nag_zero_nonlin_eqns_expert (c05qcc) is a comprehensive function that finds a solution of a system of nonlinear equations by a modification of the Powell hybrid method.

The system of equations is defined as:

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

nag_zero_nonlin_eqns_expert (c05qcc) is based on the MINPACK routine HYBRD (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 rank-1 method of Broyden. At the starting point, the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. For more details see Powell (1970).


Arguments

1: \textbf{fcn} – function, supplied by the user

\textit{External Function}

\texttt{fcn} must return the values of the functions \( f_i \) at a point \( x \), unless \texttt{iflag} = 0 on entry to \texttt{nag_zero_nonlin_eqns_expert} (c05qcc).

The specification of \texttt{fcn} is:

\texttt{void fcn (Integer n, const double x[], double fvec[], Nag_Comm \*comm, Integer \*iflag)}

1: \textbf{n} – Integer

\textit{Input}

\textit{On entry:} \( n \), the number of equations.
### c05qcc

**On entry**: the components of the point \( x \) at which the functions must be evaluated.

**Input**

**On entry**: if \( \text{iflag} = 0 \), \( \text{fvec} \) contains the function values \( f_i(x) \) and must not be changed.

**On exit**: if \( \text{iflag} > 0 \) on entry, \( \text{fvec} \) must contain the function values \( f_i(x) \) (unless \( \text{iflag} \) is set to a negative value by \( \text{fcn} \)).

**Input/Output**

**Pointer to structure of type Nag_Comm**: the following members are relevant to \( \text{fcn} \).

- **user** – double *
- **iuser** – Integer *
- **p** – Pointer

The type Pointer will be `void *`. Before calling \( \text{nag_zero_nonlin_eqns_expert} \) \( \text{(c05qcc)} \) you may allocate memory and initialize these pointers with various quantities for use by \( \text{fcn} \) when called from \( \text{nag_zero_nonlin_eqns_expert} \) \( \text{(c05qcc)} \) (see Section 3.2.1.1 in the Essential Introduction).

**Input/Output**

**On entry**: \( \text{iflag} \geq 0 \).

- \( \text{iflag} = 0 \) and \( \text{fvec} \) are available for printing (see `nprint`).
- \( \text{iflag} > 0 \) \( \text{fvec} \) must be updated.

**On exit**: in general, \( \text{iflag} \) should not be reset by \( \text{fcn} \). If, however, you wish to terminate execution (perhaps because some illegal point \( x \) has been reached), then \( \text{iflag} \) should be set to a negative integer.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Input**

**On entry**: \( n \), the number of equations.

**Constraint**: \( n > 0 \).

<table>
<thead>
<tr>
<th>( x[n] )</th>
<th>double</th>
</tr>
</thead>
</table>

**Input/Output**

**On entry**: an initial guess at the solution vector.

**On exit**: the final estimate of the solution vector.

<table>
<thead>
<tr>
<th>( \text{fvec}[n] )</th>
<th>double</th>
</tr>
</thead>
</table>

**Output**

**On exit**: the function values at the final point returned in \( x \).

<table>
<thead>
<tr>
<th>( \text{xtol} )</th>
<th>double</th>
</tr>
</thead>
</table>

**Input**

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

**Suggested value**: \( \sqrt{\epsilon} \), where \( \epsilon \) is the **machine precision** returned by \( \text{nag_machine_precision} \) \( \text{(X02AJC)} \).

**Constraint**: \( \text{xtol} \geq 0.0 \).
6: \text{maxfev} \quad \text{Integer} \quad \text{Input}

\text{On entry:} \ the \ maximum \ number \ of \ calls \ to \ \text{fcn} \ with \ \text{iflag} \neq 0. \ \text{nag_zero_nonlin_eqns_expert} \ (c05qcc) \ will \ exit \ with \ \text{fail.code} = \text{NE_TOO_MANY_FEVALS}, \ if, \ at \ the \ end \ of \ an \ iteration, \ the 
\text{number} \ of \ calls \ to \ \text{fcn} \ \text{exceeds} \ \text{maxfev}.

\text{Suggested value:} \ \text{maxfev} = 200 \times (n + 1).
\text{Constraint:} \ \text{maxfev} > 0.

