# D02AGF – NAG Fortran Library Routine Document

**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

D02AGF solves the two-point boundary-value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalizes D02HAF to include the case where parameters other than boundary values are to be determined.

## 2 Specification

```
 SUBROUTINE D02AGF(H, ERROR, PARERR, PARAM, C, N, N1, M1, AUX,
1                  BCAUX, RAAUX, PRSOL, MAT, COPY, WSPACE, WSPAC1,
2                  WSPAC2, IFAIL)
 INTEGER           N, N1, M1, IFAIL
 real              H, ERROR(N), PARERR(N1), PARAM(N1), C(M1,N),
1                  MAT(N1,N1), COPY(N1,N1), WSPACE(N,9), WSPAC1(N),
2                  WSPAC2(N)
 EXTERNAL          AUX, BCAUX, RAAUX, PRSOL
```

## 3 Description

The routine solves the two-point boundary-value problem by determining the unknown parameters $p_1, p_2, \ldots, p_{n_1}$ of the problem. These parameters may be, but need not be, boundary values (as they are in D02HAF); they may include eigenvalue parameters in the coefficients of the differential equations, length of the range of integration, etc. The notation and methods used are similar to those of D02HAF and the user is advised to study this first. (There the parameters $p_1, p_2, \ldots, p_{n_1}$ correspond to the unknown boundary conditions.) It is assumed that we have a system of $n$ first-order ordinary differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n), \quad i = 1, 2, \ldots, n,$$

and that derivatives $f_i$ are evaluated by a subroutine AUX supplied by the user. The system, including the boundary conditions given by BCAUX, and the range of integration and matching point, $r$, given by RAAUX, involves the $n_1$ unknown parameters $p_1, p_2, \ldots, p_{n_1}$ which are to be determined, and for which initial estimates must be supplied. The number of unknown parameters $n_1$ must not exceed the number of equations $n$. If $n_1 < n$, we assume that $(n - n_1)$ equations of the system are not involved in the matching process. These are usually referred to as 'driving equations'; they are independent of the parameters and of the solutions of the other $n_1$ equations. In numbering the equations for the subroutine AUX, the driving equations must be put last.

The estimated values of the parameters are corrected by a form of Newton iteration. The Newton correction on each iteration is calculated using a matrix whose $(i, j)$th element depends on the derivative of the $i$th component of the solution, $y_i$, with respect to the $j$th parameter, $p_j$. This matrix is calculated by a simple numerical differentiation technique which requires $n_1$ evaluations of the differential system.

## 4 References

None.

## 5 Parameters

Users are strongly recommended to read Section 3 and Section 8 in conjunction with this section.

**1:** H — ***real*** *Input/Output*

*On entry:* H must be set to an estimate of the step size needed for integration, $h$.

*On exit:* the last step length used.

**2:** ERROR(N) — ***real*** array *Input*

*On entry:* ERROR($i$) must be set to a small quantity to control the $i$th solution component. The element ERROR($i$) is used:

(i) in the bound on the local error in the $i$th component of the solution $y_i$ during integration,
(ii) in the convergence test on the $i$th component of the solution $y_i$ at the matching point in the Newton iteration.

The elements ERROR($i$) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

**3:** PARERR(N1) — ***real*** array *Input*

*On entry:* PARERR($i$) must be set to a small quantity to control the $i$th parameter component. The element PARERR($i$) is used:

(i) in the convergence test on the $i$th parameter in the Newton iteration,
(ii) in perturbing the $i$th parameter when approximating the derivatives of the components of the solution with respect to the $i$th parameter, for use in the Newton iteration.

The elements PARERR($i$) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

**4:** PARAM(N1) — ***real*** array *Input/Output*

*On entry:* PARAM($i$) must be set to an estimate for the $i$th parameter, $p_i$, for $i = 1, 2, \ldots, \text{N1}$.

*On exit:* the corrected value for the $i$th parameter, unless an error has occurred, when it contains the last calculated value of the parameter (possibly perturbed by PARERR($i$) $\times$ (1 + |PARAM($i$)|) if the error occurred when calculating the approximate derivatives).

