GENERAL SOFTWARE FOR TWO DIMENSIONAL PARTIAL DIFFERENTIAL EQUATIONS

by

David Kennett Melgaard

B. S., New Mexico State University, 1970

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

1976

Approved by:

Major Professor

GENERAL SOFTWARE FOR TWO DIMENSIONAL PARTIAL DIFFERENTIAL EQUATIONS

1. INTRODUCTION

Very little general purpose software for the numerical solution of partial differential equations (PDE's) now exists; so many of the time consuming and difficult processes involved in solving PDE's numerically still remain. Until recently all software was designed to solve specific problems or at best a small class of related problems [4, 9, 10, 12, 14, 15]. However, robust software developed by Madsen and Sincovec [8] is now available for the solution of a moderately wide class of one dimensional initial value nonlinear PDE's. This software utilizes the numerical method of lines [7]. Basically this method consists of discretizing all but one of the independent variables (usually the spatial variables). The resulting system of ordinary differential equations (ODE's) (usually in time) is then solved by an ODE integrator. The motivation for this approach is that the "state of the art" of the solution of systems of ODE's is much more sophisticated than that for PDE's. Current ODE integrators (Gear [1],Hindmarsh [2,3],Krogh [5],Shampine and Gordon [11]) can determine the solution of stiff nonlinear ODE's, provide dynamic capabilities for altering step size and method order to maintain stability and preserve a user specified accuracy. Therefore, by using the method of lines, a large portion of the complexity of developing suitable general software is passed to the already developed robust integrator.

We have two basic purposes in this paper. The primary purpose is to present a general software interface for two dimensional PDE's and discuss its capabilities, limitations, and ease of use. The other purpose is to discuss the matrix problem, related specifically to the numerical solution of two dimensional PDE's. We include this because efficient solution of the matrix problem is crucial in the solution of many nonlinear systems of PDE's.

The general software interface presented in Appendix A and described

in this paper is designed to simplify the solution of PDE's having one independent variable in time and two independent spatial variables. No capable robust software now exists for this problem. This software interface is a natural extension of the software developed by Madsen and Sincovec [8]. The extension consists of increasing the number of spatial variables to two and changing from three point centered differencing to five point centered differencing. Since many of the detailed comments in their paper are applicable to our interface, the reader would be well advised to familiarize himself with that paper. With this software interface, the user is required only to provide the spatial mesh and define the problem to be evaluated. The interface will then convert this system of PDE's into a system of ODE's which the ODE integrator then solves.

For the solution of nonlinear problems a Jacobian matrix is required. Since the resulting system of ODE's can become quite large for two dimensional spatial meshes, an efficient method of generating the Jacobian is essential. Consequently we also include in this paper a discussion of the related matrix problem. The Jacobian generated from five point differencing has a well defined banded structure. The emphasis in this paper will be placed on optimizing the number of function evaluations needed to generate this banded matrix. We also provide a modification in Appendix B to improve the efficiency of one of the available robust integrators, GEARB by Hindmarsh [2].

This paper consists of seven main sections. In the following section we define the general class of PDE's allowed by this interface. Included in that section are comments about some of the limitations of this class of PDE's. Section 3 defines the difference approximations used by the interface to convert the system of PDE's to a system of ODE's. The efficient generation of the Jacobian matrix in relation to this software interface is discussed in Section 4. The fifth section provides the detailed description of the use of the interface.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

With the minimal effort described in that section one will be able to obtain accurate results for a moderately wide class of PDE's. Section 6 lists some nontrivial numerical examples designed to indicate the capabilities, limitations, and ease of use of the interface. We conclude with a section containing general observations about the efficient use of the interface, comments about its validity, and its space and time requirements.

2. GENERAL PDE DEFINITION

We now define the problem structure allowed by the software interface. We believe this structure includes a reasonably wide range of the possible PDE forms.

Let NPDE denote the number of PDE's over the region R for which a solution, $u_\ell(t,x,y)$ ($\ell = 1,2,\ldots,NPDE$) is desired. R is defined by $R \equiv \{(x,y) \mid a_1 \le x \le b_1,\ a_2 \le y \le b_2\}$. The coupled system of PDE's is defined as :

