# NAG Toolbox for MATLAB

# d02ra

## 1 Purpose

d02ra solves the two-point boundary-value problem with general boundary conditions for a system of ordinary differential equations, using a deferred correction technique and Newton iteration.

## 2 Syntax

```
[np, x, y, abt, deleps, ifail] = d02ra(n, np, numbeg, nummix, tol, init,
x, y, fcn, g, ijac, jacobf, jacobg, deleps, jaceps, jacgep, 'mnp', mnp)
```

## 3 Description

d02ra solves a two-point boundary-value problem for a system of $n$ ordinary differential equations in the interval $(a,b)$ with $b > a$. The system is written in the form

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n \tag{1}$$

and the derivatives $f_i$ are evaluated by a user-supplied (sub)program **fcn**. With the differential equations (1) must be given a system of $n$ (nonlinear) boundary conditions

$$g_i(y(a), y(b)) = 0, \qquad i = 1, 2, \ldots, n,$$

where

$$y(x) = [y_1(x), y_2(x), \ldots, y_n(x)]^{\mathrm{T}}. \tag{2}$$

The functions $g_i$ are evaluated by a user-supplied (sub)program **g**. The solution is computed using a finite-difference technique with deferred correction allied to a Newton iteration to solve the finite-difference equations. The technique used is described fully in Pereyra 1979.

You must supply an absolute error tolerance and may also supply an initial mesh for the finite-difference equations and an initial approximate solution (alternatively a default mesh and approximation are used). The approximate solution is corrected using Newton iteration and deferred correction. Then, additional points are added to the mesh and the solution is recomputed with the aim of making the error everywhere less than your tolerance and of approximately equidistributing the error on the final mesh. The solution is returned on this final mesh.

If the solution is required at a few specific points then these should be included in the initial mesh. If, on the other hand, the solution is required at several specific points then you should use the interpolation functions provided in Chapter E01 if these points do not themselves form a convenient mesh.

The Newton iteration requires Jacobian matrices

$$\left(\frac{\partial f_i}{\partial y_j}\right), \left(\frac{\partial g_i}{\partial y_j(a)}\right) \text{and} \left(\frac{\partial g_i}{\partial y_j(b)}\right).$$

These may be supplied by you through (sub)program **jacobf** for $\left(\frac{\partial f_i}{\partial y_j}\right)$ and (sub)program **jacobg** for the others. Alternatively the Jacobians may be calculated by numerical differentiation using the algorithm described in Curtis *et al.* 1974.

For problems of the type (1) and (2) for which it is difficult to determine an initial approximation from which the Newton iteration will converge, a continuation facility is provided. You must set up a family of problems

$$y' = f(x, y, \epsilon), \qquad g(y(a), y(b), \epsilon) = 0, \tag{3}$$

where $f = [f_1, f_2, \ldots, f_n]^{\mathrm{T}}$ etc., and where $\epsilon$ is a continuation parameter. The choice $\epsilon = 0$ must give a

problem (3) which is easy to solve and $\epsilon = 1$ must define the problem whose solution is actually required. The function solves a sequence of problems with $\epsilon$ values

$$0 = \epsilon_1 < \epsilon_2 < \cdots < \epsilon_p = 1. \tag{4}$$

The number $p$ and the values $\epsilon_i$ are chosen by the function so that each problem can be solved using the solution of its predecessor as a starting approximation. Jacobians $\frac{\partial f}{\partial \epsilon}$ and $\frac{\partial g}{\partial \epsilon}$ are required and they may be supplied by you via the user-supplied (sub)programs **jaceps** and **jacgep** respectively or may be computed by numerical differentiation.

