TWO MODIFICATIONS TO THE SOFTWARE INTERFACE PACKAGE FOR NONLINEAR PARTIAL DIFFERENTIAL EQUATIONS

by

RICHARD LEE MORSE
B.S., Kansas State University, 1971

A MASTER'S REPORT
submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE
Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1974

Approved by:

[Signature]
Major Professor
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. MODIFICATION OF PDEONE AND ADDITION OF USER SUPPLIED AMATRX ROUTINE TO HANDLE COUPLED SYSTEMS OF PARTIAL DIFFERENTIAL EQUATIONS</td>
<td>3</td>
</tr>
<tr>
<td>2.1. Definition of Problem</td>
<td>3</td>
</tr>
<tr>
<td>2.2. Difference Approximations</td>
<td>5</td>
</tr>
<tr>
<td>2.3. Use of the PDE Interface Package</td>
<td>5</td>
</tr>
<tr>
<td>2.4. Numerical Tests for AMATRX</td>
<td>8</td>
</tr>
<tr>
<td>3. SUBROUTINE PDEJAC TO EFFICIENTLY GENERATE THE JACOBIAN MATRIX NEEDED FOR STIFF ODE METHODS</td>
<td>9</td>
</tr>
<tr>
<td>3.1. Definition of Problem</td>
<td>9</td>
</tr>
<tr>
<td>3.2. Use of PDEJAC With Stiff ODE Integrators</td>
<td>10</td>
</tr>
<tr>
<td>3.3. Numerical Tests Using PDEJAC</td>
<td>12</td>
</tr>
<tr>
<td>4. CONCLUSION</td>
<td>13</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>14</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>15</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>20</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>27</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1. Block Tridiagonal Jacobian Matrix for PDE's and Its Storage in the PW Vector</td>
<td>11</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

PDEONE is a software interface for nonlinear partial differential
equations jointly developed by Dr. Richard F. Sincovec, Kansas State
University and Dr. Niel K. Madsen, Lawrence Livermore Laboratory (1).
It is a piece of computer software which can serve as an interface which
will allow many of the recent significant developments in the field of
ODE's to be applied directly to the numerical solution of PDE's.

The method being implemented by this software package is the
so-called numerical method of lines (2). Roughly speaking, the method
of lines can be described as follows; if one has a time dependent PDE
and discretizes the spatial variables, an approximating system of
ordinary differential equations results. To solve the resulting
equations one uses ODE methods and obtains numerical approximations of
the original PDE.

The software package is designed with user convenience as a goal.
To use this package the user simply defines his system of PDE's and
supplies a spatial mesh to be used for the discretization of the problem
in PDEONE. Then an ODE integrator with its built-in error and stability
controls may be used (3, 4, 5).

This report will discuss two recent changes to the PDE interface
package developed by Sincovec and Madsen (1). The first change being
the modification of the routine PDEONE to handle systems of PDE's that are coupled in the time derivative terms. Next is the addition of the routine PDEJAC to efficiently generate the Jacobian matrix needed when stiff methods are used to solve ordinary differential equations.
Chapter 2

MODIFICATION OF PDEONE AND ADDITION OF USER SUPPLIED AMATRX
ROUTINE TO HANDLE COUPLED SYSTEMS OF NONLINEAR
PARTIAL DIFFERENTIAL EQUATIONS

2.1. Definition of Problem

Partial differential equations may have an unlimited number of
mathematical structures. Following is the structure chosen by
Dr. Richard Sincovec and Dr. Niel Madsen to solve a wide class of
realistic problems (1):

Let NPDE denote the number of PDE's on the interval \([a, b]\) and

\[
\sum_{j=1}^{NPDE} a_{k,j} \frac{\partial u_j}{\partial t} = f_k \left( t, x, u_1, u_2, \ldots, u_{NPDE}, \right.
\]

\[
\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}, \ldots, \frac{\partial u_{NPDE}}{\partial x}, \frac{1}{x} \frac{\partial}{\partial x} \left( x^c D_{k,1} \frac{\partial u_1}{\partial x} \right),
\]

\[
\frac{1}{x} \frac{\partial}{\partial x} \left( x^c D_{k,2} \frac{\partial u_2}{\partial x} \right), \ldots, \frac{1}{x} \frac{\partial}{\partial x} \left( x^c D_{k,NPDE} \frac{\partial u_{NPDE}}{\partial x} \right),
\]

\[a < x < b, \quad t > t_0, \quad k = 1,2,\ldots, NPDE,\]

denote the coupled systems of PDE's with boundary conditions,

\[
\alpha_k u_k + \beta_k \frac{\partial u_k}{\partial x} = \gamma_k \text{ at } x = a \text{ and } b, \quad t > t_0, \quad k = 1,2,\ldots, NPDE,
\]

and initial conditions,
(2.1.3) \( u_k(t_0, x) = \phi_k(x), \quad a \leq x \leq b, \quad k = 1, 2, \ldots, \text{NPDE} \).

If \( \beta_k \neq 0 \) then \( \alpha_k, \beta_k, \gamma_k \) and \( \gamma_k \) may be functions of \( t, x, \) and \( \hat{u} \equiv (u_1, u_2, \ldots, u_{\text{NPDE}}) \), but only functions of \( x \) and \( t \) otherwise; \( D_{k,j} \) and \( a_{k,j} \) \( (k, j = 1, 2, \ldots, \text{NPDE}) \) are functions of \( x, t, \) and \( \hat{u} \); \( c \) is 0, 1, or 2 depending on whether the problem is in Cartesian, cylindrical, or spherical coordinates, respectively.

By assuming that all the coefficient functions, \( \alpha_k, \beta_k, \gamma_k, D_{k,j}, a_{k,j}, f_k, \) and \( \phi_k \), are at least piecewise continuous functions of all their respective variables; problems with physical discontinuities can be defined using the software interface.

Boundary conditions for PDE's are often classified into three types: Dirichlet \( (\beta_k = 0) \), Neumann \( (\alpha_k = 0) \), or mixed \( (\alpha_k \neq 0, \beta_k \neq 0) \). The boundary condition may change with respect to time, as well as from equation to equation. Also the initial condition is not required to satisfy the boundary conditions as \( x \) approaches either \( a \) or \( b \).

