

# NAG Library Function Document

## nag\_zero\_nonlin\_eqns\_expert (c05qcc)

### 1 Purpose

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.

### 2 Specification

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

void nag_zero_nonlin_eqns_expert (
    void (*fcf)(Integer n, const double x[], double fvec[], Nag_Comm *comm,
        Integer *iflag),
    Integer n, double x[], double fvec[], double xtol, Integer maxfev,
    Integer ml, Integer mu, double epsfcf, Nag_ScaleType scale_mode,
    double diag[], double factor, Integer nprint, Integer *nfev,
    double fjac[], double r[], double qtf[], Nag_Comm *comm, NagError *fail)
```

### 3 Description

The system of equations is defined as:

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

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).

### 4 References

Moré J J, Garbow B S and Hillstom 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 Arguments

- 1: **fcf** – function, supplied by the user *External Function*
- fcf** must return the values of the functions  $f_i$  at a point  $x$ , unless **iflag** = 0 on entry to nag\_zero\_nonlin\_eqns\_expert (c05qcc).

The specification of **fcf** is:

```
void fcf (Integer n, const double x[], double fvec[], Nag_Comm *comm,
    Integer *iflag)
```

1: **n** – Integer

*Input*

On entry:  $n$ , the number of equations.

- |    |  |                     |
|----|--|---------------------|
| 2: | <b>x[n]</b> – const double<br><i>On entry:</i> the components of the point $x$ at which the functions must be evaluated.   | <i>Input</i>        |
| 3: | <b>fvec[n]</b> – double<br><i>On entry:</i> if <b>iflag</b> = 0, <b>fvec</b> contains the function values $f_i(x)$ and must not be changed.<br><i>On exit:</i> if <b>iflag</b> > 0 on entry, <b>fvec</b> must contain the function values $f_i(x)$ (unless <b>iflag</b> is set to a negative value by <b>fcn</b> ).  | <i>Input/Output</i> |
| 4: | <b>comm</b> – Nag_Comm *<br>Pointer to structure of type Nag_Comm; the following members are relevant to <b>fcn</b> .<br><br><b>user</b> – double *<br><b>iuser</b> – Integer *<br><b>p</b> – Pointer<br><br>The type Pointer will be void *. Before calling nag_zero_nonlin_eqns_expert (c05qcc) you may allocate memory and initialize these pointers with various quantities for use by <b>fcn</b> when called from nag_zero_nonlin_eqns_expert (c05qcc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation). |                     |
| 5: | <b>iflag</b> – Integer *<br><i>On entry:</i> <b>iflag</b> ≥ 0.<br><br><b>iflag</b> = 0<br><b>x</b> and <b>fvec</b> are available for printing (see <b>nprint</b> ).<br><br><b>iflag</b> > 0<br><b>fvec</b> must be updated.<br><br><i>On exit:</i> in general, <b>iflag</b> should not be reset by <b>fcn</b> . If, however, you wish to terminate execution (perhaps because some illegal point <b>x</b> has been reached), then <b>iflag</b> should be set to a negative integer.  | <i>Input/Output</i> |
- 2: **n** – Integer *Input*  
*On entry:*  $n$ , the number of equations.  
*Constraint:* **n** > 0.
- 3: **x[n]** – double *Input/Output*  
*On entry:* an initial guess at the solution vector.  
*On exit:* the final estimate of the solution vector.
- 4: **fvec[n]** – double *Output*  
*On exit:* the function values at the final point returned in **x**.
- 5: **xtol** – double *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 nag\_machine\_precision (X02AJC).  
*Constraint:* **xtol** ≥ 0.0.

- 6: **maxfev** – Integer *Input*  
*On entry:* the maximum number of calls to **fcn** with **iflag**  $\neq 0$ . nag\_zero\_nonlin\_eqns\_expert (c05qcc) will exit with **fail.code** = NE\_TOO\_MANY\_FEVALS, if, at the end of an iteration, the number of calls to **fcn** exceeds **maxfev**.  
*Suggested value:* **maxfev** =  $200 \times (\mathbf{n} + 1)$ .  
*Constraint:* **maxfev** > 0.
- 7: **ml** – Integer *Input*  
*On entry:* the number of subdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set **ml** =  $\mathbf{n} - 1$ .)  
*Constraint:* **ml**  $\geq 0$ .
- 8: **mu** – Integer *Input*  
*On entry:* the number of superdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set **mu** =  $\mathbf{n} - 1$ .)  
*Constraint:* **mu**  $\geq 0$ .
- 9: **epsfcn** – double *Input*  
*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 **epsfcn** is less than *machine precision* (returned by nag\_machine\_precision (X02AJC)) then *machine precision* is used. Consequently a value of 0.0 will often be suitable.  
