AD-A040 925 ROYAL AIRCRAFT ESTABLISHMENT FARNBOROUGH (ENGLAND) F/G 9/2
A FORTRAN SUBROUTINE TO SOLVE THE TWO-POINT BOUNDARY VALUE PROB--ETC(U)
JAN 77 L J RICHARDS

UNCLASSIFIED RAE-TM-MATH-7701 DRIC-BR-56405 NL

1 OF 1
AD A040925

END

DATE FILMED
7-77

TECH. MEMO
MATH 7701

TECH. MEMO
MATH 7701

ROYAL AIRCRAFT ESTABLISHMENT

A FORTRAN SUBROUTINE TO SOLVE THE TWO-POINT BOUNDARY VALUE PROBLEM

by

L. J. Richards

January 1977

ROYAL AIRCRAFT ESTABLISHMENT

(7) Technical Memorandum Math 7701

Received for printing 10 January 1977

(6) A FORTRAN SUBROUTINE TO SOLVE THE TWO-POINT BOUNDARY VALUE PROBLEM

(11) Jan 77

by

(10) L. J. Richards

(12) 22 p

SUMMARY

This Memorandum describes a Fortran subroutine, BOVA, which provides the solution of a set of nonlinear ordinary differential equations subject to non-mixed two-point boundary value conditions. A finite difference method is used, together with Newton's method. A specification and listing of the subroutine is given.

LIST OF CONTENTS

1 INTRODUCTION

We consider the following problem. Given a set of n nonlinear first order ordinary differential equations

$$\left.\begin{aligned} \dot{x}_1 &= f_1(x_1, x_2, \ldots, x_n) \\ \dot{x}_2 &= f_2(x_1, x_2, \ldots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, x_2, \ldots, x_n) \end{aligned}\right\} \qquad (1\text{-}1)$$

find a solution trajectory $x_1(t), x_2(t), \ldots, x_n(t)$ over the interval $[T_0, T_F]$ which satisfies n two-point boundary conditions each of the form

$$x_i(T_0) = d_i$$

or

$$x_i(T_F) = d_i$$

for some $i (1 \leq i \leq n)$. The differential equations are autonomous, ie the independent variable t does not appear in the right hand sides of (1-1). A non-autonomous system can easily be put into autonomous form by defining a new variable $x_{n+1}(t) = t$, satisfying $\dot{x}_{n+1} = 0$, with $x_{n+1}(T_0) = T_0$. Similarly, equations involving higher order derivatives can be replaced by a larger set of first order equations by defining new variables. We must assume however that each of the boundary conditions holds at only one point (either T_0 or T_F). If this assumption were not satisfied a method similar to that described in section 2 could be used, but this would involve a large increase in the amount of computation and storage required.

The Fortran subroutine described in this Memorandum was designed for use in a numerical application of optimal control theory. When an optimal control problem is solved using Pontryagin's Maximum Principle, this leads to a two-point boundary value problem. It was found that this boundary value problem could not be solved by conventional 'shooting' methods, so that an alternative method was required. The method described here is applicable to any two-point boundary value problem, but it would not be a good choice for a general purpose two-point boundary value algorithm, since in most cases more effective algorithms can be found.

2 METHOD

Equations (1-1) and (1-2) may be rewritten in a more compact notation. The differential equations are

$$\dot{\underline{x}} = \underline{f}(\underline{x}) \tag{2-1}$$

and the boundary conditions are

$$\left.\begin{aligned} C^0\underline{x}(T_0) &= \underline{d}^0 \\ C^F\underline{x}(T_F) &= \underline{d}^F \end{aligned}\right\} \tag{2-2}$$

where $\underline{x}$ is an n-vector, $\underline{d}^0$ is an ℓ-vector, d^F is an $(n - \ell)$-vector, and C^0 and C^F are $\ell \times n$ and $(n - \ell) \times n$ matrices with constant elements. We use a finite difference formula to convert the set of nonlinear differential equations (2-1) into a set of nonlinear algebraic equations, which are then solved using Newton's method. Instead of trying to determine the solution trajectory throughout the interval we consider its value only at regularly spaced 'grid points'

$$T_r = T_0 + rh \qquad (r = 0,\ldots,N)$$

where $h = (T_F - T_0)/N$ and N is some positive integer. The method will yield

$$\underline{x}(T_0),\ \underline{x}(T_1),\ \ldots,\ x(T_N) = x(T_F) \quad .$$

The finite difference approximation is similar to one suggested by Fox[1]

