NAG C Library Function Document

nag_zero_nonlin_eqns_deriv_1 (c05ubc)

1 Purpose

nag_zero_nonlin_eqns_deriv_1 (c05ubc) finds a solution of a system of nonlinear equations by a modification of the Powell hybrid method. The user must provide the Jacobian.

2 Specification

```
void nag_zero_nonlin_eqns_deriv_1 (Integer n, double x[], double fvec[], double
     fjac[], Integer tdfjac,
     void (*f)(Integer n, const double x[], double fvec[], double fjac[], Integer
          tdfjac, Integer *userflag, Nag_User *comm),
     double xtol, Nag_User *comm, NagError *fail)
```

3 Description

The system of equations is defined as:

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

nag_zero_nonlin_eqns_deriv_1 (c05ubc) is based upon the MINPACK routine HYBRJ1 (Moré *et al.* (1980)). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is calculated, but it is not recalculated until the rank-1 method fails to produce satisfactory progress. For more details see Powell (1970).

4 References

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

5 Parameters

1: **n** – Integer *Input*

On entry: the number of equations, n.

Constraint: **n** > 0.

2: **x**[**n**] – double *Input/Output*

On entry: an initial guess at the solution vector.

On exit: the final estimate of the solution vector.

3: **fvec**[**n**] – double *Output*

On exit: the function values at the final point, **x**.

4: **fjac**[**n**][**tdfjac**] – double *Output*

On exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

5: **tdfjac** – Integer *Input*

On entry: the second dimension of the array **fjac** as declared in the subroutine from which nag_zero_nonlin_eqns_deriv_1 (c05ubc) is called.

Constraint: **tdfjac** $\geq$ **n**.

6: **f** – function, supplied by the user *Function*

Depending upon the value of **userflag**, **f** must either return the values of the functions f_i at a point x or return the Jacobian at x.

Its specification is:

```
void f (Integer n, const double x[], double fvec[], double fjac[], Integer tdfjac,
      Integer *userflag, Nag_User *comm)
```

1: **n** – Integer *Input*

On entry: the number of equations, n

2: **x**[**n**] – const double *Input*

On entry: the components of the point x at which the functions or the Jacobian must be evaluated.

3: **fvec**[**n**] – double *Output*

On exit: if **userflag** $= 1$ on entry, **fvec** must contain the function values $f_i(x)$ (unless **userflag** is set to a negative value by **f**). If **userflag** $= 2$ on entry, **fvec** must not be changed.

4: **fjac**[**n** $\times$ **tdfjac**] – double *Output*

On exit: if **userflag** $= 2$ on entry, **fjac**$[(i-1) \times$ **tdfjac** $+ j - 1]$ must contain the value of $\partial f_i / \partial x_j$ at the point x, for $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, n$ (unless **userflag** is set to a negative value by **f**). If **userflag** $= 1$ on entry, **fjac** must not be changed.

5: **tdfjac** – Integer *Input*

On entry: the second dimension of the array **fjac** as declared in the subroutine from which nag_zero_nonlin_eqns_deriv_1 (c05ubc) is called.

6: **userflag** – Integer * *Input/Output*

On entry: **userflag** $= 1$ or 2.

If **userflag** $= 1$, **fvec** is to be updated.

If **userflag** $= 2$, **fjac** is to be updated.

On exit: in general, **userflag** should not be reset by **f**. If, however, the user wishes to terminate execution (perhaps because some illegal point **x** has been reached), then **userflag** should be set to a negative integer. This value will be returned through **fail.errnum**.

7: **comm** – Nag_User * *Input/Output*

Pointer to a structure of type **Nag_User** with the following member:

p – Pointer * *Input/Output*

On entry/on exit: the pointer **comm** $\rightarrow$ **p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm→p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

7: **xtol** – double *Input*

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

Suggested value: the square root of the ***machine precision***.

Constraint: **xtol** ≥ 0.0.

8: **comm** – Nag_User * *Input/Output*

Pointer to a structure of type **Nag_User** with the following member:

p – Pointer * *Input/Output*

On entry/on exit: the pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined function **f**(). An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program, e.g. `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

9: **fail** – NagError * *Input/Output*

The NAG error parameter, see the Essential Introduction.

6 Error Indicators and Warnings

NE_INT_ARG_LE

On entry, **n** must not be less than or equal to 0: **n** $= \langle value \rangle$.

NE_REAL_ARG_LT

On entry, **xtol** must not be less than 0.0: **xtol** $= \langle value \rangle$.

NE_2_INT_ARG_LT

On entry **tdfjac** $= \langle value \rangle$ while **n** $= \langle value \rangle$. These parameters must satisfy **tdfjac** $\geq$ **n**.

NE_ALLOC_FAIL

Memory allocation failed.

NE_USER_STOP

User requested termination, user flag value $= \langle value \rangle$.