**5:** C(M1,N) — ***real*** array *Output*

*On exit:* the solution when M1 $> 1$ (see below).

If M1 $= 1$ then the elements of C are not used.

**6:** N — INTEGER *Input*

*On entry:* the total number of differential equations, $n$.

**7:** N1 — INTEGER *Input*

*On entry:* the number of parameters, $n_1$.

If N1 $<$ N, the last N$-$N1 differential equations (in the subroutine AUX below) are driving equations (see Section 3).

*Constraint:* N1 $\leq$ N.

**8:** M1 — INTEGER *Input*

*On entry:* determines whether or not the final solution is computed as well as the parameter values. For

M1 $= 1$

the final solution is not calculated;

M1 $> 1$

the final values of the solution at interval (length of range)/(M1 $-$ 1) are calculated and stored sequentially in the array C starting with the values of $y_i$ evaluated at the first end-point (see subroutine RAAUX below) stored in C($1, i$).

**9:** AUX — SUBROUTINE, supplied by the user. *External Procedure*

AUX must evaluate the functions $f_i$ (i.e., the derivatives $y_i'$) for given values of its arguments, $x, y_1, \ldots, y_n, p_1, \ldots, p_{n_1}$

Its specification is:

```
SUBROUTINE AUX(F, Y, X, PARAM)
real           F(n), Y(n), X, PARAM(n1)
```

where n and n1 are the numerical values of N and N1 in the call of D02AGF.

**1:** F(n) — ***real*** array *Output*
*On exit:* the value of $f_i$, for $i = 1, 2, \ldots, n$.

**2:** Y(n) — ***real*** array *Input*
*On entry:* the value of the argument $y_i$, for $i = 1, 2, \ldots, n$.

**3:** X — ***real*** *Input*
*On entry:* the value of the argument $x$.

**4:** PARAM(n1) — ***real*** array *Input*
*On entry:* the value of the argument $p_i$, for $i = 1, 2, \ldots, n_1$.

AUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

**10:** BCAUX — SUBROUTINE, supplied by the user. *External Procedure*

BCAUX must evaluate the values of $y_i$ at the end-points of the range given the values of $p_1, \ldots, p_{n_1}$.

Its specification is:

```
SUBROUTINE BCAUX(G0, G1, PARAM)
real             G0(n), G1(n), PARAM(n1)
```

where n and n1 are the numerical values of N and N1 in the call of D02AGF.

**1:** G0(n) — ***real*** array *Output*
*On exit:* the values $y_i$, for $i = 1, 2, \ldots, n$, at the boundary point $x_0$ (see RAAUX below).

**2:** G1(n) — ***real*** array *Output*
*On exit:* the values $y_i$, for $i = 1, 2, \ldots, n$, at the boundary point $x_1$ (see RAAUX below).

**3:** PARAM(n1) — ***real*** array *Input*
*On entry:* the value of the argument $p_i$, for $i = 1, 2, \ldots, n$.

BCAUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

**11:** RAAUX — SUBROUTINE, supplied by the user. *External Procedure*

RAAUX must evaluate the end-points, $x_0$ and $x_1$, of the range and the matching point, $r$, given the values $p_1, p_2, \ldots, p_{n_1}$.

Its specification is:

```
SUBROUTINE RAAUX(X0, X1, R, PARAM)
real             X0, X1, R, PARAM(n1)
```

where n1 is the numerical value of N1 in the call of D02AGF.

**1:** X0 — ***real*** *Output*
*On exit:* must contain the left-hand end of the range, $x_0$.

**2:** X1 — ***real*** *Output*
*On exit:* must contain the right-hand end of the range $x_1$.

**3:** R — ***real*** *Output*
*On exit:* must contain the matching point, $r$.

**4:** PARAM(n1) — ***real*** array *Input*
*On entry:* the value of the argument $p_i$, for $i = 1, 2, \ldots, n_1$.

RAAUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

**12:** PRSOL — SUBROUTINE, supplied by the user. *External Procedure*

PRSOL is called at each iteration of the Newton method and can be used to print the current values of the parameters $p_i$, for $i = 1, 2, \ldots, n_1$, their errors, $e_i$, and the sum of squares of the errors at the matching point, $r$.

