nag_ode_bvp_fd_nonlin_gen (d02rac)

1. Purpose

nag_ode_bvp_fd_nonlin_gen (d02rac) 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. Specification

```
#include <nag.h>
#include <nagd02.h>

void d02rac(Integer neq, double *deleps,
            void (*fcn) (Integer neq, double x, double eps, double y[],
                         double f[], Nag_User *comm),
            Integer numbeg, Integer nummix,
            void (*g) (Integer neq, double eps, double ya[], double yb[],
                       double bc[], Nag_User *comm),
            Nag_MeshSet init, Integer mnp, Integer *np, double x[],
            double y[],
            double tol, double abt[],
            void (*jacobf) (Integer neq, double x, double eps, double y[],
                            double f[], Nag_User *comm),
            void (*jacobg) (Integer neq, double eps, double ya[],
                            double yb[], double aj[], double bj[],
                            Nag_User *comm),
            void (*jaceps) (Integer neq, double x, double eps, double y[],
                            double f[], Nag_User *comm),
            void (*jacgep) (Integer neq, double eps, double ya[], double yb[],
                            double bcep[], Nag_User *comm),
            Nag_User *comm, NagError *fail)
```

3. Description

This function solves a two-point boundary-value problem for a system of **neq** ordinary differential equations in the interval $[a, b]$ with $b > a$. The system is written in the form

$$y_i' = f_i(x, y_1, y_2, \ldots, y_{\mathbf{neq}}) \ , \ i = 1, 2, \ldots, \mathbf{neq} \tag{1}$$

and the derivatives f_i are evaluated by a function **fcn** supplied by the user. With the differential equations (1) must be given a system of **neq** (nonlinear) boundary conditions

$$g_i(y(a), y(b)) = 0 \ , \ i = 1, 2, \ldots, \mathbf{neq}$$

where

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

The functions g_i are evaluated by a function **g** supplied by the user. 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).

The user 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 a 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 the user's 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 the user 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

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

These may be supplied by the user through functions **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. The user must set up a family of problems

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

where $f = [f_1, f_2, \ldots, f_{\mathbf{neq}}]^T$ etc, and where ε is a continuation parameter. The choice $\varepsilon = 0$ must give a problem (3) which is easy to solve and $\varepsilon = 1$ must define the problem whose solution is actually required. The routine solves a sequence of problems with ε values

$$0 = \varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_p = 1. \tag{4}$$

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 $\frac{\partial f}{\partial \varepsilon}$ and $\frac{\partial g}{\partial \varepsilon}$ are required and they may be supplied by the user via routines **jaceps** and **jacgep** respectively or may be computed by numerical differentiation.

4. Parameters

neq

Input: the number of differential equations, **neq**.
Constraint: **neq** > 0.

deleps

Input: 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** $< \sqrt{machine\ precision}$ when an error exit is taken. **deleps** is used as the increment $\varepsilon_2 - \varepsilon_1$ (see (4)) and the choice **deleps** $= 0.1$ is recommended.
Output: an overestimate of the increment $\varepsilon_p - \varepsilon_{p-1}$ (in fact the value of the increment which would have been tried if the restriction $\varepsilon_p = 1$ had not been imposed). If continuation was not requested then **deleps** $= 0.0$.
If continuation is not requested then the parameters **jaceps** and **jacgep** may be replaced by the NAG defined two null functions macro, **NULL_2_FUN**.

fcn

The function **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:

```
void fcn (Integer neq, double x, double eps, double y[], double f[],
          Nag_User *comm)
```

neq
Input: the number of equations.

x
Input: the value of the argument x.

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

y[neq]
Input: $y[i-1]$ contains the value of the argument y_i, for $i = 1, 2, \ldots,$**neq**.

f[neq]
Output: $f[i-1]$ must contain the values of f_i, for $i = 1, 2, \ldots,$ **neq**.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

numbeg
Input: the number of left-hand boundary conditions (that is the number involving $y(a)$ only).
Constraint: $0 \leq$ **numbeg** $<$ **neq**.

nummix
Input: the number of coupled boundary conditions (that is the number involving both $y(a)$ and $y(b)$).
Constraint: $0 \leq$ **nummix** $\leq$ **neq** $-$ **numbeg**.

g
The function **g** must evaluate the boundary conditions in equation (3) and place them in the array **bc**.
The specification of **g** is:

```
void g (Integer neq, double eps, double ya[], double yb[], double bc[],
        Nag_User *comm)
```

neq
Input: the number of equations.

