

# NAG Toolbox for MATLAB

## c05pb

### 1 Purpose

c05pb is an easy-to-use function to find 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, ifail] = c05pb(fcn, x, 'n', n, 'xtol', xtol)
```

### 3 Description

The system of equations is defined as:

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

c05pb 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. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is calculated, but it is not recalculated 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 Parameters

#### 5.1 Compulsory Input Parameters

- 1: **fcn** – string containing name of m-file

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  $x$ .

Its specification is:

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

#### Input Parameters

- 1: **n** – int32 scalar

$n$ , the number of equations.

- 2: **x(n)** – double array

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

- 3: **fvec(n)** – double array

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

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

4: **fjac(ldfjac,n) – double array**

**ldfjac**, the first dimension of the array, must be at least **n**.

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

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  $i,j = 1,2,\dots,n$ , (unless **iflag** is set to a negative value by **fcn**).

5: **ldfjac – int32 scalar**

6: **iflag – int32 scalar**

**iflag** = 1 or 2.

**iflag** = 1

**fvec** is to be updated.

**iflag** = 2

**fjac** is to be updated.

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. This value will be returned through **ifail**.

### Output Parameters

1: **fvec(n) – double array**

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

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(ldfjac,n) – double array**

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

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  $i,j = 1,2,\dots,n$ , (unless **iflag** is set to a negative value by **fcn**).

3: **iflag – int32 scalar**

**iflag** = 1 or 2.

**iflag** = 1

**fvec** is to be updated.

**iflag** = 2

**fjac** is to be updated.

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. This value will be returned through **ifail**.

2: **x(n) – double array**

An initial guess at the solution vector.

## 5.2 Optional Input Parameters

1: **n – int32 scalar**

*Default:* The dimension of the arrays **x**, **fvec**, **fjac**. (An error is raised if these dimensions are not equal.)

*n*, the number of equations.

*Constraint:* **n** > 0.

2: **xtol – double scalar**

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

*Suggested value:* the square root of the *machine precision*.

*Default:*  $\sqrt{\epsilon}$

*Constraint:* **xtol** ≥ 0.0.

## 5.3 Input Parameters Omitted from the MATLAB Interface

ldfjac, wa, lwa

## 5.4 Output Parameters

1: **x(n) – double array**

The final estimate of the solution vector.

2: **fvec(n) – double array**

The function values at the final point, **x**.

3: **fjac(ldfjac,n) – double array**

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

4: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

## 6 Error Indicators and Warnings

Errors or warnings detected by the function:

**ifail** < 0

A negative value of **ifail** indicates an exit from c05pb because you have set **iflag** negative in user-supplied (sub)program **fcn**. The value of **ifail** will be the same as your setting of **iflag**.

**ifail** = 1

On entry, **n** ≤ 0,  
or **ldfjac** < **n**,  
or **xtol** < 0.0,  
or **lwa** < **n** × (**n** + 13)/2.

**ifail** = 2

There have been  $100 \times (\mathbf{n} + 1)$  evaluations of the functions. Consider restarting the calculation from the final point held in **x**.

**ifail** = 3

No further improvement in the approximate solution **x** is possible; **xtol** is too small.

**ifail** = 4

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 c05pb from a different starting point may avoid the region of difficulty.

## 7 Accuracy

If  $\hat{x}$  is the true solution, c05pb tries to ensure that

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

If this condition is satisfied with  $\mathbf{xtol} = 10^{-k}$ , 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 c05pb usually avoids 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 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 c05pb may incorrectly indicate convergence. The coding of the Jacobian can be checked using c05za. If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning c05pb with a tighter tolerance.

## 8 Further Comments

The time required by c05pb 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 c05pb is about  $11.5 \times n^2$  to process each evaluation of the functions and about  $1.3 \times n^3$  to process each evaluation of the Jacobian. Unless user-supplied (sub)program **fcn** can be evaluated quickly, the timing of c05pb will be strongly influenced by the time spent in **fcn**.

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

## 9 Example

```
c05pb_fcn.m

function [fvec, fjac, iflag] = c05pb_fcn(n,x,fvec,fjac,ldfjac,iflag)

    if (iflag ~= 2)
        for k = 1:n
            fvec(k) = (3.0-2.0*x(k))*x(k) + 1.0;
            if (k > 1)
                fvec(k) = fvec(k) - x(k-1);
            end
            if (k < n)
                fvec(k) = fvec(k) - 2.0*x(k+1);
            end
        end
    end
```

```

else
    fjac = zeros(n,n);
    for k = 1:n
        fjac(k,k) = 3.0 - 4.0*x(k);
        if (k > 1)
            fjac(k,k-1) = -1.0;
        end
        if (k < n)
            fjac(k,k+1) = -2.0;
        end
    end
end
end

```

```

x = [-1;
     -1;
     -1;
     -1;
     -1;
     -1;
     -1;
     -1;
     -1;
     -1];
[xOut, fvec, fjac, ifail] = c05pb('c05pb_fcn', x)

```

```

xOut =
-0.5707
-0.6816
-0.7017
-0.7042
-0.7014
-0.6919
-0.6658
-0.5960
-0.4164
fvec =
1.0e-08 *
0.6560
-0.4175
-0.5193
-0.2396
0.2022
0.4818
0.2579
-0.3884
-0.0136
fjac =
Columns 1 through 7
-0.9691    -0.2148    -0.0209     0.0470     0.0611     0.0339    -0.0188
0.2268    -0.9561    -0.1558     0.0078     0.0423     0.0328    -0.0064
-0.0174     0.1584    -0.9720    -0.1533    -0.0206     0.0090     0.0087
-0.0478    -0.0482     0.1554    -0.9653    -0.1728    -0.0306     0.0139
-0.0414    -0.0486    -0.0103     0.1910    -0.9579    -0.1638    -0.0068
-0.0072    -0.0143    -0.0058     0.0102     0.1882    -0.9588    -0.1373
0.0361     0.0408     0.0197     0.0011    -0.0011     0.1731    -0.9455
0.0591     0.0849     0.0606     0.0302     0.0175     0.0320     0.2366
0.0164     0.0304     0.0474     0.0704     0.1040     0.1402     0.1748
Columns 8 through 9
-0.0678    -0.0471
-0.0621    -0.0582
-0.0254    -0.0720
0.0072    -0.0918
0.0214    -0.1198
-0.0024    -0.1614
-0.1429    -0.2288
-0.8900    -0.3679
0.4218    -0.8676
ifail =

0

```

