NAG Fortran Library Routine Document

D02HBF

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

D02HBF solves the two-point boundary-value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalises subroutine D02HAF to include the case where parameters other than boundary values are to be determined.

2 Specification

```
      SUBROUTINE D02HBF(P, N1, PE, E, N, SOLN, M1, FCN, BC, RANGE, W, IW,
     1                  IFAIL)
      INTEGER           N1, N, M1, IW, IFAIL
      real              P(N1), PE(N1), E(N), SOLN(N,M1), W(N,IW)
      EXTERNAL          FCN, BC, RANGE
```

3 Description

D02HBF 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; 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. (The parameters $p_1, p_2, \ldots, p_{n_1}$ correspond precisely to the unknown boundary conditions in D02HAF.) 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 the derivatives f_i are evaluated by a subroutine FCN supplied by the user. The system, including the boundary conditions given by BC and the range of integration given by RANGE, 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 FCN, the driving equations must be put **first**.

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

If the parameter IFAIL is set appropriately, the routine automatically prints messages to inform the user of the flow of the calculation. These messages are discussed in detail in Section 8.

D02HBF is a simplified version of D02SAF which is described in detail in Gladwell (1979a).

4 References

Gladwell I (1979a) The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *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

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

1: P(N1) – ***real*** array *Input/Output*

On entry: an estimate for the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

On exit: the corrected value for the ith parameter, unless an error has occurred, when it contains the last calculated value of the parameter.

2: N1 – INTEGER *Input*

On entry: the number of parameters, n_1.

Constraint: $1 \leq$ N1 $\leq$ N.

3: PE(N1) – ***real*** array *Input*

On entry: the elements of PE must be given small positive values. The element PE(i) is used

(i) in the convergence test on the ith parameter in the Newton iteration, and

(ii) in perturbing the ith parameter when approximating the derivatives of the components of the solution with respect to this parameter for use in the Newton iteration.

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

Constraint: PE$(i) > 0.0$, for $i = 1, 2, \ldots,$ N1.

4: E(N) – ***real*** array *Input*

On entry: the elements of E must be given positive values.The element E(i) is used in the bound on the local error in the ith component of the solution y_i during integration.

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

Constraint: E$(i) > 0.0$, for $i = 1, 2, \ldots,$ N.

5: N – INTEGER *Input*

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

Constraint: N ≥ 2.

6: SOLN(N,M1) – ***real*** array *Output*

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

7: M1 – INTEGER *Input*

On entry: a value which controls exit values.

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 SOLN starting with the values of the solutions evaluated at the first end-point (see subroutine RANGE below) stored in the first column of SOLN.

Constraint: M1 ≥ 1.

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

FCN must evaluate the function f_i (i.e., the derivative y_i'), for $i = 1, 2, \ldots, n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F, P)
real              X, Y(n), F(n), P(n1)
```

where n and n1 are the actual values of N and N1 in the call of D02HBF.

1: X – ***real*** *Input*

On entry: the value of the argument x.

2: Y(n) – ***real*** array *Input*

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

3: F(n) – ***real*** array *Output*

On exit: the value of f_i, for $i = 1, 2, \ldots, n$. The f_i may depend upon the parameters p_j, for $j = 1, 2, \ldots, n_1$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of F in FCN.

4: P(n1) – ***real*** array *Input*

On entry: the current estimate of the parameter p_i, for $i = 1, 2, \ldots, n_1$.

FCN must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: BC – SUBROUTINE, supplied by the user. *External Procedure*

BC must place in G1 and G2 the boundary conditions at a and b respectively (see RANGE below).

Its specification is:

```
SUBROUTINE BC(G1, G2, P)
real             G1(n), G2(n), P(n1)
```

where n and n1 are the actual values of N and N1 in the call of D02HBF.

1: G1(n) – ***real*** array *Output*

On exit: the value of $y_i(a)$, (where this may be a known value or a function of the parameters p_j, for $j = 1, 2, \ldots, n_1$); $i = 1, 2, \ldots, n$.

2: G2(n) – ***real*** array *Output*

On exit: the value of $y_i(b)$, for $i = 1, 2, \ldots, n$, (where these may be known values or functions of the parameters p_j, for $j = 1, 2, \ldots, n_1$). If $n > n_1$, so that there are some driving equations, then the first $n - n_1$ values of G2 need not be set since they are never used.

3: P(n1) – ***real*** array *Input*

On entry: an estimate of the parameter p_i, for $i = 1, 2, \ldots, n_1$.

BC must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: RANGE – SUBROUTINE, supplied by the user. *External Procedure*

RANGE must evaluate the boundary points a and b, each of which may depend on the parameters $p_1, p_2, \ldots, p_{n_1}$. The integrations in the shooting method are always from a to b.

Its specification is:

```
SUBROUTINE RANGE(A, B, P)
real              A, B, P(n1)
```

where n1 is the actual value of N1 in the call of D02HBF.

1: A – ***real*** *Output*

On exit: one of the boundary points, a.

2: B – ***real*** *Output*

On exit: the second boundary point, b. Note that B > A forces the direction of integration to be that of increasing X. If A and B are interchanged the direction of integration is reversed.

3: P(n1) – ***real*** array *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

RANGE must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: W(N,IW) – ***real*** array *Output*

Used mainly for workspace.

On exit: with IFAIL = 2, 3, 4 or 5 (see Section 6), W$(i, 1)$, for $i = 1, 2, \ldots, n$ contains the solution at the point x when the error occurred. W(1,2) contains x.

12: IW – INTEGER *Input*

On entry: the second dimension of the array W as declared in the (sub)program from which D02HBF is called.

Constraint: $\mathrm{IW} \geq 3\mathrm{N} + 14 + \max(11, \mathrm{N})$.

13: 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 Chapter P01).

Before 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).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

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, N1, M1, IW, E or PE is incorrectly set.

IFAIL = 2

The step length for the integration became too short whilst calculating the residual (see Section 8).

IFAIL = 3

No initial step length could be chosen for the integration whilst calculating the residual.

Note: IFAIL = 2 or 3 can occur due to choosing too small a value for E or due to choosing the wrong direction of integration. Try varying E and interchanging a and b. These error exits can also occur for very poor initial choices of the parameters in the array P and, in extreme cases, because this routine cannot be used to solve the problem posed.

IFAIL = 4

As for IFAIL = 2 but the error occurred when calculating the Jacobian.

IFAIL = 5

As for IFAIL = 3 but the error occurred when calculating the Jacobian.

IFAIL = 6

The calculated Jacobian has an insignificant column. This can occur because a parameter p_i is incorrectly entered when posing the problem.

Note: IFAIL = 4, 5 or 6 usually indicate a badly scaled problem. The user may vary the size of PE. Otherwise the use of the more general D02SAF which affords more control over the calculations is advised.

IFAIL = 7

The linear algebra routine used (F02WEF) has failed. This error exit should not occur and can be avoided by changing the initial estimates p_i.

IFAIL = 8

The Newton iteration has failed to converge. This can indicate a poor initial choice of parameters p_i or a very difficult problem. Consider varying the elements PE(i) if the residuals are small in the monitoring output. If the residuals are large, try varying the initial parameters p_i.

IFAIL = 9

Indicate that a serious error has occurred in D02SAZ, D02SAW, D02SAX, D02SAU or D02SAV respectively. Check all array subscripts and subroutine parameter lists in the call to D02HBF. Seek expert help.

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, may be determined to a required accuracy by varying the parameter E.

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.

Wherever they occur in the routine, the error parameters contained in the arrays E and PE are used in 'mixed' form; that is E(i) always occurs in expressions of the form

$$\mathrm{E}(i) \times (1 + |y_i|)$$

and PE(i) always occurs in expressions of the form

$$\mathrm{PE}(i) \times (1 + |p_i|).$$

Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

The user may determine a suitable direction of integration a to b and suitable values for E(i) by integrations with D02PCF. The best direction of integration is usually the direction of decreasing solutions. The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. The user may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the implementation document. The monitoring information produced at each iteration includes the current parameter values, the residuals and two norms: a basic norm and a current norm. At each iteration the aim is to find parameter values which make the current norm less than the basic norm. Both these norms should tend to zero as should the residuals. (They would all be zero if the exact parameters were used as input.) For more details, in particular about the other monitoring information printed, the user is advised to consult the specification of D02SAF and, especially, the description of the parameter MONIT there.

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 of the residuals printed by the monitoring routine are much larger than the expected values of the solution at b then the coding of the subroutines FCN, BC and RANGE should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for p_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 example (ii) in Section 9);

(d) singular problems and problems on infinite ranges of integration where the values of the solution at a or b 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 of the documents for D02HBF and D02HBF. In the example programs distributed to sites, there is a single example program for D02HBF, with a main program:

```
*     D02HBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02HBF 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 D02HBF and D02HBF respectively.

9.1 Example 1

To find the solution of the differential equation

$$y'' = (y^3 - y')/2x$$

on the range $0 \le x \le 16$, 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 a truncated power series expansion

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

near the origin where p_1 is one of the parameters to be determined. We choose the interval as $[0.1, 16]$ and setting $p_2 = y'(16)$, we can determine all the boundary conditions. We take X1 = 16. We write $y = \mathrm{Y}(1)$, $y' = \mathrm{Y}(2)$, and estimate PARAM(1) = 0.2, PARAM(2) = 0.0. Note the call to X04ABF before the call to D02HBF.

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          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, N1, IW, M1
      PARAMETER        (N=2,N1=2,IW=3*N+14+11,M1=6)
*     .. Local Scalars ..
      real             H, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(N,M1), ERROR(N), PARAM(N1), PARERR(N1), W(N,IW)
*     .. External Subroutines ..
      EXTERNAL         AUX1, BCAUX1, D02HBF, RNAUX1, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1'
      CALL X04ABF(1,NOUT)
      PARAM(1) = 0.2e0
      PARAM(2) = 0.0e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-3
      ERROR(1) = 1.0e-4
      ERROR(2) = 1.0e-4
*     * Set IFAIL to 111 to obtain monitoring information *
      IFAIL = 11
*
      CALL D02HBF(PARAM,N1,PARERR,ERROR,N,C,M1,AUX1,BCAUX1,RNAUX1,W,IW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.LE.5 .AND. IFAIL.NE.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99996) 'W(1,2) = ', W(1,2), ' W(.,1) = ',
     +        (W(I,1),I=1,N)
         END IF
      ELSE
         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,PARAM)
         H = (X1-X)/real(M1-1)
```

```
         DO 20 I = 1, M1
            WRITE (NOUT,99997) X + (I-1)*H, (C(J,I),J=1,N)
   20    CONTINUE
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,1P,3e15.3)
99997 FORMAT (1X,F7.2,2F13.4)
99996 FORMAT (1X,A,F9.4,A,10e10.3)
      END
*
      SUBROUTINE AUX1(X,Y,F,PARAM)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=2)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             F(N), PARAM(N), Y(N)
*     .. Executable Statements ..
      F(1) = Y(2)
      F(2) = (Y(1)**3-Y(2))/(2.0e0*X)
      RETURN
      END
*
      SUBROUTINE RNAUX1(X,X1,PARAM)
*     .. Scalar Arguments ..
      real               X, X1
*     .. Array Arguments ..
      real               PARAM(2)
*     .. Executable Statements ..
      X = 0.1e0
      X1 = 16.0e0
      RETURN
      END
*
      SUBROUTINE BCAUX1(G,G1,PARAM)
*     .. Parameters ..
      INTEGER            N
      PARAMETER          (N=2)
*     .. Array Arguments ..
      real               G(N), G1(N), PARAM(N)
*     .. 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

```
 D02HBF Example Program Results


 Case 1

 Final parameters
       4.629E-02      3.494E-03

 Final solution
 X-value     Components of solution
    0.10       0.1025       0.0173
    3.28       0.1217       0.0042
    6.46       0.1338       0.0036
    9.64       0.1449       0.0034
   12.82       0.1557       0.0034
   16.00       0.1667       0.0035
```

