NAG Library Routine Document

D02RAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02RAF solves a 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 Specification

```
SUBROUTINE D02RAF (N, MNP, NP, NUMBEG, NUMMIX, TOL, INIT, X, Y, LDY,      &
                   ABT, FCN, G, IJAC, JACOBF, JACOBG, DELEPS, JACEPS,     &
                   JACGEP, WORK, LWORK, IWORK, LIWORK, IFAIL)
INTEGER            N, MNP, NP, NUMBEG, NUMMIX, INIT, LDY, IJAC, LWORK,    &
                   IWORK(LIWORK), LIWORK, IFAIL
REAL (KIND=nag_wp) TOL, X(MNP), Y(LDY,MNP), ABT(N), DELEPS,               &
                   WORK(LWORK)
EXTERNAL           FCN, G, JACOBF, JACOBG, JACEPS, JACGEP
```

3 Description

D02RAF 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

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

and the derivatives f_i are evaluated by FCN. With the differential equations (1) must be given a system of n (nonlinear) boundary conditions

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

where

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

The functions g_i are evaluated by 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 routines provided in Chapter E01 if these points do not themselves form a convenient mesh.

The Newton iteration requires Jacobian matrices

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

These may be supplied through JACOBF for $\left(\frac{\partial f_i}{\partial y_j}\right)$ and 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

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

where $f = [f_1, f_2, \ldots, f_n]^{\mathrm{T}}$ etc., and where ϵ 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 routine solves a sequence of problems with ϵ values

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

The number p and the values ϵ_i are chosen by the routine 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 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* (eds B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

5 Parameters

1: N – INTEGER *Input*

On entry: n, the number of differential equations.

Constraint: $\mathrm{N} > 0$.

2: MNP – INTEGER *Input*

On entry: 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: $\mathrm{MNP} \geq 32$.

3: NP – INTEGER *Input/Output*

On entry: must be set to the number of points to be used in the initial mesh.

Constraint: $4 \leq \mathrm{NP} \leq \mathrm{MNP}$.

On exit: the number of points in the final mesh.

4: NUMBEG – INTEGER *Input*

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

Constraint: $0 \leq \mathrm{NUMBEG} < \mathrm{N}$.

5: NUMMIX – INTEGER *Input*

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

Constraint: $0 \leq \text{NUMMIX} \leq \text{N} - \text{NUMBEG}$.

6: TOL – REAL (KIND=nag_wp) *Input*

On entry: a positive absolute error tolerance. If

$$a = x_1 < x_2 < \cdots < x_{\text{NP}} = b$$

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

$$\left|z_j(x_i) - y_j(x_i)\right| \leq \text{TOL}, \quad i = 1, 2, \ldots, \text{NP and } j = 1, 2, \ldots, n. \tag{5}$$

Constraint: $\text{TOL} > 0.0$.

7: INIT – INTEGER *Input*

On entry: indicates whether you wish to supply an initial mesh and approximate solution ($\text{INIT} = 1$) or whether default values are to be used, ($\text{INIT} = 0$).

Constraint: $\text{INIT} = 0$ or 1.

8: X(MNP) – REAL (KIND=nag_wp) array *Input/Output*

On entry: you must set $\text{X}(1) = a$ and $\text{X}(\text{NP}) = b$. If $\text{INIT} = 0$ on entry a default equispaced mesh will be used, otherwise you must specify a mesh by setting $\text{X}(i) = x_i$, for $i = 2, 3, \ldots, \text{NP} - 1$.

Constraints:

if $\text{INIT} = 0$, $\text{X}(1) < \text{X}(\text{NP})$;
if $\text{INIT} = 1$, $\text{X}(1) < \text{X}(2) < \cdots < \text{X}(\text{NP})$.

On exit: $\text{X}(1), \text{X}(2), \ldots, \text{X}(\text{NP})$ define the final mesh (with the returned value of NP) and $\text{X}(1) = a$ and $\text{X}(\text{NP}) = b$.

9: Y(LDY, MNP) – REAL (KIND=nag_wp) array *Input/Output*

On entry: if $\text{INIT} = 0$, then Y need not be set.

