NAG Library Routine Document

F02WGF

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

F02WGF returns leading terms in the singular value decomposition (SVD) of a real general matrix and computes the corresponding left and right singular vectors.

2 Specification

```
SUBROUTINE F02WGF (M, N, K, NCV, AV, NCONV, SIGMA, U, LDU, V, LDV, RESID,      &
                   IUSER, RUSER, IFAIL)

INTEGER            M, N, K, NCV, NCONV, LDU, LDV, IUSER(*), IFAIL
REAL (KIND=nag_wp) SIGMA(NCV), U(LDU,NCV), V(LDV,NCV), RESID(NCV),             &
                   RUSER(*)
EXTERNAL           AV
```

3 Description

F02WGF computes a few, k, of the largest singular values and corresponding vectors of an m by n matrix A. The value of k should be small relative to m and n, for example $k \sim O(\min(m,n))$. The full singular value decomposition (SVD) of an m by n matrix A is given by

$$A = U\Sigma V^{\mathrm{T}},$$

where U and V are orthogonal and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the *singular values* of A and the first $\min(m,n)$ columns of U and V are the *left* and *right singular vectors* of A.

If U_k, V_k denote the leading k columns of U and V respectively, and if Σ_k denotes the leading principal submatrix of Σ, then

$$A_k \equiv U_k \Sigma_k {V^{\mathrm{T}}}_k$$

is the best rank-k approximation to A in both the 2-norm and the Frobenius norm.

The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \qquad \text{and} \qquad A^{\mathrm{T}}u_i = \sigma_i v_i \qquad \text{so that} \qquad A^{\mathrm{T}}A\nu_i = \sigma_i^2 \nu_i \text{ and } AA^{\mathrm{T}}u_i = \sigma_i^2 u_i,$$

where u_i and v_i are the ith columns of U_k and V_k respectively.

Thus, for $m \geq n$, the largest singular values and corresponding right singular vectors are computed by finding eigenvalues and eigenvectors for the symmetric matrix $A^{\mathrm{T}}A$. For $m < n$, the largest singular values and corresponding left singular vectors are computed by finding eigenvalues and eigenvectors for the symmetric matrix AA^{T}. These eigenvalues and eigenvectors are found using routines from Chapter F12. You should read the F12 Chapter Introduction for full details of the method used here.

The real matrix A is not explicitly supplied to F02WGF. Instead, you are required to supply a routine, AV, that must calculate one of the requested matrix-vector products Ax or $A^{\mathrm{T}}x$ for a given real vector x (of length n or m respectively).

4 References

Wilkinson J H (1978) Singular Value Decomposition – Basic Aspects *Numerical Software – Needs and Availability* (ed D A H Jacobs) Academic Press

5 Parameters

1: M – INTEGER *Input*

On entry: m, the number of rows of the matrix A.

Constraint: $\mathrm{M} \geq 0$.

If $\mathrm{M} = 0$, an immediate return is effected.

2: N – INTEGER *Input*

On entry: n, the number of columns of the matrix A.

Constraint: $\mathrm{N} \geq 0$.

If $\mathrm{N} = 0$, an immediate return is effected.

3: K – INTEGER *Input*

On entry: k, the number of singular values to be computed.

Constraint: $0 < \mathrm{K} < \min(\mathrm{M}, \mathrm{N}) - 1$.

4: NCV – INTEGER *Input*

On entry: the dimension of the arrays SIGMA and RESID and the second dimension of the arrays U and V as declared in the (sub)program from which F02WGF is called. This is the number of Lanczos basis vectors to use during the computation of the largest eigenvalues of $A^{\mathrm{T}}A$ ($m \geq n$) or AA^{T} ($m < n$).

At present there is no *a priori* analysis to guide the selection of NCV relative to K. However, it is recommended that $\mathrm{NCV} \geq 2 \times \mathrm{K} + 1$. If many problems of the same type are to be solved, you should experiment with varying NCV while keeping K fixed for a given test problem. This will usually decrease the required number of matrix-vector operations but it also increases the internal storage required to maintain the orthogonal basis vectors. The optimal 'cross-over' with respect to CPU time is problem dependent and must be determined empirically.

