NAG Library Routine Document

C05QSF

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

C05QSF is an easy-to-use routine that finds a solution of a sparse system of nonlinear equations by a modification of the Powell hybrid method.

2 Specification

```
SUBROUTINE C05QSF (FCN, N, X, FVEC, XTOL, INIT, RCOMM, LRCOMM, ICOMM,      &
                   LICOMM, IUSER, RUSER, IFAIL)

INTEGER            N, LRCOMM, ICOMM(LICOMM), LICOMM, IUSER(*), IFAIL
REAL (KIND=nag_wp) X(N), FVEC(N), XTOL, RCOMM(LRCOMM), RUSER(*)
LOGICAL            INIT
EXTERNAL           FCN
```

3 Description

The system of equations is defined as:

$$f_i(x_1, x_2, \ldots, x_n) = 0, \qquad i = 1, 2, \ldots, n.$$

C05QSF is based on the MINPACK routine HYBRD1 (see Moré *et al.* (1980)). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. The Jacobian is updated by the sparse rank-1 method of Schubert (see Schubert (1970)). At the starting point, the sparsity pattern is determined and the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. Then, the sparsity structure is used to recompute an approximation to the Jacobian by forward differences with the least number of function evaluations. The subroutine you supply must be able to compute only the requested subset of the function values. The sparse Jacobian linear system is solved at each iteration with F11MEF computing the Newton step. For more details see Powell (1970) and Broyden (1965).

4 References

Broyden C G (1965) A class of methods for solving nonlinear simultaneous equations *Mathematics of Computation* **19(92)** 577–593

Moré J J, Garbow B S and Hillstrom K E (1980) User guide for MINPACK-1 *Technical Report ANL-80-74* Argonne National Laboratory

Powell M J D (1970) A hybrid method for nonlinear algebraic equations *Numerical Methods for Nonlinear Algebraic Equations* (ed P Rabinowitz) Gordon and Breach

Schubert L K (1970) Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian *Mathematics of Computation* **24(109)** 27–30

5 Arguments

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must return the values of the functions f_i at a point x.

The specification of FCN is:

```
SUBROUTINE FCN (N, LINDF, INDF, X, FVEC, IUSER, RUSER, IFLAG)
INTEGER            N, LINDF, INDF(LINDF), IUSER(*), IFLAG
REAL (KIND=nag_wp) X(N), FVEC(N), RUSER(*)
```

1: N – INTEGER *Input*

On entry: n, the number of equations.

2: LINDF – INTEGER *Input*

On entry: LINDF specifies the number of indices i for which values of $f_i(x)$ must be computed.

3: INDF(LINDF) – INTEGER array *Input*

On entry: INDF specifies the indices i for which values of $f_i(x)$ must be computed. The indices are specified in strictly ascending order.

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

On entry: the components of the point x at which the functions must be evaluated. X(i) contains the coordinate x_i.

5: FVEC(N) – REAL (KIND=nag_wp) array *Output*

On exit: FVEC(i) must contain the function values $f_i(x)$, for all indices i in INDF.

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

FCN is called with the arguments IUSER and RUSER as supplied to C05QSF. You should use the arrays IUSER and RUSER to supply information to FCN.

8: IFLAG – INTEGER *Input/Output*

On entry: IFLAG > 0.

On exit: in general, IFLAG should not be reset by FCN. If, however, you wish to terminate execution (perhaps because some illegal point X has been reached), then IFLAG should be set to a negative integer.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which C05QSF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER *Input*

On entry: n, the number of equations.

Constraint: N > 0.

3: X(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: an initial guess at the solution vector. X(i) must contain the coordinate x_i.

On exit: the final estimate of the solution vector.

4: FVEC(N) – REAL (KIND=nag_wp) array *Output*

On exit: the function values at the final point returned in X. FVEC(i) contains the function values f_i.

5: XTOL – REAL (KIND=nag_wp) *Input*

On entry: the accuracy in X to which the solution is required.

Suggested value: $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** returned by X02AJF.

Constraint: XTOL ≥ 0.0.

