NAG Library Function Document

nag_zero_nonlin_eqns_deriv_rcomm (c05rdc)

1 Purpose

nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) is a comprehensive reverse communication function that finds a solution of a system of nonlinear equations by a modification of the Powell hybrid method. You must provide the Jacobian.

2 Specification

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

```
void nag_zero_nonlin_eqns_deriv_rcomm (Integer *irevcm, Integer n,
     double x[], double fvec[], double fjac[], double xtol,
     Nag_ScaleType scale_mode, double diag[], double factor, double r[],
     double qtf[], Integer iwsav[], double rwsav[], NagError *fail)
```

3 Description

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_deriv_rcomm (c05rdc) is based on the MINPACK routine HYBRJ (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. 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 Arguments

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than fvec and fjac must remain unchanged**.

1: **irevcm** – Integer * *Input/Output*

On initial entry: must have the value 0.

On intermediate exit: specifies what action you must take before re-entering nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) **with irevcm unchanged**. The value of **irevcm** should be interpreted as follows:

irevcm $= 1$
Indicates the start of a new iteration. No action is required by you, but **x** and **fvec** are available for printing.

irevcm $= 2$
Indicates that before re-entry to nag_zero_nonlin_eqns_deriv_rcomm (c05rdc), **fvec** must contain the function values $f_i(x)$.

irevcm $= 3$

Indicates that before re-entry to nag_zero_nonlin_eqns_deriv_rcomm (c05rdc), **fjac**$[(j-1)\times$ **n** $+ i - 1]$ must contain the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

On final exit: **irevcm** $= 0$, and the algorithm has terminated.

Constraint: **irevcm** $= 0$, 1, 2 or 3.

2: **n** – Integer *Input*

On entry: n, the number of equations.

Constraint: **n** > 0.

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

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

On intermediate exit: contains the current point.

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

4: **fvec**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $\neq 2$, **fvec** must not be changed.

If **irevcm** $= 2$, **fvec** must be set to the values of the functions computed at the current point **x**.

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

5: **fjac**[**n** $\times$ **n**] – double *Input/Output*

Note: the (i, j)th element of the matrix is stored in **fjac**$[(j-1)\times$ **n** $+ i - 1]$.

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $\neq 3$, **fjac** must not be changed.

If **irevcm** $= 3$, **fjac**$[(j-1)\times$ **n** $+ i - 1]$ must contain the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

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

6: **xtol** – double *Input*

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

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

Constraint: **xtol** ≥ 0.0.

7: **scale_mode** – Nag_ScaleType *Input*

On initial entry: indicates whether or not you have provided scaling factors in **diag**.

If **scale_mode** = Nag_ScaleProvided the scaling must have been supplied in **diag**.

Otherwise, if **scale_mode** = Nag_NoScaleProvided, the variables will be scaled internally.

Constraint: **scale_mode** = Nag_NoScaleProvided or Nag_ScaleProvided.

8: **diag**[**n**] – double *Input/Output*

On initial 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 $i = 1, 2, \ldots, n$.

On intermediate exit: **diag** must not be changed.

On final exit: the scale factors actually used (computed internally if **scale_mode** $=$ Nag_NoScaleProvided).

9: **factor** – double *Input*

On initial 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** $\times \|$**diag** $\times$ **x**$\|_2$ if this is nonzero; otherwise the bound is **factor**.)

Suggested value: **factor** $= 100.0$.

Constraint: **factor** > 0.0.

10: **r**[**n** $\times$ (**n** $+$ **1**)/**2**] – double *Input/Output*

On initial entry: need not be set.

On intermediate exit: must not be changed.

On final exit: the upper triangular matrix R produced by the QR factorization of the final approximate Jacobian, stored row-wise.

11: **qtf**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate exit: must not be changed.

On final exit: the vector $Q^{\mathrm{T}} f$.

12: **iwsav**[**17**] – Integer *Communication Array*

13: **rwsav**[**4** $\times$ **n** $+$ **10**] – double *Communication Array*

The arrays **iwsav** and **rwsav** MUST NOT be altered between calls to nag_zero_nonlin_eqns_deriv_rcomm (c05rdc).