Constraint: $\mathrm{K} < \mathrm{NCV} \leq \min(\mathrm{M}, \mathrm{N})$.

5: AV – SUBROUTINE, supplied by the user. *External Procedure*

AV must return the vector result of the matrix-vector product Ax or $A^{\mathrm{T}}x$, as indicated by the input value of IFLAG, for the given vector x.

AV is called from F02WGF with the parameter IUSER and RUSER as supplied to F02WGF. You are free to use these arrays to supply information to AV.

The specification of AV is:

```
SUBROUTINE AV (IFLAG, M, N, X, AX, IUSER, RUSER)

INTEGER            IFLAG, M, N, IUSER(*)
REAL (KIND=nag_wp) X(*), AX(*), RUSER(*)
```

1: IFLAG – INTEGER *Input/Output*

On entry: if $\mathrm{IFLAG} = 1$, AX must return the m-vector result of the matrix-vector product Ax.

If $\mathrm{IFLAG} = 2$, AX must return the n-vector result of the matrix-vector product $A^{\mathrm{T}}x$.

On exit: may be used as a flag to indicate a failure in the computation of Ax or $A^{\mathrm{T}}x$. If IFLAG is negative on exit from AV, F02WGF will exit immediately with IFAIL set to IFLAG.

2: M – INTEGER *Input*

On entry: the number of rows of the matrix A.

3: N – INTEGER *Input*

On entry: the number of columns of the matrix A.

4: X(∗) – REAL (KIND=nag_wp) array *Input*

On entry: the vector to be pre-multiplied by the matrix A or A^{T}.

5: AX(∗) – REAL (KIND=nag_wp) array *Output*

On exit: if IFLAG $= 1$, contains the m-vector result of the matrix-vector product Ax.

If IFLAG $= 2$, contains the n-vector result of the matrix-vector product $A^{\mathrm{T}}x$.

6: IUSER(∗) – INTEGER array *User Workspace*
7: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

AV is called with the parameters IUSER and RUSER as supplied to F02WGF. You are free to use the arrays IUSER and RUSER to supply information to AV as an alternative to using COMMON global variables.

AV must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which F02WGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: NCONV – INTEGER *Output*

On exit: the number of converged singular values found.

7: SIGMA(NCV) – REAL (KIND=nag_wp) array *Output*

On exit: the NCONV converged singular values are stored in the first NCONV elements of SIGMA.

8: U(LDU,NCV) – REAL (KIND=nag_wp) array *Output*

On exit: the left singular vectors corresponding to the singular values stored in SIGMA.

The ith element of the jth left singular vector u_j is stored in U(i,j), for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots,$ NCONV.

9: LDU – INTEGER *Input*

On entry: the first dimension of the array U as declared in the (sub)program from which F02WGF is called.

Constraint: LDU $\geq \max(1,$ M$)$.

10: V(LDV,NCV) – REAL (KIND=nag_wp) array *Output*

On exit: the right singular vectors corresponding to the singular values stored in SIGMA.

The ith element of the jth right singular vector v_j is stored in V(i,j), for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots,$ NCONV.

11: LDV – INTEGER *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which F02WGF is called.

Constraint: LDV $\geq \max(1, \mathrm{N})$.

12: RESID(NCV) – REAL (KIND=nag_wp) array *Output*

On exit: the residual $\left\|Av_j - \sigma_j u_j\right\|$, for $m \geq n$, or $\left\|A^{\mathrm{T}}u_j - \sigma_j v_j\right\|$, for $m < n$, for each of the converged singular values σ_j and corresponding left and right singular vectors u_j and v_j.

13: IUSER($*$) – INTEGER array *User Workspace*
14: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by F02WGF, but are passed directly to AV and may be used to pass information to this routine as an alternative to using COMMON global variables.

15: 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).

F02WGF returns with IFAIL $= 0$ if at least k singular values have converged and the corresponding left and right singular vectors have been computed.

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 < 0

IFLAG was set to a negative value following a call to AV.

IFAIL $= 1$

On entry, M < 0.

IFAIL $= 2$

On entry, N < 0.

IFAIL $= 3$

On entry, K ≤ 0.