$$\begin{aligned}(2.1)\qquad \frac{\partial u_\ell}{\partial t} = f\Bigg(&t,x,y,u_1,\ldots,u_{NPDE},\frac{\partial u_1}{\partial x},\ldots,\frac{\partial u_{NPDE}}{\partial x},\frac{\partial u_1}{\partial y},\ldots,\frac{\partial u_{NPDE}}{\partial y},\\ &\frac{\partial}{\partial x}\left(DH_{\ell,1}\frac{\partial u_1}{\partial x}\right),\ldots,\frac{\partial}{\partial x}\left(DH_{\ell,NPDE}\frac{\partial u_{NPDE}}{\partial x}\right),\\ &\frac{\partial}{\partial y}\left(DV_{\ell,1}\frac{\partial u_1}{\partial y}\right),\ldots,\frac{\partial}{\partial y}\left(DV_{\ell,NPDE}\frac{\partial u_{NPDE}}{\partial y}\right)\Bigg)\\ &a_1 < x < b_1,\quad a_2 < y < b_2,\quad t > t_0,\quad \ell = 1,2,\ldots,NPDE,\end{aligned}$$

with horizontal boundary conditions,

$$(2.2)\qquad AH_\ell u_\ell + BH_\ell \frac{\partial u_\ell}{\partial y} = CH_\ell \quad \text{at } a_1 \le x \le b_1,\ y = a_2 \text{ or } y = b_2,$$

$$\ell = 1,2,\ldots,NPDE,\quad t > t_0,$$

vertical boundary conditions,

$$(2.3)\qquad AV_\ell u_\ell + BV_\ell \frac{\partial u_\ell}{\partial x} = CV_\ell \quad \text{at } x = a_1 \text{ or } x = b_1,\ a_2 \le y \le b_2,$$

$$\ell = 1,2,\ldots,NPDE \ , \ t > t_0 \ ,$$

and initial conditions

(2.4) $\quad u_\ell(t_0,x,y) = \phi_\ell(x,y) \ , \quad$ for $(x,y) \in R \ , \quad \ell = 1,2,\ldots,NPDE \ .$

All functions f_ℓ, $DH_{\ell,k}$, $DV_{\ell,k}$, AV_ℓ, BV_ℓ, CV_ℓ, AH_ℓ, BH_ℓ, CH_ℓ and ϕ_ℓ are at least piecewise continuous functions of all their respective variables. $DV_{\ell,k}$ ($DH_{\ell,k}$), the diffusion coefficient in the vertical (horizontal) direction is a function of t,x,y and $\vec{u}$. If $BV_\ell \neq 0$ ($BH_\ell \neq 0$) then the boundary coefficients, AV_ℓ, BV_ℓ and CV_ℓ (AH_ℓ, BH_ℓ and CH_ℓ) may be functions of t,x,y and $\vec{u} \equiv (u_1,u_2,\ldots,u_{NPDE})$, but otherwise they may only be functions of x,y and t. All three types of boundary conditions, Dirichlet (BH_ℓ or $BV_\ell = 0$), Neuman (AH_ℓ or $AV_\ell = 0$) or mixed ($AH_\ell \neq 0$, $AV_\ell \neq 0$, $BH_\ell \neq 0$, $BV_\ell \neq 0$), are handled in such a way that there is no need to distinguish between types. This allows the flexibility of using any boundary type with each PDE and of changing the type with respect to time. Also the initial conditions need not be consistent with the boundary conditions (i.e. satisfy the boundary conditions as x or y approaches a boundary).

One should be aware of some of the limitations of this problem definition. As noted in the following section, the difference approximations for an ODE depends on the location of the spatial mesh point associated with that ODE. To minimize the complexity and overhead of the interface, the region R is restricted to the shape of a rectangle. Also the difference approximations require the mesh to be defined with a minimum of three mesh lines in each direction. Consequently this interface can only be used to solve PDE's having one independent variable in time and two independent spatial variables (i.e. $a_1 \neq b_1$ and $a_2 \neq b_2$). The other limitation we note is that no provision is made for terms of the form $\frac{\partial^2 u}{\partial x \partial y}$. Certainly even with these limitations, the problem (2.1)-(2.4) still encompasses a reasonably wide class of PDE's.

3. SPATIAL VARIABLE DIFFERENCING

From the user defined problem (2.1)-(2.4), the software interface generates five point centered difference approximations to convert the system of PDE's to a system of ODE's. This section provides the detailed definition of these difference approximations. In order to obtain meaningful results one does not need to understand the internal details. However we feel that the better one understands our methods the better use he can make of the interface.

The user definition includes the spatial mesh, the points where a function, $u_{\ell,i,j}(t)$ approximates the true solution $u_\ell(t,x_i,y_j)$ to the system of PDE's. This mesh consists of the intersection of a sequence of (NX) lines drawn parallel to the horizontal axis with a sequence of (NY) lines drawn parallel to the vertical axis. These lines intersect the axis at $a_1 = x_1 < x_2 < \ldots < x_{NX} = b_1$ and $a_2 = y_1 < y_2 < \ldots < y_{NY} = b_2$ respectively. The mesh spacings are defined as $\Delta x_i = x_{i+1} - x_i$ and $\Delta y_j = y_{j+1} - y_j$. Careful consideration is required in choosing the mesh as the accuracy of the solutions is dependent on selecting a suitable mesh.