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

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, **irevcm** $= \langle value \rangle$.
Constraint: **irevcm** $= 0$, 1, 2 or 3.

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.

NE_NO_IMPROVEMENT

The iteration is not making good progress, as measured by the improvement from the last $\langle value \rangle$ iterations. 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 7). Otherwise, rerunning nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) from a different starting point may avoid the region of difficulty.

The iteration is not making good progress, as measured by the improvement from the last $\langle value \rangle$ Jacobian evaluations. 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 7). Otherwise, rerunning nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) from a different starting point may avoid the region of difficulty.

NE_REAL

On entry, **factor** $= \langle value \rangle$.
Constraint: **factor** > 0.0.

On entry, **xtol** $= \langle value \rangle$.
Constraint: **xtol** ≥ 0.0.

NE_TOO_SMALL

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

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_deriv_rcomm (c05rdc) tries to ensure that

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

If this condition is satisfied with **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_deriv_rcomm (c05rdc) 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 and the Jacobian are coded consistently and that the functions are reasonably well behaved. If these conditions are not satisfied, then nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) may incorrectly indicate convergence. The coding of the Jacobian can be checked using nag_check_derivs (c05zdc). If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) with a lower value for **xtol**.

8 Parallelism and Performance

nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time required by nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) 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_rcomm (c05rdc) is approximately $11.5 \times n^2$ to process each evaluation of the functions and approximately $1.3 \times n^3$ to process each evaluation of the Jacobian. The timing of nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) 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.

10 Example

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

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

10.1 Program Text

```
/* nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#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[],
                           double fjac[], Integer irevcm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  Integer  exit_status = 0, i, n = 9, irevcm;
  double   *diag = 0, *fjac = 0, *fvec = 0, *qtf = 0, *r = 0, *x = 0,
           *rwsav = 0;
  Integer  *iwsav = 0;
  double   factor, xtol;
  /* Nag Types */
  NagError fail;
  Nag_ScaleType scale_mode;

  INIT_FAIL(fail);
```

```
  printf("nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) Example Program Results\n");
  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)) ||
          !(iwsav = NAG_ALLOC(17, Integer)) ||
          !(rwsav = NAG_ALLOC(4*n + 10, 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;

  scale_mode = Nag_ScaleProvided;
  factor = 100.0;
  irevcm = 0;

  /* nag_zero_nonlin_eqns_deriv_rcomm (c05rdc).
   * Solution of a system of nonlinear equations (function values only,
   * reverse communication)
   */

  do
    {
      nag_zero_nonlin_eqns_deriv_rcomm(&irevcm, n, x, fvec, fjac, xtol,
                                       scale_mode, diag, factor, r, qtf, iwsav,
                                       rwsav, &fail);

      switch (irevcm)
        {
        case 1:
          /* x and fvec are available for printing */
          break;
        case 2:
        case 3:
          fcn(n, x, fvec, fjac, irevcm);
          break;
        }

    } while (irevcm != 0);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zero_nonlin_eqns_deriv_rcomm (c05rdc).\n%s\n",
             fail.message);
      exit_status = 1;
      if (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);
  NAG_FREE(iwsav);
  NAG_FREE(rwsav);
  return exit_status;
}
static void NAG_CALL fcn(Integer n, const double x[], double fvec[],
                         double fjac[], Integer irevcm)
{
  Integer j, k;

  if (irevcm == 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 if (irevcm == 3)
    {
      for (k = 0; k < n; k++)
        {
          for (j = 0; j < n; j++)
            fjac[j*n + k] = 0.0;
          fjac[k*n + k] = 3.0 - x[k] * 4.0;
          if (k > 0)
            fjac[(k-1)*n + k] = -1.0;
          if (k < n-1)
            fjac[(k+1)*n + k] = -2.0;
        }
    }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_zero_nonlin_eqns_deriv_rcomm (c05rdc) Example Program Results
Final approximate solution

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