The \( (a_{k,j}) \) \((k, j = 1, 2, \ldots, \text{NPDE}) \) matrix allows for a coupling of the time derivative of systems of parabolic PDE's and/or hyperbolic PDE's. The coupling may be nonlinear as each \( a_{k,j} \) may be a function of \( \hat{u} \).

Note that problem (2.1.1) - (2.1.3) is completely defined if one specifies the interval, \([a, b]\); the initial time, \( t_0 \); the vector functions \( f = (f_k), \alpha = (a_k), \beta = (\beta_k), \gamma = (\gamma_k) \) \((k = 1, 2, \ldots, \text{NPDE})\); the matrix functions \( D = (D_{k,j}), A = (a_{k,j}) \) \((k, j = 1, 2, \ldots, \text{NPDE})\); and the initial conditions \( \phi_k(x), \) \( k = 1, 2, \ldots, \text{NPDE} \). With the software interface
developed here, the user will be required to write four basic sub-
programs to define: the matrix D, the matrix A, the vector f, and the
boundary conditions (i.e., the vectors α, β, and γ); plus a main program
to specify the spatial mesh and initial conditions, to set a flag to
signal the presence of an A matrix, to call the integrator, and to
print the results.

To prevent PDEONE from continually decomposing the A matrix
and solving for the $A^{-1} f_k, k = 1,2,\ldots,\text{NPDE}$ on the right-hand side the
user has the option of setting the value of MATRIX either to 0 or 1. If
MATRIX = 1, the routine will perform a LU decomposition of the A matrix
and solve for the right-hand side where $\frac{3u_k}{\partial t} = A^{-1} f_k, k = 1,2,\ldots,\text{NPDE}$. If MATRIX = 0, PDEONE will solve $\frac{3u_k}{\partial t} = f_k, k = 1,2,\ldots,\text{NPDE}$ and there
will be no coupling of the PDE systems. The user will not have to write
the identity matrix for A and continually have it decomposed.

2.2. Difference Approximations

Once the user has defined an appropriate PDE system and the
 corresponding spatial mesh, the software interface uses consistent
centered difference approximations to convert the PDE system into an
approximating initial value system of ordinary differential equations (1).

2.3. Use of the PDE Interface Package

To use the PDE interface the user should have both a stiff and
nonstiff integrator. Unless a problem is known a priori to be stiff,
the problem should be used with a nonstiff integrator first. Because
the integrators developed by Gear (3) and Hindmarsh (4, 5) have both
stiff and nonstiff methods built into the same program they should be
considered in implementing the PDE interface. Refer to Madsen and
Sincovec (2) for a more complete discussion of the use of ODE integrators
for solving PDE's.

To use the PDE interface (given in appendix A) the user is
required to write four subroutines defining his system of PDE's in order
for the interface to discretize the PDE's and convert them into a system
of approximating ODE's. These subroutines are: D(T, X, U, DVAL, NPDE)
defining the $D_{k,j}$ ($k,j = 1,2,\ldots,\text{NPDE}$) of equation (2.1.1), $F(T, X, U,$
$UX, DUXX, FVAL, NPDE)$ defining the right-hand side of (2.1.1), BNDRY
(T, X, U, ALPHA, BETA, GAMMA, NPDE) defining the boundary conditions
(2.1.2), and AMATRX(NPDE, T, X, U) defining the coupling of the PDE's.
Refer to Sincovec and Madsen (1) for specific details on subroutines D,
F, and BNDRY. Specific details for subroutine AMATRX will be presented
in this paper.

In subroutine AMATRX, T and X are scalar quantities representing
respectively the current time and spatial variable, while U is a vector
quantity with entries $U_k$ ($k = 1,2,\ldots,\text{NPDE}$). Approximate values of the
preceding variables are passed to the user's routines by the PDE inter-
face. The user must be careful to neither divide by zero nor perform
other noncomputable operations. The user is cautioned to make sure the
A matrix is neither singular nor apt to become singular at any point
during the integration.
a. Subroutine AMATRIX is constructed as follows:

```
SUBROUTINE AMATRIX(NPDE, T, X, U)
COMMON/AMAT/ A(NPDE, NPDE), IP(NPDE), MATRIX
DIMENSION U(NPDE)
```

Here $A(k,j)$ ($k,j = 1,2,...,NPDE$) are defined. These are the values of the coupling equations $a_{k,j}$, appearing in (2.1.1). All the matrix entries $A(k,j)$ must be defined by the user (even if zero) unless the value of $\text{MATRIX} = 0$ is set in the main calling program, signifying no $A$ matrix is needed. Should $\text{MATRIX} = 0$ the values of $A(k,j)$ need not be defined.

RETURN
END

The COMMON/AMAT/ card must be replaced entirely while the actual value of NPDE (number of partial differential equations) needs to replace NPDE at each occurrence.

At the present time, the user must include the following card in the main program, PDEONE, PDEJAC, and AMATRIX.

```
COMMON/AMAT/A(NPDE, NPDE),
    IP(NPDE), MATRIX
```

The remainder of this paper will assume the usage of GEARB (4) with the driver, DRIVEB, and its modifications as used by Sincovec and Madsen (1). GEARB will be further modified to calculate the block tridiagonal Jacobian matrix by calling the subroutine PDEJAC once as opposed to the usual method of repeated calls to PDEONE. Refer to Chapter 3 of this paper for a discussion of the change.
2.4. Numerical tests for AMATRX

If AMATRX is a linear combination of systems of equations (i.e., constants, T, and X are the only quantities allowed in the values of \((a_{k,j})\) \((k,j = 1, 2, \ldots, NPDE)\) it is relatively simple to show that AMATRX is working. One combines the FVAL terms of subroutine F in the same order as AMATRX. The PDE's solved are \(\frac{\partial u}{\partial t} = A^{-1}A f = f\). This holds true for all the numerical examples given by Sincovec and Madsen (1).