Its specification is:

```
SUBROUTINE PRSOL(PARAM, RES, N1, ERR)
INTEGER          N1
real             PARAM(N1), RES, ERR(N1)
```

**1:** PARAM(N1) — ***real*** array *Input*
*On entry:* the current value of the parameters $p_i$, for $i = 1, 2, \ldots, n_1$.

**2:** RES — ***real*** *Input*
*On entry:* the sum of squares of the errors in the parameters, $\sum_{i=1}^{n_1} e_i^2$.

**3:** N1 — INTEGER *Input*
*On entry:* the number of parameters, $n_1$.

**4:** ERR(N1) — ***real*** array *Input*
*On entry:* the errors in the parameters, $e_i$, for $i = 1, 2, \ldots, n_1$.

PRSOL must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

**13:** MAT(N1,N1) — ***real*** array *Workspace*

**14:** COPY(N1,N1) — ***real*** array *Workspace*

**15:** WSPACE(N,9) — ***real*** array *Workspace*

**16:** WSPAC1(N) — ***real*** array *Workspace*

**17:** WSPAC2(N) — ***real*** array *Workspace*

**18:** IFAIL — INTEGER *Input/Output*

*On entry:* IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

# 6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

This indicates that N1 > N on entry, that is the number of parameters is greater than the number of differential equations.

IFAIL = 2

As for IFAIL = 4 (below) except that the integration failed while calculating the matrix for use in the Newton iteration.

IFAIL = 3

The current matching point $r$ does not lie between the current end-points $x_0$ and $x_1$. If the values $x_0$, $x_1$ and $r$ depend on the parameters $p_i$, this may occur at any time in the Newton iteration if care is not taken to avoid it when coding subroutine RAAUX.

IFAIL = 4

The step length for integration H has halved more than 13 times (or too many steps were needed to reach the end of the range of integration) in attempting to control the local truncation error whilst integrating to obtain the solution corresponding to the current values $p_i$. If, on failure, H has the sign of $r - x_0$ then failure has occurred whilst integrating from $x_0$ to $r$, otherwise it has occurred whilst integrating from $x_1$ to $r$.

IFAIL = 5

The matrix of the equations to be solved for corrections to the variable parameters in the Newton method is singular (as determined by F03AFF).

IFAIL = 6

A satisfactory correction to the parameters was not obtained on the last Newton iteration employed. A Newton iteration is deemed to be unsatisfactory if the sum of the squares of the residuals (which can be printed using PRSOL) has not been reduced after three iterations using a new Newton correction.

IFAIL = 7

Convergence has not been obtained after 12 satisfactory iterations of the Newton method.

A further discussion of these errors and the steps which might be taken to correct them is given in Section 8.

# 7 Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user; and the solution, if requested, is usually determined to the accuracy specified.

# 8 Further Comments

The time taken by the routine depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

There may be particular difficulty in integrating the differential equations in one direction (indicated by IFAIL = 2 or 4). The value of $r$ should be adjusted to avoid such difficulties.

If the matching point $r$ is at one of the end-points $x_0$ or $x_1$ and some of the parameters are used **only** to determine the boundary values at this point, then good initial estimates for these parameters are not required, since they are completely determined by the routine (for example, see $p_2$ in example (i) of Section 9).

Wherever they occur in the procedure, the error parameters contained in the arrays ERROR and PARERR are used in 'mixed' form; that is ERROR($i$) always occurs in expressions of the form ERROR($i$) $\times (1 + |y_i|)$, and PARERR($i$) always occurs in expressions of the form PARERR($i$) $\times (1 + |p_i|)$. Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

