n+i+1) = iwork(n+i) + iwork(i)
        End Do

!       Loop over blocks.
blocks: Do k = 1, nb

!         Initialize marker array.
          iwork(1:n) = 0

!         Mark the rows already in block K in the workspace array.
          Do l = istb(k), istb(k+1) - 1
            iwork(indb(l)) = 1
          End Do

!         Loop over levels of overlap.
          Do iover = 1, nover

!           Initialize counter of new row indices to be added.
            ind = 0

!           Loop over the rows currently in the diagonal block.
            Do l = istb(k), istb(k+1) - 1
              row = indb(l)

!             Loop over non-zero elements in row ROW.
              Do i = iwork(n+row), iwork(n+row+1) - 1
```

```
!               If the column index of the non-zero element is not in the
!               existing set for this block, store it to be added later, and
!               mark it in the marker array.
                If (iwork(icol(i))==0) Then
                  iwork(icol(i)) = 1
                  ind = ind + 1
                  iwork(2*n+1+ind) = icol(i)
                End If
              End Do
            End Do

!           Shift the indices in INDB and add the new entries for block K.
!           Change ISTB accordingly.
            nadd = ind
            If (istb(nb+1)+nadd-1>lindb) Then
              iwork(1) = -999
              Exit blocks
            End If

            Do i = istb(nb+1) - 1, istb(k+1), -1
              indb(i+nadd) = indb(i)
            End Do
            n21 = 2*n + 1
            ik = istb(k+1) - 1
            indb(ik+1:ik+nadd) = iwork(n21+1:n21+nadd)
            istb(k+1:nb+1) = istb(k+1:nb+1) + nadd
          End Do
        End Do blocks

        Return

      End Subroutine f11dufe_overlap
    End Program f11dufe
```

10.2 Program Data

```
F11DUF Example Program Data
 9                                : n
 33     : nnz
 ( 96.0, -64.0)     1     1
 (-20.0,  22.0)     1     2
 (-36.0,  14.0)     1     4
 (-12.0,  10.0)     2     1
 ( 96.0, -64.0)     2     2
 (-20.0,  22.0)     2     3
 (-36.0,  14.0)     2     5
 (-12.0,  10.0)     3     2
 ( 96.0, -64.0)     3     3
 (-36.0,  14.0)     3     6
 (-28.0,  18.0)     4     1
 ( 96.0, -64.0)     4     4
 (-20.0,  22.0)     4     5
 (-36.0,  14.0)     4     7
 (-28.0,  18.0)     5     2
 (-12.0,  10.0)     5     4
 ( 96.0, -64.0)     5     5
 (-20.0,  22.0)     5     6
 (-36.0,  14.0)     5     8
 (-28.0,  18.0)     6     3
 (-12.0,  10.0)     6     5
 ( 96.0, -64.0)     6     6
 (-36.0,  14.0)     6     9
 (-28.0,  18.0)     7     4
 ( 96.0, -64.0)     7     7
 (-20.0,  22.0)     7     8
 (-28.0,  18.0)     8     5
 (-12.0,  10.0)     8     7
 ( 96.0, -64.0)     8     8
 (-20.0,  22.0)     8     9
 (-28.0,  18.0)     9     6
```

```
(-12.0,  10.0)     9     8
( 96.0, -64.0)     9     9    : a(i), irow(i), icol(i) for i=1,nnz
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)
(100.0, 4.0)    : b(i) for i=1,n
'RGMRES'                    : method
0   0.0D-1                  : lfillg, dtolg
'N'                         : pstrag
'N'                         : milug
2   1.0D-6  100             : m, tol, maxitn
3   1                       : nb, nover
```

10.3 Program Results

```
F11DUF Example Program Results

 Converged in         8 iterations
 Final residual norm =       6.492D-04

Solution vector   X
-------------------
(  2.2040,  1.6972)
(  2.3511,  1.9275)
(  1.5931,  1.4368)
(  2.8641,  1.9762)
(  3.0687,  2.2645)
(  2.0467,  1.6948)
(  2.2065,  1.3244)
(  2.3724,  1.5170)
(  1.6254,  1.1783)
```