$$y_1 - y_0 = \tfrac{1}{2}h(\dot{y}_1 + \dot{y}_0) + Cy_{\frac{1}{2}} \quad .$$

However, we omit the correction operator $Cy_{\frac{1}{2}}$ and use the trapezoidal rule

$$y_1 - y_0 = \tfrac{1}{2}h(\dot{y}_1 + \dot{y}_0) \quad . \tag{2-3}$$

This leads, in our notation, to the vector difference formula

$$\underline{x}(T_{r+1}) - \underline{x}(T_r) = \tfrac{1}{2}h\Big(\underline{f}(\underline{x}(T_{r+1})) + \underline{f}(\underline{x}(T_r))\Big) \qquad (r = 0,\ldots,N-1) \quad \ldots\ldots \tag{2-4}$$

It is convenient to replace the N + 1 vectors $\underline{x}(T_r)$ each having n components by a single vector $\underline{z}$ having n(N + 1) components. Let

$$z_k = x_i(T_r)$$

where

$$k = i + rn \qquad (i = 1,\ldots,n,\ r = 0,\ldots,N). \qquad (2\text{-}5)$$

The equations (2-4) provide nN equations involving the components of $\underline{z}$. A further n equations come from the boundary conditions (2-2), the first ℓ of these equations involving the components $z_1, \ldots, z_n$ and the remaining $(n - \ell)$ equations involving the components $z_{nN+1}, \ldots, z_{nN+n}$. We thus have a total of n(N + 1) nonlinear algebraic equations in n(N + 1) unknowns, and we can write these equations as

$$\underline{g}(\underline{z}) = 0 \qquad (2\text{-}6)$$

where $g_k(\underline{z}) = c^0_{k1}z_1 + c^0_{k2}z_2 + \ldots + c^0_{kn}z_n - d^0_k \qquad (k = 1,\ldots,\ell) \qquad (2\text{-}7)$

$$g_k(\underline{z}) = z_{rn+i} - z_{(r-1)n+i} - \tfrac{1}{2}h\left(f_i(z_{rn+i}, \ldots, z_{rn+n}) + f_i(z_{(r-1)n+1}, \ldots, z_{(r-1)n+n})\right)$$
$$(k = i+rn+\ell,\ i = 1,\ldots,n,\ r = 0,\ldots,N-1) \qquad (2\text{-}8)$$

$$g_k(\underline{z}) = c^F_{j1}z_{nN+1} + c^F_{j2}z_{nN+2} + \ldots + c^F_{jn}z_{nN+n} - d^F_j$$
$$(k = nN+\ell+j,\ j = 1,\ldots,n-\ell). \qquad (2\text{-}9)$$

Newton's method for solving nonlinear equations is well-known[2,3] and will be only briefly described here. Starting with an initial guess $\underline{z}^0$ the method generates successive approximations $\underline{z}^1, \underline{z}^2, \ldots,$ to the true zero $\underline{z}$. The iterative process is defined by

$$\underline{z}^{i+1} = \underline{z}^i - J^{-1}(\underline{z}^i)(\underline{g}(\underline{z}^i)) \qquad (2\text{-}10)$$

where $J(\underline{z}^i)$ is the Jacobian matrix evaluated at $\underline{z}^i$, ie the matrix whose (p,q)th element is $(\partial g_p/\partial z_q)(\underline{z}^i)$. If $\underline{z}^0$ is sufficiently close to $\underline{z}$ the sequence $\underline{z}^0, \underline{z}^1, \underline{z}^2, \ldots$, converges quadratically to $\underline{z}$.

3 COMPUTATIONAL DETAILS

In using equation (2-10) it is not necessary to find the inverse of the Jacobian explicitly. It is sufficient to determine the vector $\underline{\varepsilon}^i$ satisfying

$$J(\underline{z}^i)\underline{\varepsilon}^i = \underline{g}(\underline{z}^i) \qquad (3\text{-}1)$$

and then put

$$\underline{z}^{i+1} = \underline{z}^i + \underline{\varepsilon}^i \quad . \qquad (3\text{-}2)$$

The solution of the set of linear equations (3-1) is made easier by the fact that the matrix $J(\underline{z}^i)$ has most of its elements equal to zero. In fact it has the form shown in Fig 1. This means that use can be made of an algorithm which takes advantage of the band form of the matrix to reduce the amount of storage and computation time required.