Note that **convergence is not guaranteed**. The user is strongly advised to provide an output subroutine PRSOL, as shown in the example (i) of Section 9, in order to monitor the progress of the iteration. Failure of the Newton iteration to converge (IFAIL = 6 or IFAIL = 7) usually results from poor starting approximations to the parameters, though occasionally such failures occur because the elements of one or both of the arrays PARERR or ERROR are too small. (It should be possible to distinguish these cases by studying the output from PRSOL.) Poor starting approximations can also result in the failure described under IFAIL = 4 and IFAIL = 5 in Section 6 (especially if these errors occur after some Newton iterations have been completed, that is, after two or more calls of PRSOL). More frequently, a singular matrix in the Newton method (monitored as IFAIL = 5) occurs because the mathematical problem has been posed incorrectly. The case IFAIL = 4 usually occurs because $h$ or $r$ has been poorly estimated, so these values should be checked first. If IFAIL = 2 is monitored, the solution $y_1, y_2, \ldots, y_n$ is sensitive to perturbations in the parameters $p_i$. Reduce the size of one or more values PARERR($i$) to reduce the perturbations. Since only one value $p_i$ is perturbed at any time when forming the matrix, the perturbation which is too large can be located by studying the final output from PRSOL and the values of the parameters returned by D02AGF. If this change leads to other types of failure improve the initial values of $p_i$ by other means.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters $p_i$. If it seems that too much computing time is required and, in particular, if the values ERR($i$) (available on each call of PRSOL) are much larger than the expected values of the solution at the matching point $r$, then the coding of the subroutines AUX, BCAUX and RAAUX should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for PARAM($i$).

The subroutine can be used to solve a very wide range of problems, for example:

(a) eigenvalue problems, including problems where the eigenvalue occurs in the boundary conditions;
(b) problems where the differential equations depend on some parameters which are to be determined so as to satisfy certain boundary conditions (see example (ii) in Section 9);
(c) problems where one of the end-points of the range of integration is to be determined as the point where a variable $y_i$ takes a particular value (see (ii) in Section 9);
(d) singular problems and problems on infinite ranges of integration where the values of the solution at $x_0$ or $x_1$ or both are determined by a power series or an asymptotic expansion (or a more complicated expression) and where some of the coefficients in the expression are to be determined (see example (i) in Section 9); and
(e) differential equations with certain terms defined by other independent (driving) differential equations.

# 9 Example

For this routine two examples are presented, in Section 9.1 and Section 9.2. In the example programs distributed to sites, there is a single example program for D02AGF, with a main program:

```
*     D02AGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02AGF Example Program Results'
      CALL EX1
      CALL EX2
      STOP
      END
```

The code to solve the two example problems is given in the subroutines EX1 and EX2, in Section 9.1.1 and Section 9.2.1 respectively.

## 9.1 Example 1

To find the solution of the differential equation

$$y'' = \frac{y^3 - y'}{2x}$$

on the range $0 \leq x \leq 16$, with boundary conditions $y(0) = 0.1$ and $y(16) = 1/6$.

We cannot use the differential equation at $x = 0$ because it is singular, so we take the truncated series expansion

$$y(x) = \frac{1}{10} + p_1 \frac{\sqrt{x}}{10} + \frac{x}{100}$$

near the origin (which is correct to the number of terms given in this case). Where $p_1$ is one of the parameters to be determined. We choose the range as `[0.1,16]` and setting $p_2 = y'(16)$, we can determine all the boundary conditions. We take the matching point to be 16, the end of the range, and so a good initial guess for $p_2$ is not necessary. We write $y = \mathrm{Y}(1)$, $y' = \mathrm{Y}(2)$, and estimate $p_1 = \mathrm{PARAM}(1) = 0.2$, $p_2 = \mathrm{PARAM}(2) = 0.0$.

### 9.1.1 Program Text

**Note.** The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          N, M1
      PARAMETER        (N=2,M1=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          IPRINT
*     .. Local Scalars ..
      real             DUM, H, R, X, X1
      INTEGER          I, IFAIL, J, N1
*     .. Local Arrays ..
      real             C(M1,N), COPY(N,N), ERROR(N), G(N), G1(N),
```