6: INIT – LOGICAL *Input*

On entry: INIT must be set to .TRUE. to indicate that this is the first time C05QSF is called for this specific problem. C05QSF then computes the dense Jacobian and detects and stores its sparsity pattern (in RCOMM and ICOMM) before proceeding with the iterations. This is noticeably time consuming when N is large. If not enough storage has been provided for RCOMM or ICOMM, C05QSF will fail. On exit with IFAIL $= 0$, 2, 3 or 4, ICOMM(1) contains nnz, the number of nonzero entries found in the Jacobian. On subsequent calls, INIT can be set to .FALSE. if the problem has a Jacobian of the same sparsity pattern. In that case, the computation time required for the detection of the sparsity pattern will be smaller.

7: RCOMM(LRCOMM) – REAL (KIND=nag_wp) array *Communication Array*

RCOMM **must not** be altered between successive calls to C05QSF.

8: LRCOMM – INTEGER *Input*

On entry: the dimension of the array RCOMM as declared in the (sub)program from which C05QSF is called.

Constraint: LRCOMM $\geq 12 + nnz$ where nnz is the number of nonzero entries in the Jacobian, as computed by C05QSF.

9: ICOMM(LICOMM) – INTEGER array *Communication Array*

If IFAIL $= 0$, 2, 3 or 4 on exit, ICOMM(1) contains nnz where nnz is the number of nonzero entries in the Jacobian.

ICOMM **must not** be altered between successive calls to C05QSF.

10: LICOMM – INTEGER *Input*

On entry: the dimension of the array ICOMM as declared in the (sub)program from which C05QSF is called.

Constraint: LICOMM $\geq 8 \times \mathrm{N} + 19 + nnz$ where nnz is the number of nonzero entries in the Jacobian, as computed by C05QSF.

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

IUSER and RUSER are not used by C05QSF, but are passed directly to FCN and should be used to pass information to this routine.

13: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation 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 argument, 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 $= 2$

There have been at least $200 \times (\mathrm{N} + 1)$ calls to FCN. Consider setting INIT = .FALSE. and restarting the calculation from the point held in X.

IFAIL $= 3$

No further improvement in the solution is possible. XTOL is too small: XTOL $= \langle value \rangle$.

IFAIL $= 4$

The iteration is not making good progress. This failure exit may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05QSF from a different starting point may avoid the region of difficulty. The condition number of the Jacobian is $\langle value \rangle$.

IFAIL $= 5$

IFLAG was set negative in FCN. IFLAG $= \langle value \rangle$.

IFAIL $= 6$

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

IFAIL $= 7$

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

IFAIL $= 9$

An internal error has occurred. Code $= \langle value \rangle$.

IFAIL $= 11$

On entry, N $= \langle value \rangle$.
Constraint: N > 0.

IFAIL $= 12$

On entry, XTOL $= \langle value \rangle$.
Constraint: XTOL ≥ 0.0.

IFAIL $= -99$

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

If $\hat{x}$ is the true solution, C05QSF tries to ensure that

$$\|x - \hat{x}\|_2 \leq \text{XTOL} \times \|\hat{x}\|_2.$$

If this condition is satisfied with XTOL $= 10^{-k}$, then the larger components of x have k significant decimal digits. There is a danger that the smaller components of x may have large relative errors, but the fast rate of convergence of C05QSF usually obviates this possibility.

If XTOL is less than ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL $= 3$.

Note: this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The convergence test assumes that the functions are reasonably well behaved. If this condition is not satisfied, then C05QSF may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning C05QSF with a lower value for XTOL.

8 Parallelism and Performance

C05QSF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

C05QSF 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

Local workspace arrays of fixed lengths are allocated internally by C05QSF. The total size of these arrays amounts to $8 \times n + 2 \times q$ real elements and $10 \times n + 2 \times q + 5$ integer elements where the integer q is bounded by $8 \times nnz$ and n^2 and depends on the sparsity pattern of the Jacobian.

The time required by C05QSF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05QSF to process each evaluation of the functions depends on the number of nonzero entries in the Jacobian. The timing of C05QSF is strongly influenced by the time spent evaluating the functions.

When INIT is .TRUE., the dense Jacobian is first evaluated and that will take time proportional to n^2.

Ideally the problem should be scaled so that, at the solution, the function values are of comparable magnitude.

10 Example

This example determines the values $x_1, \ldots, x_9$ which satisfy the tridiagonal equations:

$$\begin{aligned} (3 - 2x_1)x_1 - 2x_2 &= -1, \\ -x_{i-1} + (3 - 2x_i)x_i - 2x_{i+1} &= -1, \quad i = 2, 3, \ldots, 8 \\ -x_8 + (3 - 2x_9)x_9 &= -1. \end{aligned}$$

It then perturbs the equations by a small amount and solves the new system.