7: \text{ml} \quad \text{Integer} \quad \text{Input}

\text{On entry:} \ the \ number \ of \ subdiagonals \ within \ the \ band \ of \ the \ Jacobian \ matrix. \ (If \ the \ Jacobian \ is \ not \ banded, \ or \ you \ are \ unsure, \ set \ \text{ml} = n - 1.)
\text{Constraint:} \ \text{ml} \geq 0.

8: \text{mu} \quad \text{Integer} \quad \text{Input}

\text{On entry:} \ the \ number \ of \ superdiagonals \ within \ the \ band \ of \ the \ Jacobian \ matrix. \ (If \ the \ Jacobian \ is \ not \ banded, \ or \ you \ are \ unsure, \ set \ \text{mu} = n - 1.)
\text{Constraint:} \ \text{mu} \geq 0.

9: \text{epsfcn} \quad \text{double} \quad \text{Input}

\text{On entry:} \ a \ rough \ estimate \ of \ the \ largest \ relative \ error \ in \ the \ functions. \ It \ is \ used \ in \ determining \ a 
suitable \ step \ for \ a \ forward \ difference \ approximation \ to \ the \ Jacobian. \ If \ \text{epsfcn} \ \text{is} \ less \ than 
\text{machine precision} \ (\text{returned} \ by \ \text{nagMachinePrecision} \ (\text{X02AJC})) \ then \ \text{machine precision} \ is 
\text{used}. \ Consequently \ a \ value \ of \ 0.0 \ \text{will} \ \text{often} \ \text{be} \ \text{suitable}.

\text{Suggested value:} \ \text{epsfcn} = 0.0.

10: \text{scale_mode} \quad \text{Nag_ScaleType} \quad \text{Input}

\text{On entry:} \ indicates \ whether \ or \ not \ you \ have \ provided \ scaling \ factors \ in \ \text{diag}.
\text{If} \ \text{scale_mode} = \text{Nag_ScaleProvided} \ the \ scaling \ must \ have \ been \ specified \ in \ \text{diag}.
\text{Otherwise, if} \ \text{scale_mode} = \text{Nag_NoScaleProvided}, \ the \ variables \ will \ be \ scaled \ internally.
\text{Constraint:} \ \text{scale_mode} = \text{Nag_NoScaleProvided} \ or \ \text{Nag_ScaleProvided}.

11: \text{diag} \quad \text{double} \quad \text{Input/Output}

\text{On entry:} \ if \ \text{scale_mode} = \text{Nag_ScaleProvided}, \ \text{diag} \ \text{must} \ \text{contain} \ \text{multiplicative} \ \text{scale} \ \text{factors} \ \text{for} \ \text{the} \ \text{variables}.
\text{If} \ \text{scale_mode} = \text{Nag_NoScaleProvided}, \ \text{diag} \ \text{need} \ \text{not} \ \text{be} \ \text{set}.
\text{Constraint:} \ \text{if} \ \text{scale_mode} = \text{Nag_ScaleProvided}, \ \text{diag}[i - 1] > 0.0, \ \text{for} \ i = 1, 2, \ldots, n.
\text{On exit:} \ \text{the} \ \text{scale} \ \text{factors} \ \text{actually} \ \text{used} \ \text{(computed} \ \text{internally} \ \text{if} \ \text{scale_mode} = \text{Nag_NoScaleProvided}).

12: \text{factor} \quad \text{double} \quad \text{Input}

\text{On entry:} \ a \ quantity \ to \ be \ used \ in \ determining \ the \ initial \ step \ bound. \ In \ most \ cases, \ \text{factor} \ \text{should} \ \text{lie} \ \text{between} \ 0.1 \ \text{and} \ 100.0. \ \text{(The} \ \text{step} \ \text{bound} \ \text{is} \ \text{factor} \times ||\text{diag} \times x||_2 \ \text{if} \ \text{this} \ \text{is} \ \text{nonzero}; \ \text{otherwise} \ \text{the} \ \text{bound} \ \text{is} \ \text{factor}.)}

\text{Suggested value:} \ \text{factor} = 100.0.
\text{Constraint:} \ \text{factor} > 0.0.
13: **nprint** – Integer  
   *Input*  
   *On entry:* indicates whether (and how often) special calls to *fcn*, with *iflag* set to 0, are to be made for printing purposes.  
   
   **nprint** ≤ 0  
   No calls are made.  
   
   **nprint** > 0  
   *fcn* is called at the beginning of the first iteration, every *nprint* iterations thereafter and immediately before the return from nag_zero_nonlin_eqns_expert (c05qcc).  