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

ya[neq]
Input: **ya**$[i-1]$ contains the value $y_i(a)$, for $i = 1, 2, \ldots,$ **neq**.

yb[neq]
Input: **yb**$[i-1]$ contains the value $y_i(b)$, for $i = 1, 2, \ldots,$ **neq**.

bc[neq]
Output: must contain the values $g_i(y(a), y(b), \varepsilon)$, for $i = 1, 2, \ldots,$**neq**. These must be ordered as follows:

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

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

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

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer

Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

init

Input: indicates whether the user wishes to supply an initial mesh and approximate solution (**init** = **Nag_UserInitMesh**) or whether default values are to be used, (**init** = **Nag_DefInitMesh**).
Constraint: **init** = **Nag_UserInitMesh** or **Nag_DefInitMesh**.

mnp

Input: must be set to the maximum permitted number of points in the finite-difference mesh.
Constraint: **mnp** ≥ 32.

np

Input: must be set to the number of points to be used in the initial mesh.
Constraint: $4 \leq$ **np** $\leq$ **mnp**.
Output: the number of points in the final mesh.

x[mnp]

Input: the user must set **x**[0] $= a$ and **x**[**np**-1] $= b$. If **init** = **Nag_DefInitMesh** on entry a default equispaced mesh will be used, otherwise the user must specify a mesh by setting **x**$[i-1] = x_i$, for $i = 2, 3, \ldots,$**np**-1.
Constraints: **x**[0] $<$ **x**[**np**-1], if **init** = **Nag_DefInitMesh**,
x[0] $<$ **x**[1] $< \ldots <$ **x**[**np**-1], if **init** = **Nag_UserInitMesh**.
Output: **x**[0],**x**[1],...,**x**[**np**-1] define the final mesh (with the returned value of **np**) and **x**[0] = a and **x**[**np**-1] $= b$.

y[neq][mnp]

Input: if **init** = **Nag_DefInitMesh**, then **y** need not be set.
If **init** = **Nag_UserInitMesh**, then the array **y** must contain an initial approximation to the solution such that **y**$[j-1][i-1]$ contains an approximation to
$y_j(x_i)$, $i = 1, 2, \ldots,$**np** $j = 1, 2, \ldots,$**neq**.
Output: the approximate solution $z_j(x_i)$ satisfying (5) on the final mesh, that is
y$[j-1][i-1] = z_j(x_i)$, $i = 1, 2, \ldots,$**np** $j = 1, 2, \ldots,$**neq**, 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.

tol

Input: a positive absolute error tolerance. If $a = x_1 < x_2 < \ldots < x_{\mathbf{np}} = b$ 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
$|z_j(x_i) - y_j(x_i)| \leq$ **tol** $i = 1, 2, \ldots,$**np**; $j = 1, 2, \ldots,$**neq**. (5)
Constraint: **tol** > 0.0.

abt[neq]

Output: **abt**$[i-1]$, for $i = 1, 2, \ldots,$**neq**, holds the largest estimated error (in magnitude) of the ith component of the solution over all mesh points.

jacobf

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

The specification of **jacobf** is:

```
void jacobf (Integer neq, double x, double eps, double y[], double f[],
             Nag_User *comm)
```

neq
Input: the number of equations.

x
Input: the value of the argument x.

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

y[neq]
Input: **y**$[i-1]$ contains the value of the argument y_i, for $i = 1, 2, \ldots,$**neq**.

f[neq*neq]
Output: **f**$[(i-1) * \mathbf{neq} + (j-1)]$ must be set to the value of $\dfrac{\partial f_i}{\partial y_j}$, evaluated at the point (x, y), for $i, j = 1, 2, \ldots,$**neq**.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

Note that if **jacobf** is supplied then **jacobg** (see below) must also be supplied. Note that if **jacobf** is supplied and continuation is requested then **jaceps** and **jacgep** (see below) must also be supplied.

jacobg

The function **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 function **g** above.
The specification of **jacobg** is:

```
void jacobg (Integer neq, double eps, double ya[], double yb[], double aj[],
             double bj[], Nag_User *comm)
```

neq
Input: the number of equations.

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

ya[neq]
Input: **ya**$[i-1]$ contains the value $y_i(a)$, for $i = 1, 2, \ldots,$ **neq**.

yb[neq]
Input: **yb**$[i-1]$ contains the value $y_i(b)$, for $i = 1, 2, \ldots,$ **neq**.

aj[**neq**∗**neq**]

Output: **aj**$[(i-1) * \mathbf{neq} + (j-1)]$ must be set to the value $\frac{\partial g_i}{\partial y_j(a)}$, for $i, j = 1, 2, \ldots,$**neq**.

bj[**neq**∗**neq**]

Output: **bj**$[(i-1) * \mathbf{neq} + (j-1)]$ must be set to the value $\frac{\partial g_i}{\partial y_j(b)}$, for $i, j = 1, 2 \ldots,$**neq**.