# 4 References

Curtis A R, Powell M J D and Reid J K 1974 On the estimation of sparse Jacobian matrices *J. Inst. Maths. Applics.* **13** 117–119

Pereyra V 1979 PASVA3: An adaptive finite-difference Fortran program for first order nonlinear, ordinary boundary problems *Codes for Boundary Value Problems in Ordinary Differential Equations. Lecture Notes in Computer Science* (ed B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

# 5 Parameters

## 5.1 Compulsory Input Parameters

1: **n – int32 scalar**

$n$, the number of differential equations.

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

2: **np – int32 scalar**

Must be set to the number of points to be used in the initial mesh.

*Constraint*: $4 \leq \mathbf{np} \leq \mathbf{mnp}$.

3: **numbeg – int32 scalar**

The number of left-hand boundary conditions (that is the number involving $y(a)$ only).

*Constraint*: $0 \leq \mathbf{numbeg} < \mathbf{n}$.

4: **nummix – int32 scalar**

The number of coupled boundary conditions (that is the number involving both $y(a)$ and $y(b)$).

*Constraint*: $0 \leq \mathbf{nummix} \leq \mathbf{n} - \mathbf{numbeg}$.

5: **tol – double scalar**

A positive absolute error tolerance. If

$$a = x_1 < x_2 < \cdots < x_{\mathbf{np}} = b$$

is the final mesh, $z_j(x_i)$ is the $j$th component of the approximate solution at $x_i$, and $y_j(x)$ is the $j$th component of the true solution of (1) and (2), then, except in extreme circumstances, it is expected that

$$\left|z_j(x_i) - y_j(x_i)\right| \leq \mathbf{tol}, \qquad i = 1, 2, \ldots, \mathbf{np}; j = 1, 2, \ldots, n. \tag{5}$$

*Constraint*: $\mathbf{tol} > 0.0$.

6: **init – int32 scalar**

Indicates whether you wish to supply an initial mesh and approximate solution (**init** $= 1$) or whether default values are to be used, (**init** $= 0$).

*Constraint*: **init** $= 0$ or $1$.

7: **x(mnp) – double array**

You must set $\mathbf{x}(1) = a$ and $\mathbf{x}(\mathbf{np}) = b$. If **init** $= 0$ on entry a default equispaced mesh will be used, otherwise you must specify a mesh by setting $\mathbf{x}(i) = x_i$, for $i = 2, 3, \ldots \mathbf{np} - 1$.

*Constraints*:

if **init** $= 0$, $\mathbf{x}(1) < \mathbf{x}(\mathbf{np})$;
if **init** $\neq 0$, $\mathbf{x}(1) < \mathbf{x}(2) < \cdots < \mathbf{x}(\mathbf{np})$.

8: **y(ldy,mnp) – double array**

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

If **init** $= 0$, then **y** need not be set.

If **init** $\neq 0$, then the array **y** must contain an initial approximation to the solution such that $\mathbf{y}(j, i)$ contains an approximation to

$$y_j(x_i), \qquad i = 1, 2, \ldots, \mathbf{np}; j = 1, 2, \ldots, n.$$

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

**fcn** must evaluate the functions $f_i$ (i.e., the derivatives $y_i'$) at a general point $x$ for a given value of $\epsilon$, the continuation parameter (see Section 3).

Its specification is:

```
[f] = fcn(x, eps, y, n)
```

**Input Parameters**

1: **x – double scalar**

The value of the argument $x$.

2: **eps – double scalar**

$\epsilon$, the value of the continuation parameter. This is 1 if continuation is not being used.

3: **y(n) – double array**

The value of the argument $y_i$, for $i = 1, 2, \ldots, n$.

4: **n – int32 scalar**

The number of equations.

**Output Parameters**

1: **f(n) – double array**

The values of $f_i$, for $i = 1, 2, \ldots, n$.

10: **g – string containing name of m-file**

**g** must evaluate the boundary conditions in equation (3) and place them in the array **bc**.

Its specification is:

```
[bc] = g(eps, ya, yb, n)
```

**Input Parameters**

1: **eps – double scalar**

$\epsilon$, the value of the continuation parameter. This is 1 if continuation is not being used.