In order to calculate $J(\underline{z})$ one must be able to calculate partial derivatives of the functions $f_1, f_2, \ldots, f_n$ with respect to each of the variables $x_1, x_2, \ldots, x_n$. It may be that these partial derivatives are difficult or impossible to calculate directly, for example if the functions are given not in analytical form but as a series of tabular values. In this case it will be necessary to make an approximation to the derivatives by some numerical means. In Appendix B a subroutine is listed which does this using the simplest of finite difference approximations.

The success of Newton's method depends on the distance of the initial guess $\underline{z}^0$ from the true solution $\underline{z}$. The larger this distance is, the greater the number of iterations needed to converge to a given accuracy. If the distance is too large the algorithm may not converge at all, and the method will 'blow-up', ie produce estimates which increase in magnitude rapidly and without limit.

4 SPECIFICATION OF BOVA

Title Solution of two-point boundary value problem using a finite difference method.

Programing language 1900 Fortran.

Description The subroutine produces an approximate solution to the differential equations (1-1) subject to boundary conditions (1-2). The solution is given at a finite number of regularly spaced grid points.

Accuracy Single precision arithmetic is used, and the accuracy obtained depends on the number of grid points employed and on the input parameter ACC.

Experience The subroutine has performed satisfactorily on a number of test cases. See section 5.

Method See sections 2 and 3.

The call statement The CALL statement is

```
CALL BOVA(M1, M2, M3, M4, M5, M6, M7, MAXIT,
          HH, ACC, ICND, Z, NIT, EPS, AA, AM,
          INT, XY, F, FY, P)
```

where M1, M2, M3, M4, M5, M6, M7, MAXIT, NIT are integer variables. HH, ACC are real variables. ICND is an integer array of length at least M1. INT is an integer array of length at least M3. AA is a real array of length at least M4. AM is a real array of length at least M6. XY, F, FY, P are real arrays each of length at least M1.

Input parameters Before calling the subroutine, the user must set the arguments

M1 = n, the number of equations

M2 = N + 1, where N determines the number of grid points $T_0, T_1, T_2, \ldots, T_N$

M3 = M1 × M2

M4 = M3 × (3n - 1)

M5 = ℓ, the number of boundary conditions specified at the initial point T_0

M6 = M3 × (M5 + M1 - 1)

M7 = reporting parameter. If M7 > 0 the subroutine REP is called after every M7 iterations. If M7 $\leq$ 0 REP is not called. See 'subroutines called'.

MAXIT = the maximum number of iterations allowed

HH = h, the distance between grid points = $(T_F - T_0)/N$

ACC = accuracy required. The iterative process is stopped if the norm of the vector difference between successive iterates is less than ACC. The norm used is the uniform or Chebyshev norm:

$$\|\underline{\varepsilon}\|_\infty = \max\{|\varepsilon_i|: \ i = 1, \ldots, M\} \ .$$

ICND(i) = J_i $(i = 1, \ldots, \ell)$
= K_i $(i = \ell + 1, \ldots, n)$
where x_{J_i} is the ith variable specified at the initial point T_0, and x_{K_i} is the $(i - \ell)$th variable specified at the final point T_F.

Results At exit we have

NIT = the number of iterations needed for convergence to the required accuracy. If MAXIT iterations were performed without convergence to the required accuracy NIT is set to -1. If the iterative process 'blows up', ie produces estimates which increase in magnitude rapidly and without limit NIT is set to -2.

$Z(k) = x_i(T_r)$ $(k = i+rn, \ i = 1,\ldots,n, \ r = 0,\ldots,N)$.

State of other variables All the input parameters are left unchanged. The arrays EPS, AA, AM, XY, F, FY, P are used as work areas.

Subroutines called The user must provide the following four subroutines:

(1) Subroutine GUESS(Z,M,HH)

On entry, this subroutine must accept the values of $M = n(N + 1)$ and $HH = (T_F - T_0)/N$ and on termination must provide:

$$Z(k) = x_i^0(T_r) \qquad (k = i+rn, \ i = 1,\ldots,n, \ r = 0,\ldots,N)$$

where $x_i^0(T_r)$ is a guessed approximation to $x_i(T_r)$. Z is a real one-dimensional array of length M.

(2) Subroutine FUN(X,F,M)

On entry this routine must accept the values of $X = \underline{x}$ and $M = n$, and on termination must provide:

$$F(i) = f_i(\underline{x}) \qquad (i = 1,\ldots,n)$$

X and F are real one-dimensional arrays of length M .