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer

Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

Note that if **jacobg** is supplied then **jacobf** (see above) must also be supplied. If numerical differentiation is to be used to calculate the Jacobian then **jacobf** and **jacobg** may be replaced by the NAG-defined two null functions macro, **NULL_2_FUN**.

jaceps

The function **jaceps** must evaluate the derivative $\frac{\partial f_i}{\partial \varepsilon}$ given x, y and ε if continuation is being used.

The specification of **jaceps** is:

```
void jaceps (Integer neq, double x, double eps, double y[], double f[],
             Nag_User *comm)
```

neq

Input: the number of equations.

x

Input: the value of the argument x.

eps

Input: the value of the continuation parameter, ε.

y[**neq**]

Input: **y**$[i-1]$ contains the solution values y_i at the point x, for $i = 1, 2, \ldots,$**neq**.

f[**neq**]

Output: **f**$[i-1]$ must contain $f(i)$, the value $\frac{\partial f_i}{\partial \varepsilon}$ at the point (x, y), given ε, for $i = 1, 2, \ldots,$**neq**.

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer

Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

Note that if **jaceps** is defined then **jacgep** (see below) must also be defined.

jacgep

The function **jacgep** must evaluate the derivatives $\frac{\partial g_i}{\partial \varepsilon}$ if continuation is being used.

The specification of **jacgep** is:

```
void jacgep (Integer neq, double eps, double ya[], double yb[],
             double bcep[], Nag_User *comm)
```

neq
Input: the number of equations.

eps
Input: the value of the continuation parameter, ε.

ya[neq]
Input: **ya**$[i-1]$ contains the value of $y_i(a)$, for $i = 1, 2, \ldots$**neq**.

yb[neq]
Input: **yb**$[i-1]$ contains the value of $y_i(b)$, for $i = 1, 2, \ldots$,**neq**.

bcep[neq]
Output: **bcep**$[i-1]$ must contain the value of $\dfrac{\partial g_i}{\partial \varepsilon}$, for $i = 1, 2, \ldots$,**neq**.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: The pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

Note that if **jacgep** is defined then **jaceps** (see above) must also be defined. If numerical differentiation is to be used to calculate the Jacobian and continuation is not required then **jacobf**, **jacobg**, **jaceps** and **jacgep** may be replaced by the NAG-defined four null functions macro, **NULL_4_FUN**.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: The pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined functions **fcn()**, **g()**, **jacobf()**, **jacobg()**, **jaceps()**, and **jacgep()**. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program, e.g. `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **neq** must not be less than 1: **neq** = ⟨*value*⟩.
On entry, **mnp** must not be less than 32: **mnp** = ⟨*value*⟩.

NE_REAL_ARG_LE
On entry, **tol** must not be less than or equal to 0.0: **tol** = ⟨*value*⟩.

NE_INT_RANGE_CONS
On entry, **np** = ⟨*value*⟩ and **mnp** = ⟨*value*⟩. The parameter **np** must satisfy $4 \leq$ **np** $\leq$ **mnp**.
On entry, **numbeg** = ⟨*value*⟩ and **neq** = ⟨*value*⟩. The parameter **numbeg** must satisfy $0 \leq$ **numbeg** $\leq$ **neq**.
On entry, **nummix** = ⟨*value*⟩ and **neq** − **numbeg** = ⟨*value*⟩. The parameter **nummix** must satisfy $0 \leq$ **nummix** $\leq$ **neq** − **numbeg**.

NE_2_INT_ARG_ZERO
On entry, **numbeg** = 0 and **nummix** = 0. These parameters must not both be zero.

NE_BAD_PARAM
On entry parameter **init** had an illegal value.

NE_2_REAL_ARG_LE
On entry $\mathbf{x}[\mathbf{np}-1] = \langle value \rangle$ while $\mathbf{x}[0] = \langle value \rangle$.
These parameters must satisfy $\mathbf{x}[\mathbf{np}-1] > \mathbf{x}[0]$

NE_NOT_STRICTLY_INCREASING
The sequence **x** is not strictly increasing: $\mathbf{x}[\langle value \rangle] = \langle value \rangle$, $\mathbf{x}[\langle value \rangle] = \langle value \rangle$.

NE_INVALID_FUN_JAC
Only one of **jacobf** or **jacobg** has been set to non-null possibly implying user-defined jacobians. Both must be non-null.