Given the following problem:

\[
\begin{align*}
\frac{\partial u_1}{\partial t} + \frac{\partial u_2}{\partial t} &= \frac{1}{x} \frac{\partial}{\partial x} \left( x \frac{\partial u_1}{\partial x} \right) \\
\frac{\partial u_1}{\partial t} + \frac{\partial u_2}{\partial t} &= 0.
\end{align*}
\]

This problem was solved with

\[
\begin{align*}
a_{1,1} &= u_2 & a_{1,2} &= 0. \\
a_{2,1} &= 0. & a_{2,2} &= 1.
\end{align*}
\]

with boundary conditions

\[
\begin{align*}
\frac{\partial u_1}{\partial x} (t, 0) &= 0. \\
\frac{\partial u_1}{\partial x} (t, 1) &= 1.73E-9(6.25E+10-\left[u(t, 1) \right]^4) \\
u_2 (t, 0) &= u_2 (t, 1) = 2.
\end{align*}
\]

and initial conditions given by

\[
\begin{align*}
u_1 (0, x) &= 600. & u_2 (0, x) &= 2.
\end{align*}
\]
Chapter 3

SUBROUTINE PDEJAC TO EFFICIENTLY GENERATE THE JACOBIAN MATRIX NEEDED FOR STIFF ODE METHODS

3.1. Definition of Problem

An ODE system is said to be stiff if it involves both very rapidly changing and very slowly changing terms all of a decaying nature. When the problems are stiff they must be solved implicitly with the following system of nonlinear equations being solved at each step (3, 5).

Most of the methods used in solving stiff problems are based on the linear multistep formula.

\[ y_n = \sum_{j=1}^{K_1} \alpha_j y_{n-j} + h \sum_{j=0}^{K_2} \beta_j y_{n-j} \]

(3.1.1)

For Gear's method of order q \( K_1 = q \) and \( K_2 = 0 \), \( \alpha_j \) and \( \beta_j \) are constants associated with the method, \( h \) is the stepsize, \( y_k \) is an approximation to \( y(t_k) \), and \( \dot{y}_k \) is an approximation to \( \dot{y}(t_k) = f(y_k, t_k) \). Equation (3.1.1) can be written in the following form.

\[ g(y_n) \equiv y_n - h \beta_0 f(y_n, t_n) - \sum_{j=1}^{K_1} \alpha_j y_{n-j} - h \sum_{j=1}^{K_2} \beta_j y_{n-j} = 0 \]

(3.1.2)

One can then solve the nonlinear system \( g(y_n) \) by Newton's method.

\[ y_{n(m+1)} = y_{n(m)} - p^{-1}_{n(m)} g(y_{n(m)}) \]

(3.1.3)

\[ p_{n(m)} = \frac{\partial g}{\partial y} \bigg|_{y_{n(m)}} = I - h \beta_0 \frac{\partial f}{\partial y} \bigg|_{y_{n(m)}} \]
In practice, it is much less costly to replace the matrix $P_n(m)$ by

$$ (3.1.4) \quad P_n = I - h\beta_0 J_n, \quad J_n = \frac{\partial f}{\partial y} \bigg| y_n(0) $$

at a slight loss of the rate of convergence. One can also use a "chord method" by utilizing $P_{n'}$ for some $n' < n$ instead of $P_n$ at step $n$.

ODE integrators approximate the Jacobian matrix, $J_n$, by successive calls to a routine that computes $DY/DT$ for a given $T$ and $Y$. This is accomplished by Sincovec and Madsen (1) by NPDE*NPTS calls to PDEONE with each call calculating NPDE*NPTS values. Since the Jacobian matrix of equation (3.1.4) is block tridiagonal for partial differential equations of the structure (2.1.1), one need only calculate $(3*NPTS)$ blocks of NPDE*NPDE values of the Jacobian with NPDE being the number of partial differential equations and NPTS being the number of spatial grid points (Figure 1).

Because $h$ and $\beta_0$ are known, one can calculate the values for $-h\beta_0 J_n$ in one subroutine call, thus reducing the number of times the program will set up linkage to the subroutine.

Subroutine PDEJAC is written to calculate $-h\beta_0 J_n$ in one call.

It calculates only the partial derivative needed for the block tridiagonal matrix of Figure 1.

3.2. Use of PDEJAC with Stiff ODE Integrators

PDEJAC is an interface subroutine which generates the block tridiagonal matrix used in solving partial differential equations. It generates values for $-h\beta_0 J_n, \quad J_n = \frac{\partial f}{\partial y} \bigg| y_n(0)$ and stores them in a vector

and stores them in a vector...
NPDE = 3
NPTS = 6
ML = 5

= NPDE * NPDE Block
\ = PW Vector

Figure 1: Block Tridiagonal Jacobian Matrix for PDE's and Its Storage in the PW Vector
called PW. This routine will not calculate any partials outside the block tridiagonal form of Figure 1.

The PW vector is stored from the lowermost band upward (Figure 1). PDEJAC uses the four subroutines referred to in Chapter 2 (D, F, BNDRY, and AMATRX) while calculating and storing the \(- h B J_n\) matrix in the PW vector.

Since the Jacobian matrix is calculated in one call, the user must be certain to pass the appropriate values to subroutine PDEJAC from the stiff ODE integrator. The call for PDEJAC is of the form:

PDEJAC(N, NO, NPDE, NPTS, ML, T, U, UDOT, EL, FSAVE, YMAX, PW, H)

with the arguments described in Appendix B.

3.3. Numerical Tests Using PDEJAC

This writer has found if one used PDEJAC with the stiff ODE integrator, GEARB modified slightly to call PDEJAC only once, one can use the PDE interface developed by Sincovec and Madsen (1) much more efficiently.

Using the examples given by Sincovec and Madsen (1), this writer has realized a forty percent decrease in computer time to solve their example E (Cylindrical Problem) with twenty-one grid points. Five of the remaining six problems were solved with even more significant savings as the number of grid points increased.

To verify the correctness of PDEJAC, the user need only check to see if the answers using PDEJAC are identical to the answers not using PDEJAC. The user may also see if the PW vectors are the same in both cases.
Chapter 4

CONCLUSION

The author of this paper believes that the PDE interface presented by Sincovec and Madsen (1) is a significant beginning for robust software for reasonably broad classes of partial differential equations. With the additions and modifications presented in this paper, the PDE interface package will more efficiently solve more general types of equations.

With the addition of AMATRIX the user may now solve systems of coupled parabolic and hyperbolic equations. If the user does not have a system of coupled equations, essentially no efficiency will be lost in the original PDE interface (1) when MATRIX is set to zero.