IFAIL $= 4$

On entry, NCV $\leq$ K $+ 1$,
or NCV $> \min(\mathrm{M}, \mathrm{N})$.

IFAIL $= 5$

On entry, LDU $<$ M.

IFAIL = 6

On entry, LDV < N.

IFAIL = 7

No converged singular values were found to sufficient accuracy.

IFAIL = 8

The internal maximum number of iterations has been reached. Some singular values may have converged.

IFAIL = 9
IFAIL = 10
IFAIL = 20

An error occurred during an internal call. Consider increasing the size of NCV relative to K.

7 Accuracy

See Section 2.14.2 in the F08 Chapter Introduction.

8 Further Comments

None.

9 Example

This example finds the four largest singular values (σ) and corresponding right and left singular vectors for the matrix A, where A is the m by n real matrix derived from the simplest finite difference discretization of the two-dimensional kernal $k(s,t)dt$ where

$$k(s,t) = \begin{cases} s(t-1) & \text{if } 0 \leq s \leq t \leq 1 \\ t(s-1) & \text{if } 0 \leq t < s \leq 1 \end{cases}.$$

9.1 Program Text

```
!   F02WGF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module f02wgfe_mod

!     F02WGF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                  :: nin = 5, nout = 6
    Contains
!     Matrix vector subroutines

      Subroutine av(iflag,m,n,x,ax,iuser,ruser)

!       Computes  w <- A*x or w <- Trans(A)*x.

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter        :: one = 1.0_nag_wp
        Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
!       .. Scalar Arguments ..
        Integer, Intent (Inout)              :: iflag
        Integer, Intent (In)                 :: m, n
```

```
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: ax(*), ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: x(*)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: h, k, s, t
        Integer                              :: i, j
!       .. Intrinsic Procedures ..
        Intrinsic                            :: min, real
!       .. Executable Statements ..
        h = one/real(m+1,kind=nag_wp)
        k = one/real(n+1,kind=nag_wp)
        If (iflag==1) Then
          ax(1:m) = zero
          t = zero

          Do j = 1, n
            t = t + k
            s = zero
            Do i = 1, min(j,m)
              s = s + h
              ax(i) = ax(i) + k*s*(t-one)*x(j)
            End Do
            Do i = j + 1, m
              s = s + h
              ax(i) = ax(i) + k*t*(s-one)*x(j)
            End Do
          End Do
        Else
          ax(1:n) = zero
          t = zero

          Do j = 1, n
            t = t + k
            s = zero
            Do i = 1, min(j,m)
              s = s + h
              ax(j) = ax(j) + k*s*(t-one)*x(i)
            End Do
            Do i = j + 1, m
              s = s + h
              ax(j) = ax(j) + k*t*(s-one)*x(i)
            End Do
          End Do
        End If

        Return
      End Subroutine av
    End Module f02wgfe_mod
    Program f02wgfe

!     F02WGF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: f02wgf, nag_wp
      Use f02wgfe_mod, Only: av, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Integer                              :: i, ifail, k, ldu, ldv, m, n,     &
                                              nconv, ncv
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: resid(:), sigma(:), u(:,:), v(:,:)
      Real (Kind=nag_wp)                   :: ruser(1)
      Integer                              :: iuser(1)
!     .. Executable Statements ..
      Write (nout,*) 'F02WGF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) m, n, k, ncv
```

```
      ldu = m
      ldv = n
      Allocate (resid(ncv),sigma(ncv),u(ldu,ncv),v(ldv,ncv))

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call f02wgf(m,n,k,ncv,av,nconv,sigma,u,ldu,v,ldv,resid,iuser,ruser, &
        ifail)

!     Print computed residuals
      Write (nout,*) '  Singular Value    Residual'
      Write (nout,99999)(sigma(i),resid(i),i=1,nconv)

99999 Format (1X,F10.5,8X,G10.2)
    End Program f02wgfe
```

9.2 Program Data

```
F02WGF Example Program Data
 100 500 4 10         : m, n, k, ncv
```

9.3 Program Results

```
 F02WGF Example Program Results

   Singular Value    Residual
    0.00830          0.27E-18
    0.01223          0.59E-17
    0.02381          0.12E-16
    0.11274          0.78E-16
```