NE_INVALID_FUN_JAC_CONT
deleps has been set to $\langle value \rangle$ implying continuation and both **jacobf** and **jacobg** have been set to non-null implying user-defined jacobians. Hence the functions **jaceps** and **jacgep** must also be non-null.

NE_INVALID_FUN_JAC_NO_CONT
deleps has been set to $\langle value \rangle$ implying no continuation and both **jacobf** and **jacobg** have been set to non-null implying user-defined jacobians. Hence the functions **jaceps** and **jacgep** must be null.

NE_INVALID_FUN_NO_JAC_CONT
deleps has been set to $\langle value \rangle$ implying continuation and both **jacobf** and **jacobg** have been set to null implying no user-defined jacobians. Hence the functions **jaceps** and **jacgep** must also be null.

NE_ALLOC_FAIL
Memory allocation failed.

NE_CONV_MESH
A finer mesh is required for the accuracy requested; that is **mnp** is not large enough.

NE_CONV_CONT
Convergence failure. There are a number of possible causes.
a) Faulty coding of the Jacobian calculation functions.
b) If Jacobians have not been supplied then inaccurate Jacobians have been calculated internally (not very likely).
c) A poor choice of initial mesh or initial starting conditions either by the user or by default. Try using the continuation facility.

NE_CONV_ROUNDOFF
Solution cannot be improved due to roundoff error. Too much accuracy might have been requested.

NE_CONV_CONT_DEP
There is no dependence on epsilon when continuation is being used. This may 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**=**Nag_DefInitMesh**).

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

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

NE_CONV_CONT_DELEPS
deleps is required to be less than machine precision for continuation to proceed. It is likely that either the problem has no solution for some value near the current value of ε or that the problem is so difficult that even with continuation it is unlikely to be solved using this function. Using more mesh points may help.

6. Further Comments

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

In the case where the user wishes 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.

6.1. Accuracy

The solution returned by the function will be accurate to the user's 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.

6.2. References

Curtis AR, Powell MJD and Reid JK (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. In 'Codes for Boundary Value Problems in Ordinary Differential Equations'. Lecture Notes in Computer Science. *(ed B Childs, M Scott JW Daniel, E Denman and P Nelson)* **76** Springer-Verlag.

7. See Also

nag_ode_bvp_fd_nonlin_fixedbc (d02gac)
nag_ode_bvp_fd_lin_gen (d02gbc)

8. Example

We solve the differential equation

$$y''' = -yy'' - 2\varepsilon(1 - y'^2)$$

with $\varepsilon = 1$ and boundary conditions

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

to an accuracy specified by **tol** = 1.0e−4. The continuation facility is used with the continuation parameter ε introduced as in the differential equation above and with **deleps** = 0.1 initially. (The continuation facility is not needed for this problem and is used here for illustration.)