(3) Subroutine PARD(X,I,P,M)

On entry this subroutine must accept the values of X = $\underline{x}$, I = i and M = n , and on termination must provide

$$P(j) = \frac{\partial f_i}{\partial x_j}(\underline{x}) \qquad (j = 1,\ldots,n).$$

X and P are real one-dimensional arrays of length M .

(4) Subroutine REP(Z,M,ANORM,IR)

This subroutine enables the user to monitor the progress of the iterative process. On entry

$$M = n(N + 1)$$

$$Z(k) = Z_k^{IR}, \quad \text{the IRth approximation to } x_i(T_r) \qquad (k = i+rn,\ i = 1,\ldots,n,\ r = 0,\ldots,N)$$

$$ANORM = \|\underline{\varepsilon}^{(IR-1)}\|_\infty = \|\underline{Z}^{IR} - \underline{Z}^{(IR-1)}\|_\infty$$

where $\|\underline{\varepsilon}\|_\infty = \max\left\{|\varepsilon_i| : i = 1, \ldots, M\right\}$

Z is a real one-dimensional array of length M .

The ICL subroutines FPDEBM and FPMOVE are also used.

5 TEST RESULTS

We consider the problem

$$\ddot{y} = y^3 - \sin t(1 + \sin^2 t)$$

over the interval 0 to π , with boundary conditions

$$y(0) = y(\pi) = 0$$

which has the solution

$$y(t) = \sin t \quad .$$

Expressing the problem in the form of equation (1-1) it becomes

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_1^3 - \sin x_3(1 + \sin^2 x_3)$$

$$\dot{x}_3 = 1$$

$$x_1(0) = 0$$

$$x_2(\pi) = 0$$

$$x_3(0) = 0 \quad .$$

The numerical results obtained will depend on the number of grid points, the accuracy demanded and the initial guess. The following table gives the results of a number of runs.

Case No.	Maximum error	No. of iterations
1	0.0003	5
2	0.002	5
3	0.00005	5
4	0.0003	6
5	0.0003	6
6	0.0003	9

Case 1 had N = 40, ACC = 0.001 and

$$\left.\begin{aligned} x_1^0(T_r) &= 0 \\ x_2^0(T_r) &= 0 \\ x_3^0(T_r) &= T_r \end{aligned}\right\} r = 0, 1, \ldots, N \quad .$$

Case 2 - as for case 1, except N = 20

Case 3 - as for case 1, except N = 100

Case 4 - as for case 1, except ACC = 0.00001

Case 5 - as for case 1, except the subroutine PARD listed in Appendix B was used to approximate the partial derivatives

Case 6 - as for case 1, except the initial guesses were:

$$\left.\begin{aligned} x_1^0(T_r) &= 10 \\ x_2^0(T_r) &= 10 \\ x_3^0(T_r) &= T_r \end{aligned}\right\} r = 0, 1, \ldots, N \quad .$$

The following equations were also solved:

$$\begin{aligned} \dot{x}_1 &= -x_6 \\ \dot{x}_2 &= -4x_1x_6 \\ \dot{x}_3 &= -3x_1x_7 + 3x_6^3 - 3x_1^2x_6 \\ \dot{x}_4 &= -4x_9 \\ \dot{x}_5 &= -5x_2x_8 - 5x_3x_7 \\ \dot{x}_6 &= x_1 \\ \dot{x}_7 &= 2x_2 \\ \dot{x}_8 &= 3x_1x_2 - 3x_6x_7 \\ \dot{x}_9 &= 4x_1x_3 - 4x_6x_8 \\ \dot{x}_{10} &= 5x_5 \end{aligned}$$

with boundary conditions

$$\begin{aligned} x_1(0) &= x_3(0) = x_5(0) = 1 \\ x_6(0) &= 0 \\ x_3(\pi/4) &= x_5(\pi/4) = x_{10}(\pi/4) = -\sqrt{2}/2 \\ x_4(\pi/4) &= -1 \\ x_7(\pi/4) &= 1 \\ x_9(\pi/4) &= 0 \quad . \end{aligned}$$

The exact solution is

$$x_i(t) = \cos it \qquad (i = 1,\ldots,5)$$

$$x_i(t) = \sin (i - 5)t \qquad (i = 6,\ldots,10).$$

The equations were solved, using subroutine BOVA, with N = 40, ACC = 10^{-8} and initial guesses $x_i^0(T_r) = 0.5$ for all values of i and r except those specified in the boundary conditions. Six iterations were needed for convergence. The maximum error was 0.001, this error occurring in x_8, which was not specified by the boundary conditions at either end of the interval. On the ICL 1906S computer this run took 43 seconds.