If $\text{INIT} = 1$, then the array Y must contain an initial approximation to the solution such that $\text{Y}(j, i)$ contains an approximation to

$$y_j(x_i), \quad i = 1, 2, \ldots, \text{NP and } j = 1, 2, \ldots, n.$$

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

$$\text{Y}(j, i) = z_j(x_i), \quad i = 1, 2, \ldots, \text{NP and } j = 1, 2, \ldots, n,$$

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.

10: LDY – INTEGER *Input*

On entry: the first dimension of the array Y as declared in the (sub)program from which D02RAF is called.

Constraint: $\text{LDY} \geq \text{N}$.

11: ABT(N) – REAL (KIND=nag_wp) array *Output*

On exit: $\text{ABT}(i)$, for $i = 1, 2, \ldots, n$, holds the largest estimated error (in magnitude) of the ith component of the solution over all mesh points.

12: FCN – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of FCN is:

```
SUBROUTINE FCN (X, EPS, Y, F, N)
INTEGER            N
REAL (KIND=nag_wp) X, EPS, Y(N), F(N)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the independent variable.

2: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter. This is 1 if continuation is not being used.

3: Y(N) – REAL (KIND=nag_wp) array *Input*

On entry: y_i, for $i = 1, 2, \ldots, n$, the values of the dependent variables at x.

4: F(N) – REAL (KIND=nag_wp) array *Output*

On exit: the values of the derivatives f_i evaluated at x given ϵ, for $i = 1, 2, \ldots, n$.

5: N – INTEGER *Input*

On entry: n, the number of equations.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: G – SUBROUTINE, supplied by the user. *External Procedure*

G must evaluate the boundary conditions in equation (3) and place them in the array BC.

The specification of G is:

```
SUBROUTINE G (EPS, YA, YB, BC, N)
INTEGER            N
REAL (KIND=nag_wp) EPS, YA(N), YB(N), BC(N)
```

1: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter. This is 1 if continuation is not being used.

2: YA(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: YB(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: BC(N) – REAL (KIND=nag_wp) array *Output*

On exit: 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).

5: N – INTEGER *Input*

On entry: n, the number of equations.

G must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: IJAC – INTEGER *Input*

On entry: indicates whether or not you are supplying Jacobian evaluation routines.

IJAC $\neq 0$
: You must supply JACOBF and JACOBG and also, when continuation is used, JACEPS and JACGEP.

IJAC $= 0$
: Numerical differentiation is used to calculate the Jacobian and the routines D02GAW, D02GAX, D02GAY and D02GAZ respectively may be used as the dummy parameters.

15: JACOBF – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

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

If all Jacobians are to be approximated internally by numerical differentiation then it must be replaced by the NAG defined null function pointer NULLFN.

The specification of JACOBF is:

```
SUBROUTINE JACOBF (X, EPS, Y, F, N)
INTEGER            N
REAL (KIND=nag_wp) X, EPS, Y(N), F(N,N)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the independent variable.

2: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter. This is 1 if continuation is not being used.

3: Y(N) – REAL (KIND=nag_wp) array *Input*

On entry: y_i, for $i = 1, 2, \ldots, n$, the values of the dependent variables at x.

4: F(N, N) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{F}(j, i)$ must be set to the value of $\frac{\partial f_i}{\partial y_j}$, evaluated at the point (x, y), for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

5: N – INTEGER *Input*

On entry: n, the number of equations.

JACOBF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: JACOBG – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

JACOBG evaluates the Jacobians $\left(\frac{\partial g_i}{\partial y_j(a)}\right)$ and $\left(\frac{\partial g_i}{\partial y_j(b)}\right)$. The ordering of the rows of AJ and BJ must correspond to the ordering of the boundary conditions described in the specification of G.

If all Jacobians are to be approximated internally by numerical differentiation then it must be replaced by the NAG defined null function pointer NULLFN.

The specification of JACOBG is:

```
SUBROUTINE JACOBG (EPS, YA, YB, AJ, BJ, N)
INTEGER            N
REAL (KIND=nag_wp) EPS, YA(N), YB(N), AJ(N,N), BJ(N,N)
```

1: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter. This is 1 if continuation is not being used.

2: YA(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: YB(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: AJ(N, N) – REAL (KIND=nag_wp) array *Output*

On exit: AJ(i, j) must be set to the value $\frac{\partial g_i}{\partial y_j(a)}$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

5: BJ(N, N) – REAL (KIND=nag_wp) array *Output*

On exit: BJ(i, j) must be set to the value $\frac{\partial g_i}{\partial y_j(b)}$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

6: N – INTEGER *Input*

On entry: n, the number of equations.