14: **nfev** – Integer *  
   *Output*  
   *On exit:* the number of calls made to *fcn* with *iflag* > 0.  

15: *fjac*[n × n] – double  
   *Output*  
   *Note:* the (i,j)th element of the matrix is stored in *fjac*(j-1) × n + i - 1].  
   *On exit:* the orthogonal matrix *Q* produced by the QR factorization of the final approximate Jacobian.  

16: *r*[n × (n + 1)/2] – double  
   *Output*  
   *On exit:* the upper triangular matrix *R* produced by the QR factorization of the final approximate Jacobian, stored row-wise.  

17: *qtf*[n] – double  
   *Output*  
   *On exit:* the vector *Q*ᵀ*f*.  

18: **comm** – Nag_Comm *  
   The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).  

19: **fail** – NagError *  
   *Input/Output*  
   The NAG error argument (see Section 3.6 in the Essential Introduction).  

### 6 Error Indicators and Warnings  

**NE_ALLOC_FAIL**  
Dynamic memory allocation failed.  
See Section 3.2.1.2 in the Essential Introduction for further information.  

**NE_BAD_PARAM**  
On entry, argument ⟨value⟩ had an illegal value.  

**NE_DIAG_ELEMENTS**  
On entry, scale_mode = Nag_SCALEProvided and diag contained a non-positive element.  

**NE_INT**  
On entry, maxev = ⟨value⟩.  
Constraint: maxev > 0.  
On entry, ml = ⟨value⟩.  
Constraint: ml ≥ 0.  
On entry, mu = ⟨value⟩.  
Constraint: mu ≥ 0.
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n > 0 \).

**NE_INTERNAL_ERROR**
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

**NE_NO_IMPROVEMENT**
The iteration is not making good progress, as measured by the improvement from the last \( \langle \text{value} \rangle \) iterations.
The iteration is not making good progress, as measured by the improvement from the last \( \langle \text{value} \rangle \) Jacobian evaluations.

**NE_NO_LICENCE**
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

**NE_REAL**
On entry, \( \text{factor} = \langle \text{value} \rangle \).
Constraint: \( \text{factor} > 0.0 \).
On entry, \( \text{xtol} = \langle \text{value} \rangle \).
Constraint: \( \text{xtol} \geq 0.0 \).

**NE_TOO_MANY_FEVALS**
There have been at least \( \text{maxfev} \) calls to \( \text{fcn} \): \( \text{maxfev} = \langle \text{value} \rangle \). Consider restarting the calculation from the final point held in \( \text{x} \).

**NE_TOO_SMALL**
No further improvement in the solution is possible. \( \text{xtol} \) is too small: \( \text{xtol} = \langle \text{value} \rangle \).

**NE_USER_STOP**
\( \text{iflag} \) was set negative in \( \text{fcn} \). \( \text{iflag} = \langle \text{value} \rangle \).

7 **Accuracy**
If \( \hat{x} \) is the true solution and \( D \) denotes the diagonal matrix whose entries are defined by the array \( \text{diag} \), then \( \text{c05qcc} \) tries to ensure that
\[
\| D(x - \hat{x}) \|_2 \leq \text{xtol} \times \| D\hat{x} \|_2.
\]
If this condition is satisfied with \( \text{xtol} = 10^{-k} \), then the larger components of \( Dx \) have \( k \) significant decimal digits. There is a danger that the smaller components of \( Dx \) may have large relative errors, but the fast rate of convergence of \( \text{nag_zero_nonlin_eqns_expert} \) (c05qcc) usually obviates this possibility.
If \( \text{xtol} \) is less than \textit{machine precision} and the above test is satisfied with the \textit{machine precision} in place of \( \text{xtol} \), then the function exits with \textit{fail code} = \text{NE_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 convergence test assumes that the functions are reasonably well behaved. If this condition is not satisfied, then \( \text{nag_zero_nonlin_eqns_expert} \) (c05qcc) may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning \( \text{nag_zero_nonlin_eqns_expert} \) (c05qcc) with a lower value for \( \text{xtol} \).
8 Parallelism and Performance

nag_zero_nonlin_eqns_expert (c05qcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zero_nonlin_eqns_expert (c05qcc) 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 function. 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 nag_zero_nonlin_eqns_expert (c05qcc). The total size of these arrays amounts to $4 \times n$ double elements.