*Suggested value:* **epsfcn** = 0.0.
- 10: **scale\_mode** – Nag\_ScaleType *Input*  
*On entry:* indicates whether or not you have provided scaling factors in **diag**.  
If **scale\_mode** = Nag\_ScaleProvided the scaling must have been specified in **diag**.  
Otherwise, if **scale\_mode** = Nag\_NoScaleProvided, the variables will be scaled internally.  
*Constraint:* **scale\_mode** = Nag\_NoScaleProvided or Nag\_ScaleProvided.
- 11: **diag[n]** – double *Input/Output*  
*On entry:* if **scale\_mode** = Nag\_ScaleProvided, **diag** must contain multiplicative scale factors for the variables.  
If **scale\_mode** = Nag\_NoScaleProvided, **diag** need not be set.  
*Constraint:* if **scale\_mode** = Nag\_ScaleProvided, **diag**[ $i - 1$ ] > 0.0, for  $i = 1, 2, \dots, n$ .  
*On exit:* the scale factors actually used (computed internally if **scale\_mode** = Nag\_NoScaleProvided).
- 12: **factor** – double *Input*  
*On entry:* a quantity to be used in determining the initial step bound. In most cases, **factor** should lie between 0.1 and 100.0. (The step bound is **factor**  $\times \|\mathbf{diag} \times \mathbf{x}\|_2$  if this is nonzero; otherwise the bound is **factor**.)  
*Suggested value:* **factor** = 100.0.  
*Constraint:* **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**  $\leq 0$   
 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**  $\times$  **n**] – double *Output*  
**Note:** the  $(i, j)$ th element of the matrix is stored in **fjac**[( $j - 1$ )  $\times$  **n** +  $i - 1$ ].  
*On exit:* the orthogonal matrix  $Q$  produced by the  $QR$  factorization of the final approximate Jacobian.
- 16: **r**[**n**  $\times$  (**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 2.3.1.1 in How to Use the NAG Library and its Documentation).
- 19: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_DIAG\_ELEMENTS

On entry, **scale\_mode** = Nag\_ScaleProvided and **diag** contained a non-positive element.

### NE\_INT

On entry, **maxfev** =  $\langle value \rangle$ .

Constraint: **maxfev**  $> 0$ .

On entry, **ml** =  $\langle value \rangle$ .

Constraint: **ml**  $\geq 0$ .

On entry, **mu** =  $\langle value \rangle$ .

Constraint: **mu**  $\geq 0$ .

On entry, **n** =  $\langle value \rangle$ .

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_NO\_IMPROVEMENT

The iteration is not making good progress, as measured by the improvement from the last  $\langle value \rangle$  iterations.

The iteration is not making good progress, as measured by the improvement from the last  $\langle value \rangle$  Jacobian evaluations.

### NE\_NO\_LICENCE

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

See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

### NE\_REAL

On entry, **factor** =  $\langle value \rangle$ .

Constraint: **factor**  $> 0.0$ .

On entry, **xtol** =  $\langle value \rangle$ .

Constraint: **xtol**  $\geq 0.0$ .

### NE\_TOO\_MANY\_FEVALS

There have been at least **maxfev** calls to **fcn**: **maxfev** =  $\langle value \rangle$ . Consider restarting the calculation from the final point held in **x**.

### NE\_TOO\_SMALL

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

### NE\_USER\_STOP

**iflag** was set negative in **fcn**. **iflag** =  $\langle value \rangle$ .

## 7 Accuracy

If  $\hat{x}$  is the true solution and  $D$  denotes the diagonal matrix whose entries are defined by the array **diag**, then nag\_zero\_nonlin\_eqns\_expert (c05qcc) tries to ensure that

$$\|D(x - \hat{x})\|_2 \leq \mathbf{xtol} \times \|D\hat{x}\|_2.$$

If this condition is satisfied with  $\mathbf{xtol} = 10^{-k}$ , 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 nag\_zero\_nonlin\_eqns\_expert (c05qcc) 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 function exits with **fail.code** = 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 `nag_zero_nonlin_eqns_expert` (c05qcc) may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning `nag_zero_nonlin_eqns_expert` (c05qcc) with a lower value for `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  $x_1, \dots, x_9$  which satisfy the tridiagonal equations:

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

### 10.1 Program Text

```
/* nag_zero_nonlin_eqns_expert (c05qcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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[], 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

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

---