JACOBG must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

17: DELEPS – REAL (KIND=nag_wp) *Input/Output*

On entry: must be given a value which specifies whether continuation is required. If DELEPS ≤ 0.0 or DELEPS ≥ 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.

On exit: 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 JACEPS and JACGEP may each be replaced by dummy actual parameters in the call to D02RAF. (D02GAW and D02GAX respectively may be used as the dummy parameters.)

18: JACEPS – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

JACEPS evaluates the derivative $\dfrac{\partial f_i}{\partial \epsilon}$ given x and y if continuation is being used.

If all Jacobians (derivatives) are to be approximated internally by numerical differentiation, or continuation is not being used, then it must be replaced by the NAG defined null function pointer NULLFN.

The specification of JACEPS is:

```
SUBROUTINE JACEPS (X, EPS, Y, F, N)
INTEGER             N
REAL (KIND=nag_wp) X, EPS, Y(N), F(N)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the independent variable.

2: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter.

3: Y(N) – REAL (KIND=nag_wp) array *Input*

On entry: the solution values y_i, for $i = 1, 2, \dots, n$, at the point x.

4: F(N) – REAL (KIND=nag_wp) array *Output*

On exit: F(i) must contain the value $\dfrac{\partial f_i}{\partial \epsilon}$ at the point (x, y), for $i = 1, 2, \dots, n$.

5: N – INTEGER *Input*

On entry: n, the number of equations.

JACEPS must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

19: JACGEP – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

JACGEP evaluates the derivatives $\dfrac{\partial g_i}{\partial \epsilon}$ if continuation is being used.

If all Jacobians (derivatives) are to be approximated internally by numerical differentiation, or continuation is not being used, then it must be replaced by the NAG defined null function pointer NULLFN.

The specification of JACGEP is:

```
SUBROUTINE JACGEP (EPS, YA, YB, BCEP, N)
INTEGER             N
REAL (KIND=nag_wp) EPS, YA(N), YB(N), BCEP(N)
```

1: EPS – REAL (KIND=nag_wp) *Input*

On entry: ϵ, the value of the continuation parameter.

2: YA(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value of $y_i(a)$, for $i = 1, 2, \ldots, n$.

3: YB(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value of $y_i(b)$, for $i = 1, 2, \ldots, n$.

4: BCEP(N) – REAL (KIND=nag_wp) array *Output*

On exit: BCEP(i) must contain the value of $\frac{\partial g_i}{\partial \epsilon}$, for $i = 1, 2, \ldots, n$.

5: N – INTEGER *Input*

On entry: n, the number of equations.

JACGEP must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

20: WORK(LWORK) – REAL (KIND=nag_wp) array *Workspace*

21: LWORK – INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D02RAF is called.

Constraint: $\mathrm{LWORK} \geq \mathrm{MNP} \times \left(3\mathrm{N}^2 + 6\mathrm{N} + 2\right) + 4\mathrm{N}^2 + 3\mathrm{N}$.

22: IWORK(LIWORK) – INTEGER array *Workspace*

23: LIWORK – INTEGER *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02RAF is called.

Constraints:

if $\mathrm{IJAC} \neq 0$, $\mathrm{LIWORK} \geq \mathrm{MNP} \times (2 \times \mathrm{N} + 1) + \mathrm{N}$;
if $\mathrm{IJAC} = 0$, $\mathrm{LIWORK} \geq \mathrm{MNP} \times (2 \times \mathrm{N} + 1) + \mathrm{N}^2 + 4 \times \mathrm{N} + 2$.

24: IFAIL – INTEGER *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Section 3.3 in the Essential Introduction).

On entry: IFAIL must be set to a value with the decimal expansion cba, where each of the decimal digits c, b and a must have a value of 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e., hard failure with all messages printed).

On exit: $\mathrm{IFAIL} = 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

One or more of the parameters N, MNP, NP, NUMBEG, NUMMIX, TOL, DELEPS, LWORK or LIWORK is incorrectly set, or $\mathrm{X}(1) \geq \mathrm{X}(\mathrm{NP})$ or the mesh points $\mathrm{X}(i)$ 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 routines;

(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 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 ϵ when continuation is being used. This can be due to faulty coding of JACEPS or 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 ϵ (see the advisory print out from D02RAF) or that the problem is so difficult that even with continuation it is unlikely to be solved using this routine. If the latter cause is suspected then using more mesh points initially may help.

IFAIL $= 8$
IFAIL $= 9$

A serious error has occurred in an internal call. Check all array subscripts and subroutine parameter lists in calls to D02RAF. Seek expert help.

IFAIL $= -99$

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

The solution returned by the routine 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 Parallelism and Performance

Not applicable.

9 Further Comments

There are too many factors present to quantify the timing. The time taken by D02RAF 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. Monitoring information is written to a logical advisory message unit which normally default to the same unit number as the error message unit (see Section 3.4 in the Essential Introduction for details); the advisory message unit number can be changed by calling X04ABF.

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.

10 Example

This example solves the differential equation

$$y''' = -yy'' - 2\epsilon\left(1 - y'^2\right)$$

with $\epsilon = 1$ and boundary conditions

$$y(0) = y'(0) = 0, \quad y'(10) = 1$$

to an accuracy specified by $\mathrm{TOL} = 1.0\mathrm{E}{-4}$. The continuation facility is used with the continuation parameter ϵ introduced as in the differential equation above and with $\mathrm{DELEPS} = 0.1$ initially. (The continuation facility is not needed for this problem and is used here for illustration.)

10.1 Program Text