One of the largest areas of concern with the PDE interface is the generation of the Jacobian matrix for stiff equations. Most ODE integrators are capable of automatically generating the Jacobian matrix of equation 3.1.4. It is a convenient but frequently expensive feature if either NPDE*NPTS or the number of Jacobian generations are large.

Since a stiff integrator is absolutely essential for Sincovec and Madsen's approach to solving PDE's to be robust, the author feels his subroutine PDEJAC may create a significant change in the computer time used in solving partial differential equations. By using PDEJAC the user can save forty percent of his computer time and have the ODE integrator generate the Jacobian matrix needed for stiff equations.
REFERENCES


APPENDIX A

SUBROUTINE PDEONE(NPDE,NPTS,T,U,UDOT)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION U(NPDE,NPTS),UDOT(NPDE,NPTS)
C
C PDEONE IS AN INTERFACE SUBROUTINE WHICH CONVERTS COUPLED ONE
C DIMENSIONAL SYSTEMS OF PARTIAL DIFFERENTIAL EQUATIONS
C INTO A SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS, UDOT
C = F(T,X,U), BY CENTERED DIFFERENCE APPROXIMATIONS.
C THIS ROUTINE IS INTENDED TO BE USED WITH A ROBUST
C ODE INTEGRATOR.
C
C INPUT...
C NPDE = NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C NPTS = NUMBER OF SPATIAL GRID POINTS.
C T = CURRENT VALUE OF TIME.
C U = AN NPDE BY NPTS ARRAY CONTAINING THE COMPUTED
C SOLUTION AT TIME T.
C
C OUTPUT...
C UDOT = AN NPDE BY NPTS ARRAY CONTAINING THE RIGHT HAND
C SIDE OF THE RESULTING SYSTEM OF ODE'S, F(T,X,U)
C OBTAINED BY DISCRETIZING THE GIVEN PDE'S.
C
C THE USER MUST INSERT A DIMENSION STATEMENT AND A COMMON
C STATEMENT OF THE FOLLOWING FORMS:
C
C DIMENSION DVAL(**,**), UX(**), UAVG(**), ALPHA(**),
C * BETA(**), GAMMA(**)
C
C COMMON /AMAT/ A(**,**), IP(**), MATRIX
C
C THE SYMBOLS ** DENOTE THE ACTUAL NUMERICAL VALUE OF NPDE
C FOR THE PROBLEM BEING SOLVED.
C
C COMMON BLOCK AMAT CONTAINS THE (A) MATRIX, THE TRANSFORMATION
C VECTOR USED IN DECOMPOSING THE MATRIX, AND THE FLAG
C SIGNIFYING THE MATRIX IS PRESENT.
C
C COMMON BLOCK COORD CONTAINS 0,1, OR 2 DEPENDING ON WHETHER
C THE PROBLEM IS IN CARTESIAN, CYLINDRICAL, OR SPHERICAL
C COORDINATES, RESPECTIVELY.
C
C COMMON BLOCK MESH CONTAINS THE USER SPECIFIED SPATIAL
C GRID POINTS.
C
C COMMON /MESH/ XI(11)
C COMMON /COORD/ ICORD
ICORD1 = ICORD + 1
C
C UPDATE BOUNDARY VALUES AT THE LEFT BOUNDARY
C
CALL BNDRY(T,X(1),U,ALPHA,BETA,GAMMA,NPDE)
ITEST = 0
C
PDE00010
PDE00020
PDE00030
PDE00040
PDE00050
PDE00060
PDE00070
PDE00080
PDE00090
PDE0100
PDE0110
PDE0120
PDE0130
PDE0140
PDE0150
PDE0160
PDE0170
PDE0180
PDE0190
PDE0200
PDE0210
PDE0220
PDE0230
PDE0240
PDE0250
PDE0260
PDE0270
PDE0280
PDE0290
PDE0300
PDE0310
PDE0320
PDE0330
PDE0340
PDE0350
PDE0360
PDE0370
PDE0380
PDE0390
PDE0400
PDE0410
PDE0420
PDE0430
PDE0440
PDE0450
PDE0460
PDE0470
PDE0480
PDE0490
PDE0500
PDE0510
PDE0520
PDE0530
PDE0540
DO 10 K=1,NPDE
   IF (FTRA(K).NE.0.0) GO TO 10
   U(K,1) = GAMMA(K)/ALPHA(K)
   ITEST = ITEST + 1
10 CONTINUE
   DXI = 1./(X(2)-X(1))
   IF (ITEST.EQ.0) GO TO 20
   IF (ITEST.EQ.NPDE) GO TO 45
   CALL BNDHY(T,X(1),U,ALPHA,BETA,GAMMA,NPDE)
   C C EVALUATE 'DIFFUSION COEFFICIENTS', D, AT THE LEFT BOUNDARY
   C 20 CALL D(T,X(1),U,DVAL,NPDE)
   C C FORM APPROXIMATION TO DU/DX AT THE LEFT BOUNDARY
   C DO 40 K=1,NPDE
      IF(BETA(K).NE.0.0) GO TO 30
      UXIK = DXI *(U(K,2) - U(K,1))
      GO TO 40
30   UXIK = (GAMMA(K) - ALPHA(K)*U(K,1))/BETA(K)
   40 CONTINUE
C C EVALUATE U-AVERAGE IN THE FIRST INTERVAL