```
     +                MAT(N,N), PARAM(N), PARERR(N), WSPACE(N,9)
*     .. External Subroutines ..
      EXTERNAL         AUX1, BCAUX1, D02AGF, PRSOL, RNAUX1
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           /BLOCK1/IPRINT
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1'
      WRITE (NOUT,*)
*     * Set IPRINT to 1 to obtain output from PRSOL at each iteration *
      IPRINT = 0
      PARAM(1) = 0.2e0
      PARAM(2) = 0.0e0
      N1 = 2
      H = 0.1e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-3
      ERROR(1) = 1.0e-4
      ERROR(2) = 1.0e-4
      IFAIL = 1
*
      CALL D02AGF(H,ERROR,PARERR,PARAM,C,N,N1,M1,AUX1,BCAUX1,RNAUX1,
     +            PRSOL,MAT,COPY,WSPACE,G,G1,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N1)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX1(X,X1,R,PARAM)
         H = (X1-X)/5.0e0
         DO 20 I = 1, 6
            DUM = X + real(I-1)*H
            WRITE (NOUT,99997) DUM, (C(I,J),J=1,N)
   20    CONTINUE
      ELSE
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3e16.6)
99997 FORMAT (1X,F7.2,3e13.4)
      END
*
      SUBROUTINE AUX1(F,Y,X,PARAM)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             F(2), PARAM(2), Y(2)
*     .. Executable Statements ..
      F(1) = Y(2)
      F(2) = (Y(1)**3-Y(2))/(2.0e0*X)
      RETURN
```

```
      END
*
      SUBROUTINE RNAUX1(X,X1,R,PARAM)
*     .. Scalar Arguments ..
      real              R, X, X1
*     .. Array Arguments ..
      real              PARAM(2)
*     .. Executable Statements ..
      X = 0.1e0
      X1 = 16.0e0
      R = 16.0e0
      RETURN
      END
*
      SUBROUTINE BCAUX1(G,G1,PARAM)
*     .. Array Arguments ..
      real              G(2), G1(2), PARAM(2)
*     .. Local Scalars ..
      real              Z
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      Z = 0.1e0
      G(1) = 0.1e0 + PARAM(1)*SQRT(Z)*0.1e0 + 0.01e0*Z
      G(2) = PARAM(1)*0.05e0/SQRT(Z) + 0.01e0
      G1(1) = 1.0e0/6.0e0
      G1(2) = PARAM(2)
      RETURN
      END
*
```

### 9.1.2 Program Data

None.

### 9.1.3 Program Results