```
!   D02RAF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module d02rafe_mod

!     D02RAF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                               :: fcn, g, jaceps, jacgep, jacobf,  &
                                              jacobg
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter        :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter        :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
      Integer, Parameter, Public           :: iset = 1, n = 3, nin = 5, nout = 6
    Contains
      Subroutine fcn(x,eps,y,f,n)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: eps, x
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(n)
        Real (Kind=nag_wp), Intent (In)      :: y(n)
!       .. Executable Statements ..
        f(1) = y(2)
        f(2) = y(3)
        f(3) = -y(1)*y(3) - two*(one-y(2)*y(2))*eps
        Return
      End Subroutine fcn

      Subroutine g(eps,ya,yb,bc,n)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: eps
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: bc(n)
        Real (Kind=nag_wp), Intent (In)      :: ya(n), yb(n)
!       .. Executable Statements ..
        bc(1) = ya(1)
        bc(2) = ya(2)
        bc(3) = yb(2) - one
        Return
      End Subroutine g

      Subroutine jaceps(x,eps,y,f,n)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: eps, x
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(n)
        Real (Kind=nag_wp), Intent (In)      :: y(n)
!       .. Executable Statements ..
        f(1:2) = zero
        f(3) = -two*(one-y(2)*y(2))
        Return
      End Subroutine jaceps

      Subroutine jacgep(eps,ya,yb,bcep,n)

!       .. Scalar Arguments ..
```

```
        Real (Kind=nag_wp), Intent (In)      :: eps
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: bcep(n)
        Real (Kind=nag_wp), Intent (In)      :: ya(n), yb(n)
!       .. Executable Statements ..
        bcep(1:n) = zero
        Return
      End Subroutine jacgep

      Subroutine jacobf(x,eps,y,f,n)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: eps, x
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(n,n)
        Real (Kind=nag_wp), Intent (In)      :: y(n)
!       .. Executable Statements ..
        f(1:n,1:n) = zero
        f(1,2) = one
        f(2,3) = one
        f(3,1) = -y(3)
        f(3,2) = two*two*y(2)*eps
        f(3,3) = -y(1)
        Return
      End Subroutine jacobf

      Subroutine jacobg(eps,ya,yb,aj,bj,n)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: eps
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: aj(n,n), bj(n,n)
        Real (Kind=nag_wp), Intent (In)      :: ya(n), yb(n)
!       .. Executable Statements ..
        aj(1:n,1:n) = zero
        bj(1:n,1:n) = zero
        aj(1,1) = one
        aj(2,2) = one
        bj(3,2) = one
        Return
      End Subroutine jacobg
    End Module d02rafe_mod