Appendix A

LISTING OF BOVA

```
 SUBROUTINE BOVA(M1,M2,M3,M4,M5,M6,M7,MAXIT,HH,ACC
1,ICND,Z,NIT,EPS,AA,AM,INT,XY,F,FY,P)
 INTEGER INT(M3)
 REAL Z(M3),EPS(M3),AA(M4),AM(M6),XY(M1),F(M1),FY(M1),P(
1M1)
 INTEGER ICND(M1)
 COMMON/DIMS/N,NN,NNN,ND1,ND2,NE,ISTA,IRP,H,H2
 N=M1
 NN=M2
 NNN=M3
 ND1=M4
 ND2=M6
 ISTA=M5
 IRP=M7
 NE=NNN-N
 H=HH
 H2=0.5*H
 CALL BVPA(MAXIT,ACC,NIT,Z,EPS,ICND,AA,AM,XY,F,FY
1,P,INT)
 RETURN
 END
```

```
      SUBROUTINE BVPA(MAXIT,ACC,NIT,Z,EPS,ICND,AA,AM,XY,F,FY,P,INT)
C            THIS SUBROUTINE SOLVES A NONLINEAR TWO-POINT BOUNDARY VALUE
C         PROBLEM USING FINITE DIFFERENCE FORMULAE AND A
C            NEWTON-RAPHSON PROCEDURE
C
C         MAXIT IS THE MAXIMUM NUMBER OF ITERATIONS ALLOWED
C            THE PROCESS STOPS WHEN TWO SUCCESSIVE ITERATES AGREE TO AN
C         ACCURACY OF ACC,OR IF THE SOLUTION BLOWS UP
C         NIT IS THE NUMBER OF ITERATIONS NEEDED
C         (-1 IF NO CONVERGENCE,-2 IF SOLUTION BLOWS UP )
C         A CALL IS MADE TO SUBROUTINE REP AFTER EVERY IRP ITERATIONS,PR
OVIDED
C         THAT IRP IS +VE
C            C CONTAINS INFORMATION ABOUT THE BOUNDARY CONDITIONS
C            THE OTHER ARRAYS IN THE SUBROUTINE LIST ARE NEEDED BY IMPRO
VE
C               AT EXIT Z CONTAINS THE SOLUTION
      INTEGER R,INT(NNN)
      REAL Z(NNN),EPS(NNN),AA(ND1),AM(ND2),XY(N),F(N),FY(N),P(N)
      INTEGER ICND(N)
      COMMON/DIMS/N,NN,NNN,ND1,ND2,NE,ISTA,IRP,H,H2
C               FIRST WE MUST MAKE AN INITIAL GUESS AT THE SOLUTION
      CALL GUESS(Z,NNN,H)
      ANORM=0.
      DO 14 I=1,NNN
      AN=ABS(Z(I))
      IF(AN-ANORM)14,14,0
      ANORM=AN
   14 CONTINUE
      BLOW=ANORM*1.E8
C
      DO 3 R=1,MAXIT
      CALL IMPROVE(Z,EPS,ICND,AA,AM,XY,F,FY,P,INT)
C               NEW ITERATE =OLD ITERATE + EPS
C           INFINITY NORM = ANORM
      ANORM=0.
      DO 4 I=1,NNN
      Z(I)=Z(I)+EPS(I)
      AN=ABS(EPS(I))
      IF(AN-ANORM) 4,4,0
      ANORM=AN
    4 CONTINUE
      IF(IRP) 8,8,0
      IF(R/IRP*IRP-R) 8,0,8
      CALL REP(Z,NNN,ANORM,R)
    8 CONTINUE
      IF(ANORM-ACC) 5,0,0
      IF(ANORM-BLOW) 3,0,0
      NIT=-2
      GO TO 9
    3 CONTINUE
      NIT=-1
      RETURN
    5 NIT=R
    9 RETURN
      END
```

```
      SUBROUTINE IMPROVE(Z,EPS,ICND,AA,AM,XY,F,FY,P,INT)
C             THIS SUBROUTINE TAKES A GUESS Z AND CALCULATES EPS SO THAT
C             Z+EPS IS A BETTER APPROXIMATION TO THE SOLUTION THAN Z
C          A NEWTON-RAPHSON METHOD IS USED
      INTEGER R, INT(NNN)
      REAL Z(NNN),EPS(NNN),AA(ND1),AM(ND2),XY(N),F(N),FY(N),P(N)
      INTEGER ICND(N)
      COMMON/DIMS/N,NN,NNN,ND1,ND2,NE,ISTA,IRP,H,H2
C             FIRST ZEROISE THE ARRAY AA WHICH WILL CONTAIN THE MATRIX
      AA(1)=0.
      CALL FMOVE(AA(1),AA(2),ND1-1)
C
C             R DENOTES THE NUMBER OF THE PIVOT POINT (1 TO NN)
C             I DENOTES THE NUMBER OF VARIABLE (1 TO N)
C             NNN=N*NN. THIS IS THE ORDER OF THE MATRIX
C             NE=NNN-N
C             ND1=(M1+M2+1)*NNN , ND2 =M1*NNN ,WHERE M1 AND M2 ARE THE
C             NUMBER OF NONZERO SUBDIAGONALS AND SUPERDIAGONALS OF THE
C             BAND MATRIX A. TO TAKE ADVANTAGE OF THE BAND FORM WE STORE
C             THE NONZERO ELEMENTS OF A IN THE 1-DIMENSIONAL ARRAY AA, SO
 THAT
C             A(I,J) = AA(I+NNN*(M1-(I-J))
C             WE HAVE M1=ISTA+N-1   M2=2N-1-ISTA