C DO 50 K=1,NPDE
   UAVG(K) = .5*(U(K,2) + U(K,1))
50 CONTINUE
C C EVALUATE 'DIFFUSION COEFFICIENTS', D, IN THE FIRST INTERVAL
C   XAVGR = .5 * ( X(2) + X(1) )
   CALL D(T,XAVGR,UAVG,DVAL(1,1,2),NPDE)
   DXIR = DXI
   DXIL = 1.
   IF (ICORD.EQ.0) GO TO 55
   DXIL = X(1)**ICORD
   C XIR = XAVGR**ICORD + DXI
   C 55 DXIC = FLOAT(ICORD1) / (XAVGR**ICORD1 - X(1)**ICORD1)
C C EVALUATE DXUX AT THE LEFT BOUNDARY
C DO 60 K=1,NPDE
   DO 60 L=1,NPDE
      DVAL(K,L,1) = DXIC*(DVAL(K,L,2)*(U(L,2)-U(L,1))*DXIL -
                        DVAL(K,L,1)*UX(L)*DXI)
60 CONTINUE
   IF (ITEST.EQ.NPDE) GO TO 65
C C EVALUATE RIGHT HAND SIDE OF PDE'S AT THE LEFT BOUNDARY
C CALL F(T,X(1),U,UX,DVAL,UDCT,NPDE)
C C DECOMPOSE THE (A) MATRIX AND SOLVE FOR THE RIGHT HAND
C SIDE IF NECESSARY.
C IF (MATRIX.EQ.0) GO TO 65
CALL AMATRX(NPDE,T,X(I),UX)
CALL DECMATP(NPDE,NPDE,A,IP)
CALL SCLVE(NPDE,NPDE,A,UCOT(1),1,IP)
C
C SET UDOT = 0 FOR KNOWN LEFT BOUNDARY VALUES
C
65 GO TO 70 K = 1,NPDE
IF (BETA(K) .EQ. 0.0) UDOT(K) = 0.0
70 CONTINUE
C
C UPDATE BOUNDARY VALUES AT THE RIGHT BOUNDARY
C
CALL BNDRY(I,T,X(NPTS),U(I,NPTS),ALPHA,BETA,GAMMA,NPDE)
ITEST = 0
DO 75 K = 1,NPDE
IF (BETA(K) .NE. 0.0) GO TO 75
U(K,NPTS) = GAMMA(K)/ALPHA(K)
ITEST = ITEST + 1
75 CONTINUE
C
C MAIN LOOP TO FCRM ODE'S AT THE INTERIOR GRID POINTS
C
IBCK = 1
IFWD = 2
ILIM = NPTS - 1
DO 100 I = 2,ILIM
ITEMP = IBCK
IBCK = IFWD
IFWD = ITEMP
XAVGL = XAVGR
XAVGR = .5*(X(I+1) + X(I))
DXI = 1./IX(I+1) - X(I-1))
DXIL = DXIR
DXIR = 1.
IF (ICORD .NE. 0) DXIR = XAVGR**ICORD
DXIC = FLCAT(IORD1) / (XAVGR**ICORD1 - XAVGL**ICORD1)
C
C EVALUATE DU/DX AND U-AVERAGE AT THE I-TH GRID POINT
C
DO 80 K = 1,NPDE
UX(K) = DXI*(U(K,I+1) - U(K,I-1))
UAVG(K) = .5*(U(K,I+1) + U(K,I))
80 CONTINUE
C
C EVALUATE 'DIFFUSION COEFFICIENTS', D, IN THE I-TH INTERVAL
C
CALL DIT,XAVGR,UAVG,CVAL(I,I,IFWD,NPDE)
C
C EVALUATE DUXX AT THE I-TH GRID POINT
C
DO 90 K = 1,NPDE
DO 90 L = 1,NPDE
DVAL(K,L,IBCK) = DXIC*(DVAL(K,L,IFWC)*(U(L,I+1) - U(L,I)) - DVAL(K,L,IBCK)*(U(L,I)) - U(L,I-1) * DXIL)
90 CONTINUE
C EVALUATE RIGHT HAND SIDE OF PDE'S AT THE I-TH GRID POINT
C
CALL FIT(X(I),U(1,I),UX,DVAL(1,1,IBCK),UDOT(1,I),NPDE)
C
C DECOMPOSE THE (A) MATRIX AND SOLVE FOR THE RIGHT HAND
C SIDE IF NECESSARY.
C
IF (MATRIX.EQ.0) GO TO 100
CALL AMATRIX(NPDE,T,X(I),U(1,I))
CALL DECOMP(NPDE,NPDE,A,IP)
CALL SOLVE(NPDE,NPDE,A,UDOT(1,I),IP)
100 CONTINUE
C FINISH UPDATED THE RIGHT BOUNDARY IF NECESSARY
C
IF (ITTER.ME.EQ.0) GO TO 120
IF(ITTER.EQ.NPDE) GO TO 155
CALL BNDRT(T,X(NPTS),U(1,NPTS),ALPHA,BETA,GAMA,NPDE)
C
C EVALUATE 'DIFFUSION COEFFICIENTS', D, AT THE RIGHT BOUNDARY
C
120 CALL D(T,X(NPTS),U(1,NPTS),DVAL(1,1,IBCK),NPDE)
C
C FORM APPROXIMATIONS TO DU/DX AT THE RIGHT BOUNDARY
C
DXI = 1. / (X(NPTS) - X(ILIM))
DO 140 K=1,NPDE
IF(TMETA(K),NE.0) GO TO 130
UX(K) = DXI*(U(K,NPTS) - U(K,ILIM))
GO TO 140
130 UX(K) = (DAMA(K) - ALPHA(K)*U(K,NPTS))/BETA(K)
140 CONTINUE
DXIL = DXIR
DXIR = 1
IF (ICORD.NE.0) DXIR = X(NPTS)**ICORD
DXIC = FLOA(ICORD1)/X(NPTS)**ICORD1-XAVR**ICORD1
C
C EVALUATE DUXX AT THE RIGHT BOUNDARY
C
DO 150 K=1,NPDE
DO 150 L=1,NPDE
DVAL(K,L,IBCK) = DXIC*(DVAL(K,L,ECK)*UX(L)*DXIR
* -DVAL(K,L,IFWD)*(U(L,NPTS)-U(L,ILIM))*DXIL)
150 CONTINUE
C
C EVALUATE RIGHT SIDE OF PDE'S AT THE RIGHT BOUNDARY
C
CALL FIT(X(NPTS),U(1,NPTS),UX,DVAL(1,1,IBCK),
* UDOT(1,NPTS),NPDE)
C
C DECOMPOSE THE (A) MATRIX AND SOLVE FOR THE RIGHT HAND
C SIDE IF NECESSARY.
C
IF (MATRIX.EQ.0) GO TO 155
CALL AMATRIX(NPDE,T,X(NPTS),U(1,NPTS))
CALL DECOMP(NPDE,NPDE,A,IP)
CALL SCLVE(NPDE,NPDE,A,UDOT(1,NPTS),IP)