2: **ya(n) – double array**

The value $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: **yb(n) – double array**

The value $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: **n – int32 scalar**

$n$, the number of equations.

**Output Parameters**

1: **bc(n) – double array**

The values $g_i(y(a), y(b), \epsilon)$, for $i = 1, 2, \ldots, n$. These must be ordered as follows:

(i) first, the conditions involving only $y(a)$ (see **numbeg**);

(ii) next, the **nummix** coupled conditions involving both $y(a)$ and $y(b)$ (see **nummix**); and,

(iii) finally, the conditions involving only $y(b)$ (**n** − **numbeg** − **nummix**).

11: **ijac – int32 scalar**

Indicates whether or not you are supplying Jacobian evaluation functions.

**ijac** $\neq 0$

You must supply user-supplied (sub)programs **jacobf** and **jacobg** and also, when continuation is used, user-supplied (sub)programs **jaceps** and **jacgep**.

**ijac** $= 0$

Numerical differentiation is used to calculate the Jacobian and the functions **d02gaz d02gay d02gaz d02gax** respectively may be used as the dummy parameters.

12: **jacobf – string containing name of m-file**

**jacobf** must evaluate the Jacobian $\left(\frac{\partial f_i}{\partial y_j}\right)$, for $i, j = 1, 2, \ldots, n$, given $x$ and $y_j$, for $j = 1, 2, \ldots, n$.

If **ijac** $= 0$, then numerical differentiation is used to calculate the Jacobian and the function **d02gaz** may be substituted for this parameter.

Its specification is:

```
[f] = jacobf(x, eps, y, n)
```

**Input Parameters**

1: **x – double scalar**

The value of the argument $x$.

2: **eps – double scalar**

The value of the continuation parameter $\epsilon$. This is 1 if continuation is not being used.

3: **y(n) – double array**

The value of the argument $y_i$, for $i = 1, 2, \ldots, n$.

4: **n – int32 scalar**

$n$, the number of equations.

**Output Parameters**

1: **f(n,n) – double array**

**f**$(i,j)$ must be set to the value of $\dfrac{\partial f_i}{\partial y_j}$, evaluated at the point $(x, y)$, for $i,j = 1, 2, \ldots, n$.

13: **jacobg – string containing name of m-file**

**jacobg** must evaluate the Jacobians $\left(\dfrac{\partial g_i}{\partial y_j(a)}\right)$ and $\left(\dfrac{\partial g_i}{\partial y_j(b)}\right)$. The ordering of the rows of **aj** and **bj** must correspond to the ordering of the boundary conditions described in the specification of user-supplied (sub)program **g**.

If **ijac** $= 0$, then numerical differentiation is used to calculate the Jacobian and the function **d02gay** may be substituted for this parameter.

Its specification is:

```
[aj, bj] = jacobg(eps, ya, yb, n)
```

**Input Parameters**

1: **eps – double scalar**

$\epsilon$, the value of the continuation parameter. This is 1 if continuation is not being used.

2: **ya(n) – double array**

The value $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: **yb(n) – double array**

The value $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: **n – int32 scalar**

$n$, the number of equations.

**Output Parameters**

1: **aj(n,n) – double array**

**aj**$(i,j)$ must be set to the value $\dfrac{\partial g_i}{\partial y_j(a)}$, for $i,j = 1, 2, \ldots, n$.

2: **bj(n,n) – double array**

**bj**$(i,j)$ must be set to the value $\dfrac{\partial g_i}{\partial y_j(b)}$, for $i,j = 1, 2 \ldots, n$.

14: **deleps – double scalar**

Must be given a value which specifies whether continuation is required. If **deleps** $\leq 0.0$ or **deleps** $\geq 1.0$ then it is assumed that continuation is not required. If $0.0 <$ **deleps** $< 1.0$ then it is assumed that continuation is required unless **deleps** $< \sqrt{\textit{machine precision}}$ when an error exit is taken. **deleps** is used as the increment $\epsilon_2 - \epsilon_1$ (see (4))) and the choice **deleps** $= 0.1$ is recommended.