C             WHERE ISTA IS THE NO. OF BOUNDARY CONDITIONS AT THE INITIAL
 POINT
C             H IS THE STEP-LENGTH. H2=0.5*H
C             THE RIGHT HAND SIDES OF THE EQUATIONS ARE STORED IN EPS
C             THIS WILL BE OVERWRITTEN LATER
C
C             FIRST DEAL WITH BOUNDARY CONDITIONS
C             THE DIFFERENCE EQUATION PROVIDES (NN-1)*N EQUATIONS
C             THE REMAINING N EQUATIONS COME FROM THE BOUNDARY CONDITIONS
C             SPLIT BETWEEN INITIAL AND FINAL POINTS
      M1=ISTA+N-1
      M2=N+N-1-ISTA
      DO 4 K=1,ISTA
      EPS(K)=0.
      I=ICND(K)
    5 AA(K+NNN*(M1+I-K))=1.
    4 CONTINUE
      DO 14 K=ISTA+1,N
      IW1=NE+K
      EPS(IW1)=0.
      I=ICND(K)
   15 AA(IW1+NNN*(M1+NE+I-IW1))=1.
   14 CONTINUE
```

```
C             NOW FOR THE DIFFERENCE EQUATION
C             EACH ROW OF A CONSISTS OF 2N NONZERO ELEMENTS
C             THE 2ND N COEFFICIENTS OF ROW(ISTA+(R-1)N+I) ARE ALMOST
C             THE SAME AS THE 1ST N OF ROW (ISTA+RN+I)
C             THE COEFFICIENTS INVOLVE PARTIAL DERIVATIVES,AND THE POINTS
 AT
C             WHICH THESE ARE EVALUATED ARE INDEPENDENT OF I, SO WE CAN
C             FIND THESE POINTS IN THE OUTER LOOP
C
C             INITIALISE FY BEFORE STARTING OUTER LOOP
      DO 12 K=1,N
   12 XY(K)=Z(K)
      CALL FUN(XY,FY,N)
C             SET UP 1ST N COEFFICIENTS IN FIRST N EQUATIONS
   31 DO 22 I=1,N
      CALL PARD(XY,I,P,N)
      IRA=ISTA+I
      IW=M1-IRA
C             WE NOW FIND A(IRA,K)  (K=1,...,N)
      DO 23 K=1,N
   23 AA(IRA+NNN*(IW+K))=-H2*P(K)
      IW1=IRA+NNN*(IW+I)
      AA(IW1)=AA(IW1)-1.
   22 CONTINUE
      DO 1 R=1,NN-2
      M=R*N
      L=M-N
      DO 3 I=1,N
      XY(I)=Z(M+I)
C             ALSO NOW FIND FUNCTION VALUES
    3 F(I)=FY(I)
      CALL FUN(XY,FY,N)
C             BEGIN INNER LOOP
   32 DO 2 I=1,N
      IRA=ISTA+(R-1)*N+I
C             FIND PARTIAL DERIVATIVES FOR 2ND N COEFFICIENTS OF
C             EQUATION IRA AND 1ST N COEFFICIENTS OF EQUATION IRA+N
      CALL PARD(XY,I,P,N)
C             WE NOW FIND A(IRA,M+K) AND A(IRA+N,M+K)
      IW=M1+M-IRA
      DO 7 K=1,N
      IW1=IRA+NNN*(IW+K)
      IW2=IW1+N*(1-NNN)
      AA(IW1)=-H2*P(K)
    7 AA(IW2)=AA(IW1)
      IW1=IRA+NNN*(IW+I)
      IW2=IW1+N*(1-NNN)
      AA(IW1)=AA(IW1)+1.
      AA(IW2)=AA(IW2)-1.
```

```
C             NOW FIND EPS(IRA)
      EPS(IRA)=H2*(F(I)+FY(I))-Z(M+I)+Z(L+I)
    2 CONTINUE
    1 CONTINUE
C             WE HAVE MISSED OUT THE 2ND N COEFFICIENTS IN THE LAST N EQU
ATIONS
C             SO SET THESE UP NOW
      M=NE
      L=M-N
      DO 18 I=1,N
      XY(I)=Z(M+I)
   18 F(I)=FY(I)
      CALL FUN(XY,FY,N)
   33 DO 19 I=1,N
      IRA=ISTA+L+I
      CALL PARD(XY,I,P,N)
      IW=M1+M-IRA
      DO 20 K=1,N
      IW1=IRA+NNN*(IW+K)
   20 AA(IW1)=-H2*P(K)
      IW1=IRA+NNN*(IW+I)
      AA(IW1)=AA(IW1)+1.
      EPS(IRA)=H2*(F(I)+FY(I))-Z(M+I)+Z(L+I)
   19 CONTINUE
C             WE HAVE FINISHED SETTING UP THE EQUATIONS AND CAN NOW PROCE
ED
C             TO SOLVE THEM
   34 IW=1
C             AM AND INT ARE WORK ARRAYS USED BY FPDEBM
      CALL FPDEBM(NNN,IW,M1,M2,AA(1),AM(1),EPS(1),INT(1),IV2)
C             EPS HAS BEEN OVERWRITTEN, AND NOW CONTAINS THE REQUIRED SOL
UTION
C
C             CHECK FOR SINGULARITY
      IF(IV2) 0,10,10
      WRITE(2,11)
   11 FORMAT(1X,16H SINGULAR MATRIX)
   21 FORMAT(1X,6E15.6)
      STOP
   10 RETURN
      END
```