NE_TOO_MANY_FUNC_EVAL

There have been at least $100 \times (\mathbf{n} + 1)$ evaluations of **f**().

Consider restarting the calculation from the point held in **x**.

NE_XTOL_TOO_SMALL

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

NE_NO_IMPROVEMENT

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 8). Otherwise, rerunning nag_zero_nonlin_eqns_deriv_1 (c05ubc) from a different starting point may avoid the region of difficulty.

7 Accuracy

If $\hat{x}$ is the true solution, nag_zero_nonlin_eqns_deriv_1 (c05ubc) tries to ensure that

$$\|x - \hat{x}\| \leq \mathbf{xtol} \times \|\hat{x}\|.$$

If this condition is satisfied with $\mathbf{xtol} = 10^{-k}$, 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 nag_zero_nonlin_eqns_deriv_1 (c05ubc) usually avoids the possibility.

If **xtol** is less than ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of **xtol**, then the function exits with **NE_XTOL_TOO_SMALL**.

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 test assumes that the functions and Jacobian are coded consistently and that the functions are reasonably well behaved. If these conditions are not satisfied then nag_zero_nonlin_eqns_deriv_1 (c05ubc) may incorrectly indicate convergence. The coding of the Jacobian can be checked using nag_check_deriv_1 (c05zcc). If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning nag_zero_nonlin_eqns_deriv_1 (c05ubc) with a tighter tolerance.

8 Further Comments

The time required by nag_zero_nonlin_eqns_deriv_1 (c05ubc) 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 nag_zero_nonlin_eqns_deriv_1 (c05ubc) is about $11.5 \times n^2$ to process each evaluation of the functions and about $1.3 \times n^3$ to process each evaluation of the Jacobian. Unless **f** can be evaluated quickly, the timing of nag_zero_nonlin_eqns_deriv_1 (c05ubc) will be strongly influenced by the time spent in **f**.

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

9 Example

To determine the values $x_1, \ldots, x_9$ which satisfy the tridiagonal equations:

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

9.1 Program Text

```
/* nag_zero_nonlin_eqns_deriv_1(c05ubc) Example Program
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * Mark 7 revised, 2001.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagc05.h>
#include <nagx02.h>

#ifdef __cplusplus
extern "C" {
#endif
static void f(Integer n, double x[], double fvec[], double fjac[],
              Integer tdfjac, Integer *userflag, Nag_User *comm);
#ifdef __cplusplus
```

```
}
#endif

#define NMAX 9
#define TDFJAC NMAX

int main(void)
{

  double fjac[NMAX*NMAX], fvec[NMAX], x[NMAX];
  Integer j;
  double xtol;
  static NagError fail;
  Nag_User comm;
  Integer n = NMAX;

  Vprintf("c05ubc Example Program Results\n");
  /* The following starting values provide a rough solution. */
  for (j=0; j<n; j++)
    x[j] = -1.0;
  xtol = sqrt(X02AJC);
  c05ubc(n, x, fvec, fjac, (Integer)TDFJAC, f, xtol, &comm, &fail);
  if (fail.code == NE_NOERROR)
    {
      Vprintf("Final approximate solution\n\n");
      for (j=0; j<n; j++)
        Vprintf("%12.4f%s",x[j], (j%3==2 || j==n-1) ? "\n" : " " );
      return EXIT_SUCCESS;
    }
  else
    {
      Vprintf("%s\n", fail.message);
      if (fail.code == NE_TOO_MANY_FUNC_EVAL ||
          fail.code == NE_XTOL_TOO_SMALL ||
          fail.code ==  NE_NO_IMPROVEMENT)
        {
          Vprintf("Approximate solution\n\n");
          for (j=0; j<n; j++)
            Vprintf("%12.4f%s",x[j], (j%3==2 || j==n-1) ? "\n" : " " );
        }
      return EXIT_FAILURE;
    }
}

static void f(Integer n, double x[], double fvec[], double fjac[],
              Integer tdfjac, Integer *userflag, Nag_User *comm)
{
#define FJAC(I,J) fjac[((I))*tdfjac+(J)]
  Integer j, k;

  if (*userflag != 2)
    {
      for (k=0; k<n; k++)
        {
          fvec[k] = (3.0-x[k]*2.0) * x[k] + 1.0;
          if (k>0)  fvec[k] -= x[k-1];
          if (k<n-1) fvec[k] -= x[k+1] * 2.0;
        }
    }
  else
    {
      for (k=0; k<n; k++)
        {
          for (j=0; j<n; j++)
            FJAC(k,j)=0.0;
          FJAC(k,k) = 3.0 - x[k] * 4.0;
          if (k>0)
            FJAC(k,k-1) = -1.0;
          if (k<n-1)
            FJAC(k,k+1)= -2.0;
        }
    }
}
```

9.2 Program Data

9.3 Program Results

```
c05ubc Example Program Results
Final approximate solution

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