NAG Toolbox

nag_roots_sys_deriv_easy (c05rb)

1 Purpose

nag_roots_sys_deriv_easy (c05rb) is an easy-to-use 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 Syntax

```
[x, fvec, fjac, user, ifail] = nag_roots_sys_deriv_easy(fcn, x, 'n', n, 'xtol',
xtol, 'user', user)
```

```
[x, fvec, fjac, user, ifail] = c05rb(fcn, x, 'n', n, 'xtol', xtol, 'user', user)
```

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_roots_sys_deriv_easy (c05rb) is based on the MINPACK routine HYBRJ1 (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 requested, but it is not asked for 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 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 Parameters

5.1 Compulsory Input Parameters

1: **fcn** – SUBROUTINE, supplied by the user.

Depending upon the value of **iflag**, **fcn** must either return the values of the functions f_i at a point x or return the Jacobian at x.

```
[fvec, fjac, user, iflag] = fcn(n, x, fvec, fjac, user, iflag)
```

Input Parameters

1: **n** – INTEGER

n, the number of equations.

2: **x**(**n**) – REAL (KIND=nag_wp) array

The components of the point x at which the functions or the Jacobian must be evaluated.

3: **fvec**(**n**) – REAL (KIND=nag_wp) array

If **iflag** $= 2$, **fvec** contains the function values $f_i(x)$ and must not be changed.

4: **fjac**(**n**, **n**) – REAL (KIND=nag_wp) array

If **iflag** $= 1$, **fjac** contains the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$, and must not be changed.

5: **user** – INTEGER array

fcn is called from nag_roots_sys_deriv_easy (c05rb) with the object supplied to nag_roots_sys_deriv_easy (c05rb).

6: **iflag** – INTEGER

iflag $= 1$ or 2.

iflag $= 1$
  fvec is to be updated.

iflag $= 2$
  fjac is to be updated.

Output Parameters

1: **fvec**(**n**) – REAL (KIND=nag_wp) array

If **iflag** $= 1$ on entry, **fvec** must contain the function values $f_i(x)$ (unless **iflag** is set to a negative value by **fcn**).

2: **fjac**(**n**, **n**) – REAL (KIND=nag_wp) array

If **iflag** $= 2$ on entry, **fjac**(i, j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$, (unless **iflag** is set to a negative value by **fcn**).

3: **user** – INTEGER array

4: **iflag** – INTEGER

In general, **iflag** should not be reset by **fcn**. If, however, you wish to terminate execution (perhaps because some illegal point **x** has been reached), then **iflag** should be set to a negative integer.

2: **x**(**n**) – REAL (KIND=nag_wp) array

An initial guess at the solution vector.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the array **x**.

n, the number of equations.

Constraint: **n** > 0.

2: **xtol** – REAL (KIND=nag_wp)

Suggested value: $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** returned by nag_machine_precision (x02aj).

Default: $\sqrt{\textit{\textbf{machine precision}}}$

The accuracy in **x** to which the solution is required.

Constraint: **xtol** ≥ 0.0.

3: **user** – INTEGER array

user is not used by nag_roots_sys_deriv_easy (c05rb), but is passed to **fcn**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

1: **x**(**n**) – REAL (KIND=nag_wp) array

The final estimate of the solution vector.

2: **fvec**(**n**) – REAL (KIND=nag_wp) array

The function values at the final point returned in **x**.

3: **fjac**(**n**, **n**) – REAL (KIND=nag_wp) array

The orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

4: **user** – INTEGER array

5: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 2$ (*warning*)

There have been at least $100 \times (\mathbf{n} + 1)$ calls to **fcn**. Consider restarting the calculation from the point held in **x**.

ifail $= 3$ (*warning*)

No further improvement in the solution is possible.

ifail $= 4$ (*warning*)

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 7). Otherwise, rerunning nag_roots_sys_deriv_easy (c05rb) from a different starting point may avoid the region of difficulty.

ifail $= 5$ (*warning*)

iflag was set negative in **fcn**.

ifail $= 11$

Constraint: $\mathbf{n} > 0$.

ifail $= 12$

Constraint: **xtol** ≥ 0.0.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

If $\hat{x}$ is the true solution, nag_roots_sys_deriv_easy (c05rb) tries to ensure that

$$\|x - \hat{x}\|_2 \leq \textbf{xtol} \times \|\hat{x}\|_2.$$

If this condition is satisfied with $\textbf{xtol} = 10^{-k}$, 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_roots_sys_deriv_easy (c05rb) 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 **ifail** $= 3$.

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_roots_sys_deriv_easy (c05rb) may incorrectly indicate convergence. The coding of the Jacobian can be checked using nag_roots_sys_deriv_check (c05zd). If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning nag_roots_sys_deriv_easy (c05rb) with a lower value for **xtol**.

8 Further Comments

Local workspace arrays of fixed lengths are allocated internally by nag_roots_sys_deriv_easy (c05rb). The total size of these arrays amounts to $n \times (n + 13)/2$ double elements.

The time required by nag_roots_sys_deriv_easy (c05rb) 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_roots_sys_deriv_easy (c05rb) 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_roots_sys_deriv_easy (c05rb) 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.

9 Example

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

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

9.1 Program Text

```
     function c05rb_example

fprintf('c05rb example results\n\n');

% The following starting values provide a rough solution.
x = -ones(9, 1);
[xOut, fvec, fjac, user, ifail] = c05rb(@fcn, x);
switch ifail
  case {0}
    fprintf('\nFinal 2-norm of the residuals = %12.4e\n', norm(fvec));
    fprintf('\nFinal approximate solution\n');
    disp(xOut);
  case {2, 3, 4}
    fprintf('\nApproximate solution\n');
    disp(xOut);
end

function [fvec, fjac, user, iflag] = fcn(n, x, fvec, fjac, user, iflag)
  coeff = [-1, 3, -2, -2, -1];
  nd = double(n); % Can't use 64 bit integers in loops
  if (iflag ~= 2)
    fvec(1:nd) = (coeff(2)+coeff(3)*x(1:nd)).*x(1:nd) - coeff(5);
    fvec(2:nd) = fvec(2:nd) + coeff(1)*x(1:(nd-1));
    fvec(1:(nd-1)) = fvec(1:(nd-1)) + coeff(4)*x(2:nd);
  else
    fjac = zeros(nd, nd);

    fjac(1,1) = coeff(2) + 2*coeff(3)*x(1);
    fjac(1,2) = coeff(4);
    for k = 2:nd-1
      fjac(k,k-1) = coeff(1);
      fjac(k,k) = coeff(2) + 2*coeff(3)*x(k);
      fjac(k,k+1) = coeff(4);
    end
    fjac(nd,nd-1) = coeff(1);
    fjac(nd,nd) = coeff(2) + 2*coeff(3)*x(nd);
  end
```

9.2 Program Results

```
     c05rb example results

Final 2-norm of the residuals =   1.1926e-08

Final approximate solution
   -0.5707
   -0.6816
   -0.7017
   -0.7042
   -0.7014
   -0.6919
   -0.6658
   -0.5960
   -0.4164
```