nag_zero_nonlin_eqns_deriv (c05pbc)

1. Purpose

*nag_zero_nonlin_eqns_deriv (c05pbc)* 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(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, NagError *fail)
```

3. Description

The system of equations is defined as:

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

*nag_zero_nonlin_eqns_deriv* 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* 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:
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 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.

  fjac[n * tdfjac]
  Output: if userflag = 2 on entry, fjac[(i-1)*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.

  tdfjac
  Input: the last dimension of array fjac as declared in the function from which
  nag_zero_nonlin_eqns_deriv is called.

  userflag
  Input: userflag = 1 or 2.
  If userflag = 1, fvec is to be updated.
  If userflag = 2, fjac is to be updated.

  Output: 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.

xtol
  Input: the accuracy in x to which the solution is required.
  Suggested value: the square root of the machine precision.
  Constraint: xtol ≥ 0.0.

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 = ⟨value⟩.

NE_REAL_ARG_LT
  On entry, xtol must not be less than 0.0: xtol = ⟨value⟩.

NE_2_INT_ARG_LT
  On entry tdfjac = ⟨value⟩ while n = ⟨value⟩. These parameters must satisfy tdfjac ≥ 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⟩.
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 from a different starting point may avoid the region of difficulty.

6. Further Comments

The time required by nag_zero_nonlin_eqns_deriv 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 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 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 \( \hat{x} \) is the true solution, nag_zero_nonlin_eqns_deriv tries to ensure that

\[
\| x - \hat{x} \| \leq xtol \times \| \hat{x} \|.
\]

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 usually avoids the possibility.

If \( xtol \) is less than \textit{machine precision} and the above test is satisfied with the \textit{machine precision} in place of \( xtol \), then the routine exits with \textit{NE_XTOL_TOO_SMALL}.

\textbf{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 may incorrectly indicate convergence. The coding of the Jacobian can be checked using nag_check_deriv (c05zbc). If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning nag_zero_nonlin_eqns_deriv with a tighter tolerance.

6.2. References


7. See Also

nag_zero_nonlin_eqns (c05nbc)
nag_check_deriv (c05zbc)

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

/* nag_zero_nonlin_eqns_deriv(c05pbc) 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);
#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;
Integer n = NMAX;
Vprintf("c05pbc 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);
c05pbc(n, x, fvec, fjac, (Integer)TDFJAC, f, xtol, &fail);
if (fail.code == NE_NOERROR)
{
    Vprintf("Final approximate solution\n");
    for (j=0; j<n; j++)
        Vprintf("%.4f",x[j], (j%3==2 || j==n-1) ? "\n" : "");
exi(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");
        for (j=0; j<n; j++)
            Vprintf("%.4f",x[j], (j%3==2 || j==n-1) ? "\n" : "");
    }
    exit(EXIT_FAILURE);
}

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

3.c05pbc.4 [NP:275/5/5/pdf]
double x[], fvec[], fjac[];
Integer tdfjac;
Integer *userflag;
#endif
{
#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;
    }
}

8.2. Program Data
None.

8.3. Program Results

c05pbc Example Program Results
Final approximate solution

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