15: **jaceps – string containing name of m-file**

**jaceps** must evaluate the derivative $\dfrac{\partial f_i}{\partial \epsilon}$ given $x$ and $y$ if continuation is being used.

If **ijac** $= 0$, then numerical differentiation is used to calculate the Jacobian and the function **d02gaz** may be substituted for this parameter.

Its specification is:

```
[f] = jaceps(x, eps, y, n)
```

**Input Parameters**

1: **x – double scalar**

The value of the argument $x$.

2: **eps – double scalar**

$\epsilon$, the value of the continuation parameter.

3: **y(n) – double array**

The solution values $y_i$ at the point $x$, for $i = 1, 2, \ldots, n$.

4: **n – int32 scalar**

$n$, the number of equations.

**Output Parameters**

1: **f(n) – double array**

**f**$(i)$ must contain the value $\dfrac{\partial f_i}{\partial \epsilon}$ at the point $(x, y)$, for $i = 1, 2, \ldots, n$.

16: **jacgep – string containing name of m-file**

**jacgep** must evaluate the derivatives $\dfrac{\partial g_i}{\partial \epsilon}$ if continuation is being used.

If **ijac** $= 0$, then numerical differentiation is used to calculate the Jacobian and the function **d02gax** may be substituted for this parameter.

Its specification is:

```
[bcep] = jacgep(eps, ya, yb, n)
```

**Input Parameters**

1: **eps – double scalar**

$\epsilon$, the value of the continuation parameter.

2: **ya**(**n**) – **double array**

The value of $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: **yb**(**n**) – **double array**

The value of $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: **n** – **int32 scalar**

$n$, the number of equations.

**Output Parameters**

1: **bcep**(**n**) – **double array**

**bcep**$(i)$ must contain the value of $\dfrac{\partial g_i}{\partial \epsilon}$, for $i = 1, 2, \ldots, n$.

## 5.2 Optional Input Parameters

1: **mnp** – **int32 scalar**

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

**mnp** must be set to the maximum permitted number of points in the finite-difference mesh. If **lwork** or **liwork** are too small then internally **mnp** will be replaced by the maximum permitted by these values. (A warning message will be output if on entry **ifail** is set to obtain monitoring information.)

*Constraint*: **mnp** $\geq 32$.

## 5.3 Input Parameters Omitted from the MATLAB Interface

ldy, work, lwork, iwork, liwork

## 5.4 Output Parameters

1: **np** – **int32 scalar**

The number of points in the final mesh.

2: **x**(**mnp**) – **double array**

$\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(\mathbf{np})$ define the final mesh (with the returned value of **np**) and $\mathbf{x}(1) = a$ and $\mathbf{x}(\mathbf{np}) = b$.

3: **y**(**ldy,mnp**) – **double array**

The approximate solution $z_j(x_i)$ satisfying (5) on the final mesh, that is

$$\mathbf{y}(j, i) = z_j(x_i), \qquad i = 1, 2, \ldots, \mathbf{np}; j = 1, 2, \ldots, n,$$

where **np** is the number of points in the final mesh. If an error has occurred then **y** contains the latest approximation to the solution. The remaining columns of **y** are not used.

4: **abt**(**n**) – **double array**

**abt**$(i)$, for $i = 1, 2, \ldots, n$, holds the largest estimated error (in magnitude) of the $i$th component of the solution over all mesh points.

5: **deleps – double scalar**

An overestimate of the increment $\epsilon_p - \epsilon_{p-1}$ (in fact the value of the increment which would have been tried if the restriction $\epsilon_p = 1$ had not been imposed). If continuation was not requested then **deleps** $= 0.0$.