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

$$y' = \tan\phi$$

$$v' = \frac{-(p_1 \sin\phi + 0.00002v^2)}{v\cos\phi}$$

$$\phi' = \frac{-p_1}{v^2}$$

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

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

We write $y = \mathrm{Y}(1)$, $v = \mathrm{Y}(2)$, $\phi = \mathrm{Y}(3)$. We estimate $p_1 = \mathrm{PARAM}(1) = 32$, $p_2 = \mathrm{PARAM}(2) = 6000$ and $p_3 = \mathrm{PARAM}(3) = 0.54$ (though this last 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          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, N1, IW, M1
      PARAMETER        (N=3,N1=3,IW=3*N+14+11,M1=6)
*     .. Local Scalars ..
      real             H, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(N,M1), ERROR(N), PARAM(N1), PARERR(N1), W(N,IW)
*     .. External Subroutines ..
      EXTERNAL         AUX2, BCAUX2, D02HBF, RNAUX2, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2'
      CALL X04ABF(1,NOUT)
      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
*     * Set IFAIL to 111 to obtain monitoring information *
      IFAIL = 11
*
      CALL D02HBF(PARAM,N1,PARERR,ERROR,N,C,M1,AUX2,BCAUX2,RNAUX2,W,IW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.LE.5 .AND. IFAIL.NE.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99996) 'W(1,2) = ', W(1,2), ' W(.,1) = ',
     +        (W(I,1),I=1,N)
         END IF
      ELSE
         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 RNAUX2(X,X1,PARAM)
         H = (X1-X)/real(M1-1)
         DO 20 I = 1, M1
            WRITE (NOUT,99997) X + (I-1)*H, (C(J,I),J=1,N)
   20    CONTINUE
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,1P,3e15.3)
99997 FORMAT (1X,F7.0,2F13.1,F13.3)
99996 FORMAT (1X,A,F9.4,A,10e10.3)
      END
*
      SUBROUTINE AUX2(X,Y,F,PARAM)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=3)
*     .. Scalar Arguments ..
      real            X
*     .. Array Arguments ..
      real            F(N), PARAM(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC       COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -PARAM(1)*TAN(Y(3))/Y(2) - 0.00002e0*Y(2)/COS(Y(3))
      F(3) = -PARAM(1)/Y(2)**2
      RETURN
      END
*
      SUBROUTINE RNAUX2(X,X1,PARAM)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Scalar Arguments ..
      real              X, X1
*     .. Array Arguments ..
      real              PARAM(N)
*     .. Executable Statements ..
      X = 0.0e0
      X1 = PARAM(2)
      RETURN
      END
```

```
*
      SUBROUTINE BCAUX2(G,G1,PARAM)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Array Arguments ..
      real              G(N), G1(N), PARAM(N)
*     .. 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
```

9.2.2 Program Data

None.

9.2.3 Program Results

```
 D02HBF Example Program Results


 Case 2

 Final parameters
       3.239E+01      5.962E+03     -5.353E-01

 Final solution
 X-value     Components of solution
      0.           0.0        500.0        0.500
   1192.         529.6        451.6        0.328
   2385.         807.2        420.3        0.123
   3577.         820.4        409.4       -0.103
   4769.         556.1        420.0       -0.330
   5962.          -0.0        450.0       -0.535
```