NAG Toolbox

nag_roots_sparsys_func_expert (c05qs)

1 Purpose

nag_roots_sparsys_func_expert (c05qs) is an easy-to-use function that finds a solution of a sparse system of nonlinear equations by a modification of the Powell hybrid method.

2 Syntax

```
[x, fvec, rcomm, icomm, user, ifail] = nag_roots_sparsys_func_expert(fcn, x, init, rcomm, icomm, 'n', n, 'xtol', xtol, 'lrcomm', lrcomm, 'licomm', licomm, 'user', user)
[x, fvec, rcomm, icomm, user, ifail] = c05qs(fcn, x, init, rcomm, icomm, 'n', n, 'xtol', xtol, 'lrcomm', lrcomm, 'licomm', licomm, 'user', user)
```

3 Description

The system of equations is defined as:

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

nag_roots_sparsys_func_expert (c05qs) is based on the MINPACK routine HYBRD1 (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 sparse rank-1 method of Schubert (see Schubert (1970)). At the starting point, the sparsity pattern is determined and the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. Then, the sparsity structure is used to recompute an approximation to the Jacobian by forward differences with the least number of function evaluations. The function you supply must be able to compute only the requested subset of the function values. The sparse Jacobian linear system is solved at each iteration with nag_sparse_direct_real_gen_lu (f11me) computing the Newton step. For more details see Powell (1970) and Broyden (1965).

4 References

Broyden C G (1965) A class of methods for solving nonlinear simultaneous equations *Mathematics of Computation* **19(92)** 577–593

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

Schubert L K (1970) Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian *Mathematics of Computation* **24(109)** 27–30

5 Parameters

5.1 Compulsory Input Parameters

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

fcn must return the values of the functions f_i at a point x.

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

Input Parameters

1: **n** – INTEGER

n, the number of equations.

2: **lindf** – INTEGER

lindf specifies the number of indices i for which values of $f_i(x)$ must be computed.

3: **indf**(**lindf**) – INTEGER array

indf specifies the indices i for which values of $f_i(x)$ must be computed. The indices are specified in strictly ascending order.

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

The components of the point x at which the functions must be evaluated. **x**(i) contains the coordinate x_i.

5: **user** – INTEGER array

fcn is called from nag_roots_sparsys_func_expert (c05qs) with the object supplied to nag_roots_sparsys_func_expert (c05qs).

6: **iflag** – INTEGER

iflag > 0.

Output Parameters

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

fvec(i) must contain the function values $f_i(x)$, for all indices i in **indf**.

2: **user** – INTEGER array

3: **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. **x**(i) must contain the coordinate x_i.

3: **init** – LOGICAL

init must be set to *true* to indicate that this is the first time nag_roots_sparsys_func_expert (c05qs) is called for this specific problem. nag_roots_sparsys_func_expert (c05qs) then computes the dense Jacobian and detects and stores its sparsity pattern (in **rcomm** and **icomm**) before proceeding with the iterations. This is noticeably time consuming when **n** is large. If not enough storage has been provided for **rcomm** or **icomm**, nag_roots_sparsys_func_expert (c05qs) will fail. On exit with **ifail** $= 0$, 2, 3 or 4, **icomm**(1) contains nnz, the number of nonzero entries found in the Jacobian. On subsequent calls, **init** can be set to *false* if the problem has a Jacobian of the same sparsity pattern. In that case, the computation time required for the detection of the sparsity pattern will be smaller.

4: **rcomm**(**lrcomm**) – REAL (KIND=nag_wp) array

rcomm must not be altered between successive calls to nag_roots_sparsys_func_expert (c05qs).

5: **icomm**(**licomm**) – INTEGER array

If **ifail** $= 0$, 2, 3 or 4 on exit, **icomm**(1) contains nnz where nnz is the number of nonzero entries in the Jacobian.

icomm must not be altered between successive calls to nag_roots_sparsys_func_expert (c05qs).

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

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

Constraint: **xtol** ≥ 0.0.

3: **lrcomm** – INTEGER

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

The dimension of the array **rcomm**.