```
D02AGF Example Program Results


Case 1

Final parameters
    0.464269E-01    0.349429E-02

Final solution
X-value     Components of solution
   0.10   0.1025E+00   0.1734E-01
   3.28   0.1217E+00   0.4180E-02
   6.46   0.1338E+00   0.3576E-02
   9.64   0.1449E+00   0.3418E-02
  12.82   0.1557E+00   0.3414E-02
  16.00   0.1667E+00   0.3494E-02
```

## 9.2 Example 2

To find the gravitational constant $p_1$ and the range $p_2$ over which a projectile must be fired to hit the target with a given velocity. The differential equations are

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-(p_1 \sin\phi + 0.00002v^2)}{v\cos\phi} \\ \phi' &= \frac{-p_1}{v^2} \end{aligned}$$

on the range $0 < x < p_2$ with boundary conditions

$$\begin{aligned} y = 0, \quad v = 500, \quad \phi = 0.5 \quad \text{at} \quad x = 0 \\ y = 0, \quad v = 450, \quad \phi = p_3 \quad \text{at} \quad x = p_2. \end{aligned}$$

We write $y$ = Y(1), $v$ = Y(2), $\phi$ = Y(3), and we take the matching point $r = p_2$. We estimate $p_1$ = PARAM(1) = 32, $p_2$ = PARAM(2) = 6000 and $p_2$ = PARAM(3) = 0.54 (though this estimate is not important).

### 9.2.1 Program Text

**Note.** The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          N, M1
      PARAMETER        (N=3,M1=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          IPRINT
*     .. Local Scalars ..
      real             DUM, H, R, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(M1,N), COPY(N,N), ERROR(N), G(N), G1(N),
     +                 MAT(N,N), PARAM(N), PARERR(N), WSPACE(N,9)
*     .. External Subroutines ..
      EXTERNAL         AUX2, BCAUX2, D02AGF, PRSOL, RNAUX2
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           /BLOCK1/IPRINT
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2'
      WRITE (NOUT,*)
*     * Set IPRINT to 1 to obtain output from PRSOL at each iteration *
      IPRINT = 0
      H = 10.0e0
      PARAM(1) = 32.0e0
      PARAM(2) = 6000.0e0
      PARAM(3) = 0.54e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-4
```

```
      PARERR(3) = 1.0e-4
      ERROR(1) = 1.0e-2
      ERROR(2) = 1.0e-2
      ERROR(3) = 1.0e-2
      IFAIL = 1
*
      CALL D02AGF(H,ERROR,PARERR,PARAM,C,N,N,M1,AUX2,BCAUX2,RNAUX2,
     +            PRSOL,MAT,COPY,WSPACE,G,G1,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX2(X,X1,R,PARAM)
         H = (X1-X)/5.0e0
         DO 20 I = 1, 6
            DUM = X + real(I-1)*H
            WRITE (NOUT,99997) DUM, (C(I,J),J=1,N)
   20    CONTINUE
      ELSE
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3e16.6)
99997 FORMAT (1X,F7.0,3e13.4)
      END
*
      SUBROUTINE AUX2(F,Y,X,PARAM)
*     .. Scalar Arguments ..
      real              X
*     .. Array Arguments ..
      real              F(3), PARAM(3), Y(3)
*     .. Local Scalars ..
      real              C, S
*     .. Intrinsic Functions ..
      INTRINSIC         COS, SIN
*     .. Executable Statements ..
      C = COS(Y(3))
      S = SIN(Y(3))
      F(1) = S/C
      F(2) = -(PARAM(1)*S+0.00002e0*Y(2)*Y(2))/(Y(2)*C)
      F(3) = -PARAM(1)/(Y(2)*Y(2))
      RETURN
      END
*
      SUBROUTINE RNAUX2(X,X1,R,PARAM)
*     .. Scalar Arguments ..
      real                R, X, X1
*     .. Array Arguments ..
      real                PARAM(3)
*     .. Executable Statements ..
      X = 0.0e0
      X1 = PARAM(2)
      R = PARAM(2)
```

```
      RETURN
      END
*
      SUBROUTINE BCAUX2(G,G1,PARAM)
*     .. Array Arguments ..
      real              G(3), G1(3), PARAM(3)
*     .. Executable Statements ..
      G(1) = 0.0e0
      G(2) = 500.0e0
      G(3) = 0.5e0
      G1(1) = 0.0e0
      G1(2) = 450.0e0
      G1(3) = PARAM(3)
      RETURN
      END
*
      SUBROUTINE PRSOL(PARAM,RESID,N1,ERR)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             RESID
      INTEGER          N1
*     .. Array Arguments ..
      real             ERR(N1), PARAM(N1)
*     .. Scalars in Common ..
      INTEGER          IPRINT
*     .. Local Scalars ..
      INTEGER          I
*     .. Common blocks ..
      COMMON           /BLOCK1/IPRINT
*     .. Executable Statements ..
      IF (IPRINT.NE.0) THEN
         WRITE (NOUT,99999) 'Current parameters   ', (PARAM(I),I=1,N1)
         WRITE (NOUT,99998) 'Residuals   ', (ERR(I),I=1,N1)
         WRITE (NOUT,99998) 'Sum of residuals squared  ', RESID
         WRITE (NOUT,*)
      END IF
      RETURN
*
99999 FORMAT (1X,A,6(e14.6,2X))
99998 FORMAT (1X,A,6(e12.4,1X))
      END
```

### 9.2.2 Program Data

None.

### 9.2.3 Program Results

```
Case 2

Final parameters
    0.323729E+02    0.596317E+04   -0.535231E+00

Final solution
X-value     Components of solution
     0.   0.0000E+00   0.5000E+03   0.5000E+00
  1193.   0.5298E+03   0.4516E+03   0.3281E+00
  2385.   0.8076E+03   0.4203E+03   0.1231E+00
  3578.   0.8208E+03   0.4094E+03  -0.1032E+00
  4771.   0.5563E+03   0.4200E+03  -0.3296E+00
  5963.   0.0000E+00   0.4500E+03  -0.5352E+00
```