8.1. Program Text

```
/* nag_ode_bvp_fd_nonlin_gen(d02rac) Example Program
 *
 * Copyright 1994 Numerical Algorithms Group.
 *
 * Mark 3, 1994.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagd02.h>

#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double eps, double y[], double f[],
                Nag_User *comm);
#else
static void fcn();
#endif

#ifdef NAG_PROTO
static void g(Integer neq, double eps, double ya[], double yb[],
```

```
                double bc[], Nag_User *comm);
#else
static void g();
#endif

#ifdef NAG_PROTO
static void jaceps(Integer neq, double x, double eps, double y[], double f[],
                   Nag_User *comm);
#else
static void jaceps();
#endif

#ifdef NAG_PROTO
static void jacgep(Integer neq, double eps, double ya[], double yb[],
                   double bcep[], Nag_User *comm);
#else
static void jacgep();
#endif

#ifdef NAG_PROTO
static void jacobf(Integer neq, double x, double eps, double y[],
                   double f[], Nag_User *comm);
#else
static void jacobf();
#endif

#ifdef NAG_PROTO
static void jacobg(Integer neq, double eps, double ya[], double yb[],
                   double aj[], double bj[], Nag_User *comm);
#else
static void jacobg();
#endif

#define NEQ 3
#define MNP 40

main()
{
  Integer i, j;
  double x[MNP], y[NEQ][MNP];
  Integer np;
  double deleps;
  Integer numbeg, nummix;
  double abt[NEQ];
  double tol;
  Integer neq, mnp;
  Nag_User comm;

  Vprintf("d02rac Example Program Results\n");

  Vprintf ("\nCalculation using analytic Jacobians\n\n");


  neq = NEQ;
  mnp = MNP;
  tol = 1.0e-4;
  np = 17;
  numbeg = 2;
  nummix = 0;
  x[0] = 0.0;
  x[np-1] = 10.0;
  deleps = 0.1;

  d02rac(neq, &deleps, fcn, numbeg, nummix, g, Nag_DefInitMesh, mnp, &np, x,
         (double *)y, tol, abt, jacobf, jacobg, jaceps, jacgep,
         &comm, NAGERR_DEFAULT);


  Vprintf ("Solution on final mesh of %ld points \n", np);
  Vprintf ("      X           Y(1)        Y(2)        Y(3)\n");
```

```
  for (j=0; j< np; ++j)
    {
      Vprintf (" %9.3f ", x[j]);
      for (i=0; i<neq; ++i)
        Vprintf ("   %9.4f", y[i][j]);
      Vprintf ("\n");
    }

  Vprintf ("\n\nMaximum estimated error by components \n");

  for (i=1; i<=3; ++i)
    Vprintf (" %9.2e", abt[i-1]);
  Vprintf (" \n");

  exit(EXIT_SUCCESS);
}


#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double eps, double y[], double f[],
                Nag_User *comm)
#else
     static void fcn(neq, x, eps, y, f, comm)
     Integer neq;
     double x, eps;
     double y[], f[];
     Nag_User *comm;
#endif
{
  f[0] = y[1];
  f[1] = y[2];
  f[2] = -y[0] * y[2] - (1.0 - y[1]*y[1])*2.0*eps;

}


#ifdef NAG_PROTO
static void g(Integer neq, double eps, double ya[], double yb[],
              double bc[], Nag_User *comm)
#else
     static void g(neq, eps, ya, yb, bc, comm)
     Integer neq;
     double eps;
     double ya[], yb[], bc[];
     Nag_User *comm;
#endif
{
  bc[0] = ya[0];
  bc[1] = ya[1];
  bc[2] = yb[1] - 1.0;
}                               /* g */


#ifdef NAG_PROTO
static void jaceps(Integer neq, double x, double eps, double y[],
                   double f[], Nag_User *comm)

#else
     static void jaceps(neq, x, eps, y, f, comm)
     Integer neq;
     double x, eps;
     double y[], f[];
     Nag_User *comm;
#endif
{
  f[0] = 0.0;
  f[1] = 0.0;
  f[2] = (1.0 - y[1]*y[1]) * -2.0;
}
```

```
#ifdef NAG_PROTO
static void jacgep(Integer neq, double eps, double ya[], double yb[],
                   double bcep[], Nag_User *comm)
#else
     static void jacgep(neq, eps, ya, yb, bcep, comm)
     Integer neq;
     double eps;
     double ya[], yb[], bcep[];
     Nag_User *comm;
#endif
{
  Integer i;

  for (i=0; i< neq; ++i)
    bcep[i] = 0.0;

}

#ifdef NAG_PROTO
static void jacobf(Integer neq, double x, double eps, double y[],
                   double f[], Nag_User *comm)
#else
     static void jacobf(neq, x, eps, y, f, comm)
     Integer neq;
     double x, eps;
     double y[], f[];
     Nag_User *comm;
#endif
{

  Integer i, j;

#define Y(I) y[(I)-1]
#define F(I,J) f[((I)-1)*neq+(J)-1]

  for (i=1; i<= neq; ++i)
    {
      for (j=1; j<= neq; ++j)
        F(i, j) = 0.0;
    }
  F(1,2) = 1.0;
  F(2,3) = 1.0;
  F(3,1) = -Y(3);
  F(3,2) = Y(2) * 4.0 * eps;
  F(3,3) = -Y(1);
}


#ifdef NAG_PROTO
static void jacobg(Integer neq, double eps, double ya[], double yb[],
                   double aj[], double bj[], Nag_User *comm)
#else
     static void jacobg(neq, eps, ya, yb, aj, bj, comm)
     Integer neq;
     double eps;
     double ya[], yb[], aj[], bj[];
     Nag_User *comm;
#endif
{
  Integer i, j;
#define YA(I) ya[(I)-1]
#define YB(I) yb[(I)-1]
#define AJ(I,J) aj[((I)-1)*neq+(J)-1]
#define BJ(I,J) bj[((I)-1)*neq+(J)-1]

  for (i=1; i<= neq; ++i)
    for (j=1; j<= neq; ++j)
      {
```

```
        AJ(i,j) = 0.0;
        BJ(i,j) = 0.0;
      }
  AJ(1,1) = 1.0;
  AJ(2,2) = 1.0;
  BJ(3,2) = 1.0;
}
```

8.2. Program Data

None.

8.3. Program Results

```
d02rac Example Program Results

Calculation using analytic Jacobians

Solution on final mesh of 33 points
      X          Y(1)        Y(2)        Y(3)
     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
```