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

```c
#include <nag.h>
#include <nagc05.h>

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

3. Description

The system of equations is defined as:

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

nag_zero_nonlin_eqns_deriv_1 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. Parameters

- **n**
  - Input: the number of equations, \( n \).
  - Constraint: \( n > 0 \).

- **x[n]**
  - Input: an initial guess at the solution vector.
  - Output: the final estimate of the solution vector.

- **fvec[n]**
  - Output: the function values at the final point, \( x \).

- **fjac[n][tdfjac]**
  - Output: the orthogonal matrix \( Q \) produced by the QR factorization of the final approximate Jacobian.

- **tdfjac**
  - Input: the last dimension of array \( fjac \) as declared in the function from which nag_zero_nonlin_eqns_deriv_1 is called.
  - Constraint: \( tdfjac \geq n \).

- **f**
  - 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 \( x \).
The specification of \( f \) is:

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

- **n**: Input: the number of equations, \( n \)
- **x[n]**: Input: the components of the point \( x \) at which the functions or the Jacobian must be evaluated.
- **fvec[n]**: Output: if \( \text{userflag} = 1 \) on entry, \( fvec \) must contain the function values \( f_i(x) \) (unless \( \text{userflag} \) is set to a negative value by \( f \)). If \( \text{userflag} = 2 \) on entry, \( fvec \) must not be changed.
- **fjac[n \ast tdfjac]**: Output: if \( \text{userflag} = 2 \) on entry, \( fjac[(i-1)\ast tdfjac+j-1] \) must contain the value of \( \frac{\partial f_i}{\partial x_j} \) at the point \( x \), for \( i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \) (unless \( \text{userflag} \) is set to a negative value by \( f \)). If \( \text{userflag} = 1 \) on entry, \( fjac \) must not be changed.
- **tdfjac**: Input: the last dimension of array \( fjac \) as declared in the function from which \( \text{nag_zero_nonlin_eqns_deriv_1} \) is called.
- **userflag**: Input: \( \text{userflag} = 1 \) or 2.
  - If \( \text{userflag} = 1 \), \( fvec \) is to be updated.
  - If \( \text{userflag} = 2 \), \( fjac \) is to be updated.
  - Output: in general, \( \text{userflag} \) should not be reset by \( f \). If, however, the user wishes to terminate execution (perhaps because some illegal point \( x \) has been reached), then \( \text{userflag} \) should be set to a negative integer. This value will be returned through \( \text{fail.errnum} \).

**xtol**: Input: the accuracy in \( x \) to which the solution is required.
- Suggested value: the square root of the \textit{machine precision}.
- Constraint: \( xtol \geq 0.0 \).

**comm**: Input/Output: pointer to a structure of type Nag_User with the following member:
- **p - Pointer**: Input/Output: 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. \( \text{comm.p} = (\text{Pointer})\&s \). The type pointer will be \textit{void *} with a C compiler that defines \textit{void *} and \textit{char *} otherwise.

**fail**: The NAG error parameter, see the Essential Introduction to the NAG C Library.

### 5. Error Indications and Warnings

**NE_INT_ARG_LE**
- On entry, \( n \) must not be less than or equal to 0: \( n = \langle \text{value} \rangle \).

**NE_REAL_ARG_LT**
- On entry, \( xtol \) must not be less than 0.0: \( xtol = \langle \text{value} \rangle \).

**NE_2_INT_ARG_LT**
- On entry \( tdfjac = \langle \text{value} \rangle \) while \( n = \langle \text{value} \rangle \). These parameters must satisfy \( tdfjac \geq n \).
NE_ALLOC_FAIL
Memory allocation failed.

NE_USER_STOP
User requested termination, user flag value = ⟨value⟩.

NE_TOO_MANY_FUNC_EVAL
There have been at least 100 * (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 = ⟨value⟩.

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

6. Further Comments
The time required by nag_zero_nonlin_eqns_deriv_1 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 is about 11.5 × n^2 to process each evaluation
 of the functions and about 1.3 × n^3 to process each evaluation of the Jacobian. Unless f can be
 evaluated quickly, the timing of nag_zero_nonlin_eqns_deriv_1 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.

6.1. Accuracy
If ˆx is the true solution, nag_zero_nonlin_eqns_deriv_1 tries to ensure that

   ∥x − ˆx∥ ≤ xtol × ∥ˆx∥.

   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 nag_zero_nonlin_eqns_deriv_1 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 routine 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
   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 with a tighter tolerance.

6.2. References
Laboratory, ANL-80-74.
Nonlinear Algebraic Equations P Rabinowitz (ed) Gordon and Breach.

7. See Also
nag_zero_nonlin_eqns_1 (c05tbc)
nag_check_deriv_1 (c05zcc)
8. Example

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

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

8.1. Program Text

```c
/* nag_zero_nonlin_eqns_deriv_1(c05ubc) Example Program */
*/

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

#ifdef NAG_PROTO
static void f(Integer n, double x[], double fvec[], double fjac[],
               Integer tdfjac, Integer *userflag, Nag_User *comm);
#else
static void f();
#endif

#define NMAX 9
#define TDFJAC NMAX

main()
{
    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");
        for (j=0; j<n; j++)
            Vprintf("%12.4f",x[j], (j%3==2 || j==n-1) ? \n" : " );
        exit(EXIT_SUCCESS);
    } else
    {
        Vprintf("%s", 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");
            for (j=0; j<n; j++)
                Vprintf("%12.4f",x[j], (j%3==2 || j==n-1) ? \n" : " );
            exit(EXIT_FAILURE);
        }
    }

3.c05ubc.4 [NP:3275/5/pdf]"
#ifdef NAG_PROTO
static void f(Integer n, double x[], double fvec[], double fjac[],
               Integer tdfjac, Integer *userflag, Nag_User *comm)
#else
static void f(n, x, fvec, fjac, tdfjac, userflag, comm)
#endif
{
    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;
        }
    }
}

8.2. Program Data

None.

8.3. Program Results

**c05ubc Example Program Results**

Final approximate solution

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5707</td>
<td>-0.6816</td>
<td>-0.7017</td>
</tr>
<tr>
<td>-0.7042</td>
<td>-0.7014</td>
<td>-0.6919</td>
</tr>
<tr>
<td>-0.6658</td>
<td>-0.5960</td>
<td>-0.4164</td>
</tr>
</tbody>
</table>