C
C SET UDOT = 0 FOR KNOWN RIGHT BOUNDARY VALUES
C
155 DO 160 K=1,NPDE
   IF (BETA(K).EQ.0.0) UDOT(K,NPTS)=0.
160 CONTINUE
RETURN
END
APPENDIX B

SUBROUTINE POEJACIN, KO, NPDE, NPTS, ML, T, U, UDOT, EL, FSAVE,
 & YMAX, PW, H)
(IMP PLICIT REAL A-H, Q-Z)
DIMENSION U(NPDE, NPTS), UDOT(NPDE, NPTS), FSAVE(NO),
 & YMAX(NO), PW(H)
C
C POEJAC IS AN INTERFACE SUBROUTINE WHICH GENERATES THE
C BLOCK TRIDIAGONAL JACOBIAN MATRIX NEEDED FOR STIFF METHODS
C WHEN SOLVING PARTIAL DIFFERENTIAL EQUATIONS. THIS
C ROUTINE WAS WRITTEN TO BE COMBINED WITH STIFF INTEGRATORS
C AND CALCULATE THE JACOBIAN MATRIX IN ONE CALL.
C THE VALUES OF L-kJ0 = JACOBIAN NEEDED FOR STIFF METHODS ARE
C CALCULATED IN THIS ROUTINE AND STORED IN PW IN THE
C APPROPRIATE PLACES. PW IS STORED FROM THE LOWERMOST
C BAND UPWARD WITH N LOCATIONS FOR EACH BAND.
C
C INPUT:
C N = NUMBER OF ODE'S, IS EQUAL TO NPDE*NPTS
C NO = NUMBER OF DIFFERENTIAL EQUATIONS INITIALLY.
C NPDE = NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C NPTS = NUMBER OF SPATIAL GRID POINTS.
C ML = BAND WIDTH OF THE LOWER BAND.
C T = CURRENT VALUE OF TIME.
C U = AN NPDE BY NPTS ARRAY CONTAINING THE COMPUTED
C SOLUTION AT TIME T.
C UDOT = AN NPDE BY NPTS ARRAY CONTAINING THE RIGHT HAND
C SIDE OF THE RESULTING SYSTEM OF ODE'S, F(T, X, U)
C OBTAINED BY DISCRETIZING THE GIVEN PDE'S.
C EL = METHOD COEFFICIENT.
C FSAVE = VECTOR CONTAINING ORIGINAL VALUE OF THE DERIVATIVE.
C YMAX = VECTOR CONTAINING MAXIMUM VALUE OF EACH Y SEEN.
C H = THE STEP SIZE TO BE ATTEMPTED BY THE INTEGRATOR.
C
C OUTPUT:
C PW = A VECTOR CONTAINING THE PARTIAL DERIVATIVES FOR THE
C JACOBIAN MATRIX.
C
C THE USER MUST INSERT A DIMENSION STATEMENT AND A COMMON
C STATEMENT OF THE FOLLOWING FORMS:
C DIMENSION DVAL(*,*,2), UX(*), UAVG(*), ALPH(*),
C BETA(*), GAMMA(*),
C COMMON /AMAT/ A(*,*,*), IP(*), MATRIX
C THE SYMBOLS ** DENOTE THE ACTUAL NUMERICAL VALUE OF NPDE
C FOR THE PROBLEM BEING SOLVED.
C COMMON BLOCK AMAT CONTAINS THE (A) MATRIX, THE TRANSFORMATION
C VECTOR USED IN DECOMPOSING THE MATRIX, AND THE FLAG
C SIGNIFYING THE MATRIX IS PRESENT.
C COMMON BLOCK COORD CONTAINS 0,1, OR 2 DEPENDING ON WHETHER
C THE PROBLEM IS IN CARTESIAN, CYLINDRICAL, OR SPHERICAL
C
PDJ00010
PDJ00020
PDJ00030
PDJ00040
PDJ00050
PDJ00060
PDJ00070
PDJ00080
PDJ00090
PDJ01000
PDJ01010
PDJ01020
PDJ01030
PDJ01040
PDJ01050
PDJ01060
PDJ01070
PDJ01080
PDJ01090
PDJ01200
PDJ01210
PDJ01220
PDJ01230
PDJ01240
PDJ01250
PDJ01260
PDJ01270
PDJ01280
PDJ01290
PDJ01300
PDJ01310
PDJ01320
PDJ01330
PDJ01340
PDJ01350
PDJ01360
PDJ01370
PDJ01380
PDJ01390
PDJ01400
PDJ01410
PDJ01420
PDJ01430
PDJ01440
PDJ01450
PDJ01460
PDJ01470
PDJ01480
PDJ01490
PDJ01500
PDJ01510
PDJ01520
C COORDINATES, RESPECTIVELY.
C COMMON BLOCK MESH CONTAINS THE USER SPECIFIED SPATIAL
C GRID POINTS.
C
COMMON/COORD/ICORD
COMMON /MESH/ X(I)
C
ANoise = 2.22044605D-16
EPSJ = DsQRT(ANoise)
N1 = 0
NOML = NO * ML
NM1 = NO - 1
DO = 0.
DI = D1 + FSAVE(I)*2
RO = DAR3(HI)/DQRT(D1)*1.0D0*ANoise
J1 = NOML
C
C CALCULATE THE NEEDED PARTIALS WITH RESPECT TO EACH Y.
C
DO 290 J=1,N
N1 = N1 + 1
J1 = J1 + KO
YJ = U(J,1)
R = EPSJ*MAX(J)
R = DMAX1(R,RO)
UT(J,1) = U(J,1) + R
DI = -EL*H/R
IWORK = IWORK + 1