Constraint: **lrcomm** $\geq 12 + nnz$ where nnz is the number of nonzero entries in the Jacobian, as computed by nag_roots_sparsys_func_expert (c05qs).

4: **licomm** – INTEGER

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

The dimension of the array **icomm**.

Constraint: **licomm** $\geq 8 \times$ **n** $+ 19 + nnz$ where nnz is the number of nonzero entries in the Jacobian, as computed by nag_roots_sparsys_func_expert (c05qs).

5: **user** – INTEGER array

user is not used by nag_roots_sparsys_func_expert (c05qs), 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**. **fvec**(i) contains the function values f_i.

3: **rcomm**(**lrcomm**) – REAL (KIND=nag_wp) array

4: **icomm**(**licomm**) – INTEGER array

icomm must not be altered between successive calls to nag_roots_sparsys_func_expert (c05qs).

5: **user** – INTEGER array

6: **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 $200 \times (\mathbf{n} + 1)$ calls to **fcn**. Consider setting **init** $= \mathit{false}$ and 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.

ifail $= 5$ (*warning*)

iflag was set negative in **fcn**.

ifail $= 6$

lrcomm is too small.

ifail $= 7$

licomm is too small.

ifail $= 9$

An internal error has occurred. Code

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_sparsys_func_expert (c05qs) 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_sparsys_func_expert (c05qs) 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 are reasonably well behaved. If this condition is not satisfied, then nag_roots_sparsys_func_expert (c05qs) may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning nag_roots_sparsys_func_expert (c05qs) with a lower value for **xtol**.

8 Further Comments

Local workspace arrays of fixed lengths are allocated internally by nag_roots_sparsys_func_expert (c05qs). The total size of these arrays amounts to $8 \times n + 2 \times q$ double elements and $10 \times n + 2 \times q + 5$ integer elements where the integer q is bounded by $8 \times nnz$ and n^2 and depends on the sparsity pattern of the Jacobian.

The time required by nag_roots_sparsys_func_expert (c05qs) 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_sparsys_func_expert (c05qs) to process each evaluation of the functions depends on the number of nonzero entries in the Jacobian. The timing of nag_roots_sparsys_func_expert (c05qs) is strongly influenced by the time spent evaluating the functions.

When **init** is *true*, the dense Jacobian is first evaluated and that will take time proportional to n^2.

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:

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

It then perturbs the equations by a small amount and solves the new system.

9.1 Program Text

```
function c05qs_example

fprintf('c05qs example results\n\n');

% The following starting values provide a rough solution.
x = -ones(9, 1);
rcomm = zeros(39, 1);
icomm = zeros(118, 1, nag_int_name);
for i=0:1

  user = i; % Perturb the system?
  init = (i==0);

  [x, fvec, rcomm, icomm, user, ifail] = ...
```

```
     c05qs(@fcn, x, init, rcomm, icomm,'user', user);

  switch ifail
    case {0}
      fprintf('\nFinal 2-norm of the residuals = %12.4e\n', norm(fvec));
      fprintf('\nFinal approximate solution\n');
      disp(x);
    case {2, 3, 4}
      fprintf('\nApproximate solution\n');
      disp(x);
  end

end

function [fvec, user, iflag] = fcn(n, lindf, indf, x, user, iflag)
  fvec = zeros(n, 1);
  ind = 1;
  iflag = nag_int(0);
  alpha = (1/2)^7;
  theta = user*alpha;
  for i = 1:double(n)
    if indf(ind) ~= i
      continue;
    end
    fvec(i) = (3-(2+theta)*x(i))*x(i) + 1;
    if (i > 1)
      fvec(i) = fvec(i) - x(i-1);
    end
    if (i < n)
      fvec(i) = fvec(i) - 2*x(i+1);
    end
    ind = ind + 1;
    if (ind > lindf)
     break;
    end
  end
```

9.2 Program Results

```
     c05qs example results

Final 2-norm of the residuals =   1.7592e-09

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

Final 2-norm of the residuals =   2.6329e-13

Final approximate solution
   -0.5697
   -0.6804
   -0.7004
   -0.7029
   -0.7000
   -0.6906
   -0.6646
   -0.5951
   -0.4159
```