If continuation is not requested then the user-supplied (sub)programs **jaceps** and **jacgep** may be replaced by dummy actual parameters in the call to d02ra. (**d02gaz d02gax** respectively may be used as the dummy parameters.)

6: **ifail – int32 scalar**

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

## 6 Error Indicators and Warnings

**Note**: d02ra may return useful information for one or more of the following detected errors or warnings.

**ifail** $= 1$

One or more of the parameters **n**, **mnp**, **np**, **numbeg**, **nummix**, **tol**, **deleps**, **lwork** or **liwork** has been incorrectly set, or $\mathbf{x}(1) \geq \mathbf{x}(\mathbf{np})$ or the mesh points $\mathbf{x}(i)$ are not in strictly ascending order.

**ifail** $= 2$

A finer mesh is required for the accuracy requested; that is **mnp** is not large enough. This error exit normally occurs when the problem being solved is difficult (for example, there is a boundary layer) and high accuracy is requested. A poor initial choice of mesh points will make this error exit more likely.

**ifail** $= 3$

The Newton iteration has failed to converge. There are several possible causes for this error:

(i) faulty coding in one of the Jacobian calculation functions;

(ii) if **ijac** $= 0$ then inaccurate Jacobians may have been calculated numerically (this is a very unlikely cause); or,

(iii) a poor initial mesh or initial approximate solution has been selected either by you or by default or there are not enough points in the initial mesh. Possibly, you should try the continuation facility.

**ifail** $= 4$

The Newton iteration has reached round-off error level. It could be however that the answer returned is satisfactory. The error is likely to occur if too high an accuracy is requested.

**ifail** $= 5$

The Jacobian calculated by (sub)program **jacobg** (or the equivalent matrix calculated by numerical differentiation) is singular. This may occur due to faulty coding of **jacobg** or, in some circumstances, to a zero initial choice of approximate solution (such as is chosen when **init** $= 0$).

**ifail** $= 6$

There is no dependence on $\epsilon$ when continuation is being used. This can be due to faulty coding of (sub)program **jaceps** or (sub)program **jacgep** or, in some circumstances, to a zero initial choice of approximate solution (such as is chosen when **init** $= 0$).

**ifail** $= 7$

**deleps** is required to be less than ***machine precision*** for continuation to proceed. It is likely that either the problem (3) has no solution for some value near the current value of $\epsilon$ (see the advisory print out from d02ra) or that the problem is so difficult that even with continuation it is unlikely to

be solved using this function. If the latter cause is suspected then using more mesh points initially may help.

**ifail** = 8 (d02ra)
**ifail** = 9 (**d02rar**)

A serious error has occurred in a call in the specified function. Check all array subscripts and (sub)program parameter lists in calls to d02ra. Seek expert help.

## 7 Accuracy

The solution returned by the function will be accurate to your tolerance as defined by the relation (5) except in extreme circumstances. The final error estimate over the whole mesh for each component is given in the array **abt**. If too many points are specified in the initial mesh, the solution may be more accurate than requested and the error may not be approximately equidistributed.

## 8 Further Comments

There are too many factors present to quantify the timing. The time taken by d02ra is negligible only on very simple problems.

You are strongly recommended to set **ifail** to obtain self-explanatory error messages, and also monitoring information about the course of the computation.

In the case where you wish to solve a sequence of similar problems, the use of the final mesh and solution from one case as the initial mesh is strongly recommended for the next.