C
C CALCULATE WHICH BLOCK THE PARTIAL IS NEEDED IN.
C
JEQ = ((J - 1) / NPDE) + 1
ILIM = NPTS - 1
ICORD1 = ICORD + 1
IF (JEQ.GE.ILIM) GO TO 72
C
C CALCULATE UDOT WITHOUT ANY BOUNDARY CONDITIONS.
C
IF (JEQ.GT.2.AND.JEQ.LT.ILIM) GO TO 83
C
C UPDATE BOUNDARY VALUES AT THE LEFT BOUNDARY
C
CALL BNDRY1(T,X(I),U,ALPHA,BETA,GAMMA,NPDE)
IEST = 0
DO 10 K=1,NPDE
IF (BETA(K).NE.0.0) GO TO 10
U(K,1) = GA4*K(A)/ALPHA(K)
IEST = IEST + 1
10 CONTINUE
D1 = 1./X(2)-X(1))
IF (IEST.EQ.0) GO TC 20
IF (IEST.EQ.NPDE) GO TO 45
CALL BNDRY1(T,X(I),U,ALPHA,BETA,GAMMA,NPDE)
C
C EVALUATE 'DIFFUSION COEFFICIENTS', D, AT THE LEFT BOUNDARY
C
C 20 CALL D(T,X(1),U,OVAL,NOPE)
C
C FORM APPROXIMATION TO DU/DX AT THE LEFT BOUNDARY
C
DO 40 K=1,NOPE  
   IF (HETA(K).NE.0.0) GO TO 30  
   UX(K) = DXI *(U(K,2) - U(K,1))  
   GO TO 40
30   UX(K) = (GAMMA(K) - ALPHA(K)*U(K,1))/HETA(K)
   40 CONTINUE
C
C EVALUATE U-AVERAGE IN THE FIRST INTERVAL
C
45 DO 50 K=1,NOPE  
   UAVG(K) = .5*(U(K,2) + U(K,1))  
50 CONTINUE
C
C EVALUATE *DIFFUSION COEFFICIENTS*, D, IN THE FIRST INTERVAL
C
   XAVGR = .5 * (X(2) + X(1))  
   CALL D(T,XAVGR,UAVG,OVAL(1,1,2),NOPE)
   DXIR = DXI
   DXIL = 1.
   IF (ICORD.EQ.0) GO TO 55
   DXIL = X(1)**ICORD
   DXIR = XAVGR*ICORD*DXI
   55 DXIC = FLT(XICD1) / (XAVGR**ICORD1 - X(1)**ICORD1)
C
C EVALUATE DUXX AT THE LEFT BOUNDARY
C
DO 60 K=1,NOPE  
   DO 61 L=1,NOPE  
      OVAL(K,L,1) = DXIC*(OVAL(K,L,2)*(U(L,2)-U(L,1)))*DXIR -  
      OVAL(K,L,1)*UX(L)*DXIL
   60 CONTINUE
C
C EVALUATE RIGHT HAND SIDE OF PDE'S AT THE LEFT BOUNDARY
C
CALL F(T,X(1),U,UX,OVAL,UDCT,NOPE)
C
C DECOMPOSE THE (A) MATRIX IF NECESSARY
C
IF (MATRIX.EQ.0) GO TO 65
   CALL AM hard(NODES,T,X(1),U)
   CALL DECOMP(NODES,NOPE,A,IP)
   CALL SOLVE(NODES,NOPE,A,UDOT(1,1),IP)
65 CONTINUE
C
C SET UDOT = 0 FOR KNOWN LEFT BOUNDARY VALUES
C
60 DO 70 K=1,NOPE  
   IF (BETA(K).EQ.0.0) UDOT(K,1)=0.0
70 CONTINUE
C
C UPDATE BOUNDARY VALUES AT THE RIGHT BOUNDARY
C
CALL RNDYRI(T,X(NPTS),U(1,NPTS),ALPHA,BETA,GAMMA,NPDE)
  ITEST = 0
  DO 75 K=1,NPDE
    IF(BETA(K).GE.0.0) GO TO 75
    U(K,NPTS) = GAMMA(K)/ALPHA(K)
    ITEST = ITEST + 1
  CONTINUE
  75 IF(JEQ.EQ.1) GO TO 79
  IF(JEQ.EQ.2) GO TO 80
  IF(JEQ.EQ.ILIM) GO TO 81
  C CALCULATE ONE BLOCK BEFORE THE RIGHT BOUNDARY.
  C
  IBEG = ILIM
  IEND = ILIM
  II = IBEG - 1
  GO TO 87
  C CALCULATE ONE BLOCK AFTER THE LEFT BOUNDARY.
  C
  79 IBEG = 2
  IEND = 2
  II = 1
  GO TO 84
  C CALCULATE TWO BLOCKS AFTER THE LEFT BOUNDARY.
  C
  80 IBEG = 2
  IEND = 3
  II = 1
  GO TO 84
  C CALCULATE TWO BLOCKS BEFORE THE RIGHT BOUNDARY.
  C
  81 IBEG = NPTS - 2
  IEND = ILIM
  II = IBEG - 1
  GO TO 87
  C CALCULATE THREE BLOCKS WITHOUT BOUNDARY CONDITIONS.
  C
  83 IBEG = JEQ - 1
  IEND = JEQ + 1
  II = IBEG - 1
  C MAIN LOOP TO FORM ODE'S AT THE INTERIOR GRID POINTS
  C
  87 XAVGR = .5*(X(IBEG) + X(IBEG-1))
  DXIR = 1.
  IF(ICCRO.EQ.0) DXIP = XAVGR**ICORD
  DXIR = DXIR/(X(IBEG) - X(IBEG-1))
  DO 88 K=1,NPDE
    UAVG(K) = .5*(U(K,IBEG) + U(K,IBEG-1))
  CONTINUE
  CALL DIT,XAVGR,UAVG,CVAL(1,1,2),NPDE
  C SET ZEROES ONE BLOCK BEFORE.
C
DO 93 I=1,NPDE
93 UDN(:,Ibeg-I) = 0.
84 IBCK = 1
IFWD = 2
I2 = Iend+1
IF (JEQ.GE.ILIM) GO TO 97