The time required by nag_zero_nonlin_eqns_expert (c05qcc) 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_expert (c05qcc) to process each evaluation of the functions is approximately $11.5 \times n^2$. The timing of nag_zero_nonlin_eqns_expert (c05qcc) is strongly influenced by the time spent evaluating the functions.

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

The number of function evaluations required to evaluate the Jacobian may be reduced if you can specify ml and mu accurately.

10 Example

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

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

10.1 Program Text

/* nag_zero_nonlin_eqns_expert (c05qcc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 25, 2014. */

#include <nag.h>
#include <nagx04.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagc05.h>
#include <nagx02.h>

#ifdef __cplusplus
extern "C" {
#endif

static void NAG_CALL fcn(Integer n, const double x[], double fvec[], Nag_Comm *comm, Integer *iflag);

#ifdef __cplusplus
}
#endif

static Integer nprint = 0;


int main(void)
{
    static double ruser[1] = {-1.0};
    Integer exit_status = 0, i, n = 9, maxfev, ml, mu, nfev;
    double *diag = 0, *fjac = 0, *fvec = 0, *qtf = 0, *r = 0, *x = 0;
    double epsfcn, factor, xtol;
    /* Nag Types */
    NagError fail;
    Nag_Comm comm;
    Nag_ScaleType scale_mode;

    INIT_FAIL(fail);

    printf("nag_zero_nonlin_eqns_expert (c05qcc) Example Program Results\n");
    /* For communication with user-supplied functions: */
    comm.user = ruser;
    if (n > 0)
    {
        if (!(diag = NAG_ALLOC(n, double)) ||
            !(fjac = NAG_ALLOC(n*n, double)) ||
            !(fvec = NAG_ALLOC(n, double)) ||
            !(qtf = NAG_ALLOC(n, double)) ||
            !(r = NAG_ALLOC(n*(n+1)/2, double)) ||
            !(x = NAG_ALLOC(n, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else
    {
        printf("Invalid n.\n");
        exit_status = 1;
        goto END;
    }

    /* The following starting values provide a rough solution. */
    for (i = 0; i < n; i++)
        x[i] = -1.0;
    /* nag_machine_precision (x02ajc).
    * The machine precision
    */
    xtol = sqrt(nag_machine_precision);
    for (i = 0; i < n; i++)
        diag[i] = 1.0;
    maxfev = 2000;
    ml = 1;
    mu = 1;
    epsfcn = 0.0;
    scale_mode = Nag_ScaleProvided;
    factor = 100.0;

    /* nag_zero_nonlin_eqns_expert (c05qcc).
    * Solution of a system of nonlinear equations (function
    * values only)
    */
    nag_zero_nonlin_eqns_expert(fcn, n, x, fvec, xtol, maxfev, ml, mu,
                                epsfcn, scale_mode, diag, factor, nprint, &nfev,
                                fjac, r, qtf, &comm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zero_nonlin_eqns_expert (c05qcc).\n%s\n", fail.message);
    }
exit_status = 1;
   if (fail.code != NE_TOO_MANY_FEVALS &&
       fail.code != NE_TOO_SMALL &&
       fail.code != NE_NO_IMPROVEMENT)
       goto END;
   }
   printf(fail.code == NE_NOERROR ? "Final approximate" : "Approximate");
   printf(" solution\n\n");
   for (i = 0; i < n; i++)
      printf("%12.4f%s", x[i], (i%3 == 2 || i == n-1)?"\n":" ");
   if (fail.code != NE_NOERROR)
      exit_status = 2;
END:
   NAG_FREE(diag);
   NAG_FREE(fjac);
   NAG_FREE(fvec);
   NAG_FREE(qtf);
   NAG_FREE(r);
   NAG_FREE(x);
   return exit_status;
}
static void NAG_CALL fcn(Integer n, const double x[],
const double fvec[],
Nag_Comm *comm, Integer *iflag)
{
   Integer k;
   if (comm->user[0] == -1.0)
   {
      printf("(User-supplied callback fcn, first invocation.)\n");
      comm->user[0] = 0.0;
   }
   if (*iflag == 0)
   {
      if (nprint > 0)
      {
         /* Insert print statements here if desired. */
      }
   }
   else
   {
      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;
      }
      /* Set iflag negative to terminate execution for any reason. */
      *iflag = 0;
   }

10.2 Program Data

None.
### 10.3 Program Results

nag_zero_nonlin_eqns_expert (c05qcc) Example Program Results
(User-supplied callback fcn, first invocation.)

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>