## 9 Example

d02ra_fcn.m

```
function f = fcn(x, eps, y, n)
      f=zeros(n,1);
      f(1) = y(2);
      f(2) = y(3);
      f(3) = -y(1)*y(3) - 2.0d0*(1.0d0-y(2)*y(2))*eps;
```

d02ra_g.m

```
function bc = g(eps, ya, yb, n)
      bc=zeros(n,1);
      bc(1) = ya(1);
      bc(2) = ya(2);
      bc(3) = yb(2) - 1;
```

d02ra_jaceps.m

```
function f = jaceps(x, eps, y, n)
      f=zeros(n,1);
      f(1) = 0.0d0;
      f(2) = 0.0d0;
      f(3) = -2.0d0*(1.0d0-y(2)*y(2));
```

d02ra_jacgep.m

```
function bcep = jacgep(eps, ya, yb, n)
  bcep=zeros(n,1);
```

d02ra_jacobf.m

```
function f = jacobf(x, eps, y, n)
      f=zeros(n,n);
      f(1,2) = 1.0d0;
      f(2,3) = 1.0d0;
      f(3,1) = -y(3);
      f(3,2) = 4.0d0*y(2)*eps;
      f(3,3) = -y(1);
```

d02ra_jacobg.m

```
function [aj, bj] = jacobg(eps, ya, yb, n)
      aj=zeros(n,n);
      bj=zeros(n,n);
      aj(1,1) = 1.0D0;
      aj(2,2) = 1.0D0;
      bj(3,2) = 1.0D0;
```

```
n = int32(3);
np = int32(17);
numbeg = int32(2);
nummix = int32(0);
tol = 0.0001;
init = int32(0);
x = zeros(40,1);
x(np) = 10;
y = zeros(3,40);
ijac = int32(1);
deleps = 0.1;
[npOut, xOut, yOut, abt, delepsOut, ifail] = ...
    d02ra(n, np, numbeg, nummix, tol, init, x, y, 'd02ra_fcn', 'd02ra_g',
...
        ijac, 'd02ra_jacobf', 'd02ra_jacobg', deleps, 'd02ra_jaceps',
'd02ra_jacgep')
```

```
npOut =
          33
xOut =
         0
    0.0625
    0.1250
    0.1875
    0.2500
    0.3750
    0.5000
    0.6250
    0.7031
    0.7812
    0.9375
    1.0938
    1.2500
    1.4583
    1.6667
    1.8750
    2.0312
    2.1875
    2.5000
    2.6562
    2.8125
    3.1250
    3.7500
    4.3750
    5.0000
    5.6250
```

```
    6.2500
    6.8750
    7.5000
    8.1250
    8.7500
    9.3750
   10.0000
         0
         0
         0
         0
         0
         0
         0
yOut =
  Columns 1 through 7
         0    0.0032    0.0125    0.0275    0.0476    0.1015    0.1709
         0    0.1016    0.1954    0.2816    0.3605    0.4976    0.6097
    1.6872    1.5626    1.4398    1.3203    1.2054    0.9924    0.8048
  Columns 8 through 14
    0.2530    0.3095    0.3695    0.4978    0.6346    0.7776    0.9748
    0.6999    0.7467    0.7871    0.8513    0.8977    0.9308    0.9598
    0.6438    0.5563    0.4784    0.3490    0.2502    0.1763    0.1077
  Columns 15 through 21
    1.1768    1.3815    1.5362    1.6915    2.0031    2.1591    2.3153
    0.9773    0.9876    0.9922    0.9952    0.9983    0.9990    0.9994
    0.0639    0.0367    0.0238    0.0151    0.0058    0.0035    0.0021
  Columns 22 through 28
    2.6277    3.2526    3.8776    4.5026    5.1276    5.7526    6.3776
    0.9998    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    0.0007    0.0001    0.0000    0.0000   -0.0000    0.0000   -0.0000
  Columns 29 through 35
    7.0026    7.6276    8.2526    8.8776    9.5026         0         0
    1.0000    1.0000    1.0000    1.0000    1.0000         0         0
    0.0000   -0.0000    0.0000   -0.0000    0.0000         0         0
  Columns 36 through 40
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
abt =
   1.0e-04 *
    0.6924
    0.1805
    0.6421
delepsOut =
    0.8000
ifail =
           0
```