C
SET ZERODES ONE BLOCK AFTER.
C
DO 95 I=1,NPDE
95 UDOT(:,Iend+1) = 0.
97 DO 100 I=IBEG,IEND
ITEMP = IBCK
IBCK = IFWD
IFWD = ITEMP
XAVGR = XAVGR
XAVGR = .5*(X(I+1) + X(I))
DX1 = 1./X(I+1)-X(I)
DXIL = DXIR
DXIR = 1.
IF (ICORD.GE.0) DXIR = XAVGR**ICORD
DXIR = DXIR / (X(I+1)-X(I))
EXIC = FLCAT(ICORD) / (XAVGR**ICORD1 - XAVGR**ICORD1)

C
EVALUATE DU/DX AND U-AVERAGE AT THE I-TH GRID POINT
C
DO 85 K=1,NPDE
UX(K) = DX1*(U(K,I+1) - U(K,I-1))
UAVG(K) = .5*(U(K,I+1) + U(K,I))
85 CONTINUE

C
EVALUATE DIFFUSION COEFFICIENTS', D, IN THE I-TH INTERVAL
C
CALL DIT,XAVGR,UVG,H((1,1),IFWD),NPDE)

C
EVALUATE DUXX AT THE I-TH GRID POINT
C
DO 90 K=1,NPDE
DO 90 L=1,NPDE
DVAL(K,L,IBCK) = DXIC*(DVAL(K,L,IFWD)*U(L,I+1) -
* U(L,I)) + DXIR - DVAL(K,L,IBCK)*U(L,I)
90 CONTINUE

C
EVALUATE RIGHT HAND SIDE OF PDE'S AT THE I-TH GRID POINT
C
CALL F(I,X(I),U(I),UX,DVAL(1,1),IBCK),UDOT(1,1),NPDE)

C
DECOMPOSE THE (A) MATRIX IF NECESSARY
C
IF (MATRIX,F.O.G) GO TO 100
CALL AMATRIX(NPDE,I,X(I),U(1,1))
CALL ECOPY(NPDE,NPDE,A,P)
CALL SULVE(NPDE,NPDE,A,UDOT(1,1),P)
100 CONTINUE

C
C FINISH UPDATING THE RIGHT BOUNDARY IF NECESSARY

C IF (JEO.LT.IL1) GO TO 165
110 IF (ITEST.EQ.2) GO TO 120
IF (ITEST.EQ.NPDE) GO TO 155
CALL AMDRYX(INPTS),U(1,NPTS),ALPHA,BETA,GAMMA,NPDE)
C
C EVALUATE DIFFUSION COEFFICIENTS**, D, AT THE RIGHT BOUNDARY
C
120 CALL DIT(INPTS),U(1,NPTS),DVAL(1,1,IRCK),NPDE)
C
C FORM APPROXIMATIONS TO DU/DX AT THE RIGHT BOUNDARY
C
DXI = 1. / (INPTS) - X(ILIM))
DO 140 K=1,NPDE
   IF (BETA(K).NE.0.,O) GO TO 130
   UX(K) = DXI *(U(K,NPTS) - U(K,ILIM))
   GO TO 140
130 UX(K) = (GAMMA(K) - ALPHA(K)*U(K,NPTS))/BETA(K)
140 CONTINUE
DXIL = DXIR
DXXIR = 1
IF (ICORD.NE.0) DXIR = X(INPTS)**ICORD
   DXIC = FLOAT(ICORD) / (INPTS)**ICORD - XAVG**ICORD)
C
C EVALUATE DUXX AT THE RIGHT BOUNDARY
C
DO 150 K=1,NPDE
   DO 150 L=1,NPDE
      DVAL(K,L,IRCK) = DXIC*(DVAL(K,L,IRCK)*UX(L)*DXIR
         - U(L,NPTS)-U(L,ILIM)*DXIL)
150 CONTINUE
I2 = NPTS
C
C EVALUATE RIGHT HAND SIDE OF PDE'S AT THE RIGHT BOUNDARY
C
CALL FIT(INPTS),U(1,NPTS),UX,DVAL(1,1,IRCK),
   UDCNT(1,NPTS),NPDE)
C
C DECOMPOSE THE A MATRIX IF NECESSARY
C
IF (MATRIX.EQ.0) GO TO 155
CALL AMATRIX(NPDE,T,XINPTS),U(1,NPTS)
CALL DECOMP(NPDE,NPDE,A,IP)
CALL SOLVE(NPDE,NPDE,A,UDCNT(1,NPTS),IP)
C
C SET UDOT = 0 FOR KNOWN RIGHT BOUNDARY VALUES
C
155 DO 160 K=1,NPDE
   IF (BETA(K).EQ.0.,O) UDOKT(K,NPTS)=0.
160 CONTINUE
165 CONTINUE
120 I = 11,12
   IBEG = 1
   IEND = NPDE
C
C CALCULATE THE NEEDED PW'S FOR BANDS.
C
IF (1.EQ.12.AND.N1.EQ.1.AND.JEQ.LT.IL1) GO TO 285
IF (1.EQ.11.AND.N1.EQ.NPDE.AND.JEQ.GT.2) GO TO 285
IF (1.EQ.11.AND.JEQ.GT.2) IBEG = N1 + 1
IF (1.EQ.12.AND.JEQ.LT.IL1) IEND = N1 - 1
DO 280 L=IBEG,IEND
  K = (I-1)*NPDE+L
  JSAVE = J1-NM1*K
  PW(JSAVE) = (UDOT(L,1) - FSAVE(K))*D1
  IF (UDOT(L,1).EQ.0.) PW(JSAVE) = 0.
280 CONTINUE
285 CONTINUE
IF (N1.EQ.NPDE) N1 = 0
290 UI(J,1) = VJ
RETURN
END
ACKNOWLEDGMENTS

The author wishes to express his sincere thanks and appreciation to his advisory committee: Dr. Richard F. Sincock, Associate Professor of Computer Science at Kansas State University; Dr. Nasir Ahmed, Associate Professor of Computer Science at Kansas State University; and Dr. Paul S. Fisher, Head of the Department of Computer Science at Kansas State University. A special note of appreciation is given to Dr. Sincock for providing the basis of this report and for giving freely of his time. Finally special thanks to Teresa Morse, for her many hours of help in preparing and editing this paper.
TWO MODIFICATIONS TO THE SOFTWARE INTERFACE PACKAGE
FOR NONLINEAR PARTIAL DIFFERENTIAL EQUATIONS

by

RICHARD LEE MORSE
B.S., Kansas State University, 1971

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1974
ABSTRACT

This paper presents two modifications to a software interface package for nonlinear partial differential equations being jointly developed by Dr. Richard F. Sincovec, Kansas State University and Dr. Niels K. Madsen, Lawrence Livermore Laboratory. The first change discussed is a modification of the PDEONE routine so that the package can handle coupled sets of partial differential equations. The second modification is the addition of the PDEJAC routine that will generate the Jacobian matrix in an efficient manner.