    Program d02rafe

!     D02RAF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02raf, nag_wp, x04abf
      Use d02rafe_mod, Only: fcn, g, iset, jaceps, jacgep, jacobf, jacobg, n,  &
                             nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: deleps, tol
      Integer                              :: ifail, ijac, init, j, ldy,       &
                                              liwork, lwork, mnp, np, numbeg,  &
                                              nummix, outchn
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: abt(:), work(:), x(:), y(:,:)
      Integer, Allocatable                 :: iwork(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02RAF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) mnp, np
      ldy = n
      liwork = mnp*(2*n+1) + n
```

```
      lwork = mnp*(3*n*n+6*n+2) + 4*n*n + 3*n
      Allocate (abt(n),work(lwork),x(mnp),y(ldy,mnp),iwork(liwork))

      outchn = nout
      Write (nout,*)
      Call x04abf(iset,outchn)
      Read (nin,*) tol, deleps
      Read (nin,*) init, ijac, numbeg, nummix
      Read (nin,*) x(1), x(np)

!     ifail: behaviour on error exit
!             =1 for quiet-soft exit
!     * Set IFAIL to 111 to obtain monitoring information *
      ifail = 1
      Call d02raf(n,mnp,np,numbeg,nummix,tol,init,x,y,ldy,abt,fcn,g,ijac, &
        jacobf,jacobg,deleps,jaceps,jacgep,work,lwork,iwork,liwork,ifail)

      If (ifail==0 .Or. ifail==4) Then
        Write (nout,*) 'Calculation using analytic Jacobians'
        If (ifail==4) Write (nout,99996) 'On exit from D02RAF IFAIL = 4'
        Write (nout,*)
        Write (nout,99999) 'Solution on final mesh of ', np, ' points'
        Write (nout,*) '      X(I)        Y1(I)        Y2(I)        Y3(I)'
        Write (nout,99998)(x(j),y(1:n,j),j=1,np)
        Write (nout,*)
        Write (nout,*) 'Maximum estimated error by components'
        Write (nout,99997) abt(1:n)
      Else
        Write (nout,99996) ' ** D02RAF returned with IFAIL = ', ifail
      End If

99999 Format (1X,A,I2,A)
99998 Format (1X,F10.3,3F13.4)
99997 Format (11X,1P,3E13.2)
99996 Format (1X,A,I5)
    End Program d02rafe
```

10.2 Program Data

```
D02RAF Example Program Data
   40  17              : max mesh size, initial mesh size
   1.0E-4 1.0E-1       : tol, deleps
   0  1  2  0          : init, ijac, numbeg, nummix
   0.0  10.0           : domain end-points
```

10.3 Program Results

```
 D02RAF Example Program Results

 Calculation using analytic Jacobians

 Solution on final mesh of 33 points
       X(I)        Y1(I)        Y2(I)        Y3(I)
      0.000       0.0000       0.0000       1.6872
      0.062       0.0032       0.1016       1.5626
      0.125       0.0125       0.1954       1.4398
      0.188       0.0275       0.2816       1.3203
      0.250       0.0476       0.3605       1.2054
      0.375       0.1015       0.4976       0.9924
      0.500       0.1709       0.6097       0.8048
      0.625       0.2530       0.6999       0.6438
      0.703       0.3095       0.7467       0.5563
      0.781       0.3695       0.7871       0.4784
      0.938       0.4978       0.8513       0.3490
      1.094       0.6346       0.8977       0.2502
      1.250       0.7776       0.9308       0.1763
      1.458       0.9748       0.9598       0.1077
      1.667       1.1768       0.9773       0.0639
      1.875       1.3815       0.9876       0.0367
      2.031       1.5362       0.9922       0.0238
```

```
   2.188        1.6915        0.9952        0.0151
   2.500        2.0031        0.9983        0.0058
   2.656        2.1591        0.9990        0.0035
   2.812        2.3153        0.9994        0.0021
   3.125        2.6277        0.9998        0.0007
   3.750        3.2526        1.0000        0.0001
   4.375        3.8776        1.0000        0.0000
   5.000        4.5026        1.0000        0.0000
   5.625        5.1276        1.0000       -0.0000
   6.250        5.7526        1.0000        0.0000
   6.875        6.3776        1.0000       -0.0000
   7.500        7.0026        1.0000        0.0000
   8.125        7.6276        1.0000       -0.0000
   8.750        8.2526        1.0000        0.0000
   9.375        8.8776        1.0000       -0.0000
  10.000        9.5026        1.0000        0.0000

 Maximum estimated error by components
                6.92E-05      1.81E-05      6.42E-05
```