10.1 Program Text

```
!   C05QSF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module c05qsfe_mod

!     C05QSF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: fcn
!     .. Parameters ..
      Integer, Parameter, Public       :: n = 9, nout = 6
    Contains
      Subroutine fcn(n,lindf,indf,x,fvec,iuser,ruser,iflag)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: one = 1.0E0_nag_wp
        Real (Kind=nag_wp), Parameter  :: three = 3.0E0_nag_wp
        Real (Kind=nag_wp), Parameter  :: two = 2.0E0_nag_wp
        Real (Kind=nag_wp), Parameter  :: alpha = (one/two)**7
!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: iflag
        Integer, Intent (In)           :: lindf, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: fvec(n)
        Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
        Real (Kind=nag_wp), Intent (In) :: x(n)
        Integer, Intent (In)           :: indf(lindf)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: theta
        Integer                        :: i, ind
!       .. Intrinsic Procedures ..
        Intrinsic                      :: real
!       .. Executable Statements ..
        iflag = 0
        theta = real(iuser(1),kind=nag_wp)*alpha
        Do ind = 1, lindf
          i = indf(ind)
          fvec(i) = (three-(two+theta)*x(i))*x(i) + one
          If (i>1) Then
            fvec(i) = fvec(i) - x(i-1)
          End If
          If (i<n) Then
            fvec(i) = fvec(i) - two*x(i+1)
          End If
        End Do
        Return

      End Subroutine fcn
    End Module c05qsfe_mod
    Program c05qsfe

!     .. Use Statements ..
      Use nag_library, Only: c05qsf, dnrm2, nag_wp, x02ajf
      Use c05qsfe_mod, Only: fcn, n, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: fnorm, xtol
      Integer                          :: i, ifail, j, licomm, lrcomm
      Logical                          :: init
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: fvec(:), rcomm(:), x(:)
```

```
      Real (Kind=nag_wp)                 :: ruser(1)
      Integer, Allocatable               :: icomm(:)
      Integer                            :: iuser(1)
!     .. Intrinsic Procedures ..
      Intrinsic                          :: sqrt
!     .. Executable Statements ..
      Write (nout,*) 'C05QSF Example Program Results'

      xtol = sqrt(x02ajf())
      lrcomm = 12 + 3*n
      licomm = 8*n + 19 + 3*n

      Allocate (fvec(n),x(n),rcomm(lrcomm),icomm(licomm))

!     The following starting values provide a rough solution.
      x(1:n) = -1.0E0_nag_wp

      Do i = 0, 1
        ifail = -1

!       Perturb the system?
        iuser(1) = i

        init = (i==0)
        Call c05qsf(fcn,n,x,fvec,xtol,init,rcomm,lrcomm,icomm,licomm,iuser,    &
          ruser,ifail)

        Select Case (ifail)
        Case (0)
!         The NAG name equivalent of dnrm2 is f06ejf
          fnorm = dnrm2(n,fvec,1)
          Write (nout,*)
          Write (nout,99999) 'Final 2-norm of the residuals =', fnorm
          Write (nout,*)
          Write (nout,*) 'Final approximate solution'
          Write (nout,*)
          Write (nout,99998)(x(j),j=1,n)
        Case (2:4)
          Write (nout,*)
          Write (nout,*) 'Approximate solution'
          Write (nout,*)
          Write (nout,99998)(x(j),j=1,n)
        End Select
      End Do

99999 Format (1X,A,E12.4)
99998 Format (1X,3F12.4)
    End Program c05qsfe
```

10.2 Program Data

None.

10.3 Program Results

```
 C05QSF Example Program Results

 Final 2-norm of the residuals =  0.1759E-08

 Final approximate solution

      -0.5707     -0.6816     -0.7017
      -0.7042     -0.7014     -0.6919
      -0.6658     -0.5960     -0.4164

 Final 2-norm of the residuals =  0.2633E-12
```

```
 Final approximate solution

     -0.5697     -0.6804     -0.7004
     -0.7029     -0.7000     -0.6906
     -0.6646     -0.5951     -0.4159
```