Appendix B

LISTING OF SUBROUTINE PARD

```
      SUBROUTINE PARD(X,I,P,N)
      REAL X(N),P(N)
      REAL F(10)
      COMMON /EST/Q(10,10)
      IF(I-1) 5,0,5
      CALL FUN(X,P,N)
      DO 1 J=1,N
      DEL=0.01*X(J)
      IF(ABS(DEL)-1.E-8) 0,0,2
      DEL=1.E-2
    2 XX=X(J)
      X(J)=XX+DEL
      CALL FUN(X,F,N)
      X(J)=XX
      DO 1 K=1,N
    1 Q(K,J)=(F(K)-P(K))/DEL
    5 DO 6 J=1,N
    6 P(J)=Q(I,J)
      RETURN
      END
```

REFERENCES

No.	Author	Title, etc
1	L. Fox	*The numerical solution of two-point boundary problems.* Oxford, Oxford University Press (1957)
2	P. Henrici	*Elements of numerical analysis.* New York, John Wiley & Sons (1964)
3	F.S. Acton	*Numerical methods that work.* New York, Harper & Row (1970)

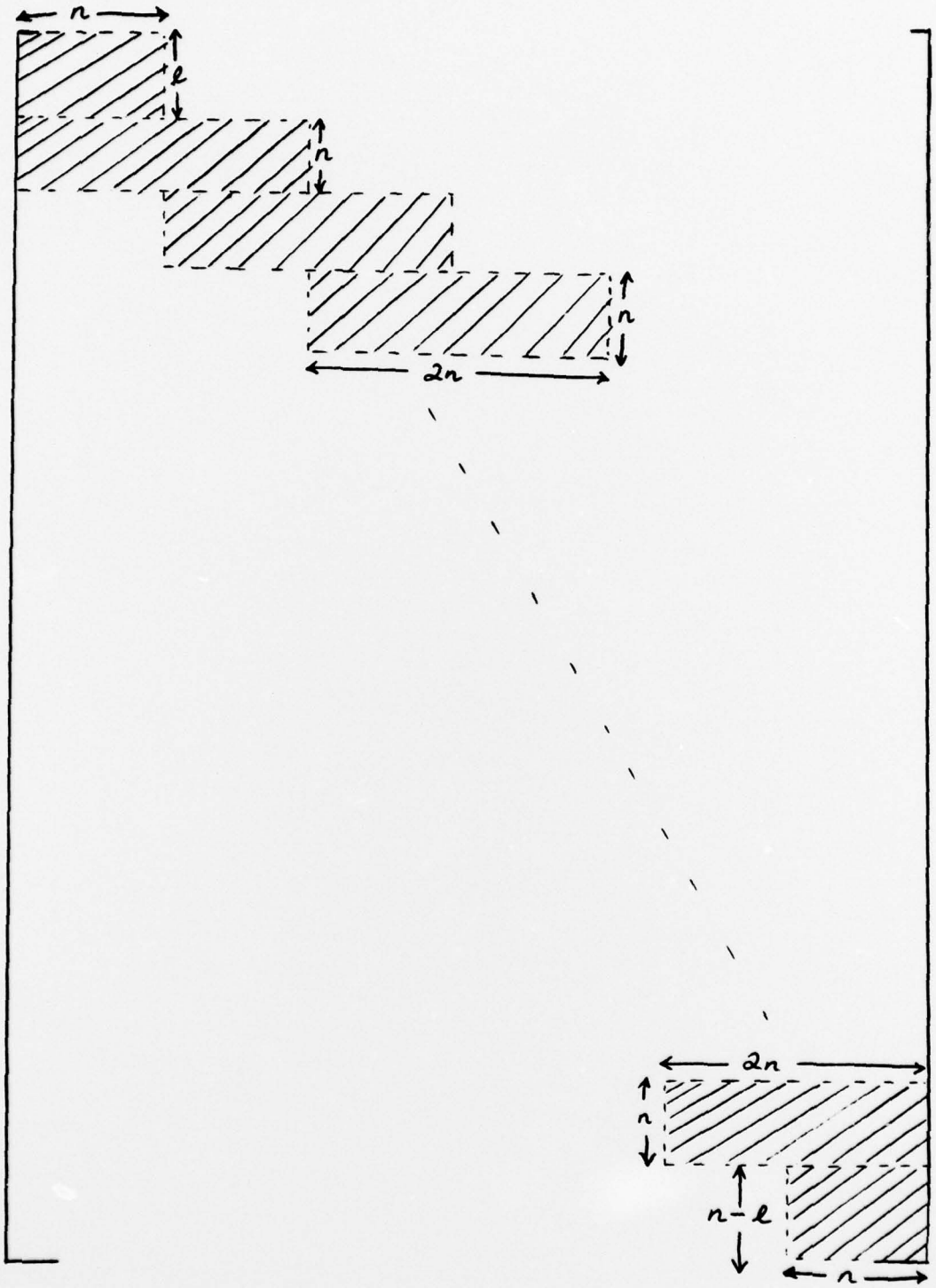

Fig 1 Structure of matrix

REPORT DOCUMENTATION PAGE

Overall security classification of this page

UNLIMITED

As far as possible this page should contain only unclassified information. If it is necessary to enter classified information, the box above must be marked to indicate the classification, e.g. Restricted, Confidential or Secret.

1. DRIC Reference (to be added by DRIC)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification/Marking
	RAE TM Math 7701	N/A	UNLIMITED

5. DRIC Code for Originator	6. Originator (Corporate Author) Name and Location
850100	Royal Aircraft Establishment, Farnborough, Hants, UK
5a. Sponsoring Agency's Code	6a. Sponsoring Agency (Contract Authority) Name and Location
N/A	N/A

7. Title A Fortran subroutine to solve the two-point boundary value problems

7a. (For Translations) Title in Foreign Language

7b. (For Conference Papers) Title, Place and Date of Conference

8. Author 1. Surname, Initials	9a. Author 2	9b. Authors 3, 4	10. Date	Pages	Refs.
Richards, L.J.	-	-	January 1977	20	3

11. Contract Number	12. Period	13. Project	14. Other Reference Nos.
N/A	N/A		

15. Distribution statement

(a) Controlled by – Unlimited

(b) Special limitations (if any) –

16. Descriptors (Keywords) (Descriptors marked * are selected from TEST)

Boundary value problems*. Ordinary differential equations*.

17. Abstract

This Memorandum describes a Fortran subroutine, BOVA, which provides the solution of a set of nonlinear ordinary differential equations subject to non-mixed two-point boundary value conditions. A finite difference method is used, together with Newton's method. A specification and listing of the subroutine is given.

F5910/1

RAE Form A143