The differencing used to generate the ODE depends on the position of its associated mesh point. The difference equations presented in this paper were derived using an integration technique similar to that mentioned in Varga [13]. The differencing depends on whether the mesh point is 1) in the interior of the mesh, 2) on a side of the mesh but not on a corner or 3) on a corner. We will present each of these cases separately. In all cases we denote the approximating function $u_{\ell,i,j}(t)$ by $u_{\ell,i,j}$.

Case 1

We define first the approximating difference equations necessary to evaluate (2.1) for the mesh point (x_i,y_j) when $1 < i < NX$ and $1 < j < NY$. This

requires approximating values for the arguments $\vec{u}$, $\vec{u}_x$, $\vec{u}_y$, $\frac{\partial}{\partial x}\left(DH_{\ell,k}\frac{\partial \vec{u}}{\partial x}\right)$ and $\frac{\partial}{\partial y}\left(DV_{\ell,k}\frac{\partial \vec{u}}{\partial y}\right)$ (k = 1,2,...,NPDE) of (2.1). The difference approximations we use for the $\ell^{\underline{th}}$ PDE are:

$$(3.2a)\qquad u_k(t,x,y) \approx u_{k,i,j}$$

$$(3.2b)\qquad \frac{\partial u_k(t,x,y)}{\partial x} \approx \frac{u_{k,i+1,j} - u_{k,i-1,j}}{\Delta x_i + \Delta x_{i-1}}$$

$$(3.2c)\qquad \frac{\partial u_k(t,x,y)}{\partial y} \approx \frac{u_{k,i,j+1} - u_{k,i,j-1}}{\Delta y_j + \Delta y_{j-1}}$$

$$(3.2d)\qquad \frac{\partial}{\partial x}\left(DH_{\ell,k,i,j}\frac{\partial u_k(t,x,y)}{\partial x}\right) \approx \frac{1}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}}$$

$$\left(DH_{\ell,k,i+\frac{1}{2},j}\left(\frac{u_{k,i+1,j} - u_{k,i,j}}{\Delta x_i}\right) - DH_{\ell,k,i-\frac{1}{2},j}\left(\frac{u_{k,i,j} - u_{k,i-1,j}}{\Delta x_{i-1}}\right)\right),$$

$$(3.2e)\qquad \frac{\partial}{\partial y}\left(DV_{\ell,k,i,j}\frac{\partial u_k(t,x,y)}{\partial y}\right) \approx \frac{1}{y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}}$$

$$\left(DV_{\ell,k,i,j+\frac{1}{2}}\left(\frac{u_{k,i,j+1} - u_{k,i,j}}{\Delta y_j}\right) - DV_{\ell,k,i,j-\frac{1}{2}}\left(\frac{u_{k,i,j} - u_{k,i,j-1}}{\Delta y_{j-1}}\right)\right),$$

where

$$(3.2f)\qquad x_{i\pm\frac{1}{2}} \equiv \frac{x_{i\pm1} + x_i}{2}, \qquad y_{j\pm\frac{1}{2}} \equiv \frac{y_{j\pm1} + y_j}{2},$$

$$\vec{u}_{k,i\pm\frac{1}{2},j} \equiv \left(\frac{u_{1,i\pm1,j} + u_{1,i,j}}{2}, \ldots, \frac{u_{NPDE,i\pm1,j} + u_{NPDE,i,j}}{2}\right),$$

$$\vec{u}_{k,i,j\pm\frac{1}{2}} \equiv \left(\frac{u_{1,i,j\pm1} + u_{1,i,j}}{2}, \ldots, \frac{u_{NPDE,i,j\pm1} + u_{NPDE,i,j}}{2} \right) ,$$

$$DH_{\ell,k,i\pm\frac{1}{2},j} \equiv DH_{\ell,k}\left(t, x_{i\pm\frac{1}{2}}, y_j, \vec{u}_{k,i\pm\frac{1}{2},j}\right) ,$$

$$DV_{\ell,k,i,j\pm\frac{1}{2}} \equiv DV_{\ell,k}\left(t, x_i, y_{j\pm\frac{1}{2}}, \vec{u}_{k,i,j\pm\frac{1}{2}}\right) .$$

Case 2

For the second case the approximating difference equations are defined for a mesh boundary point (x_i, y_j) on either a horizontal ($1 < i < NX$, $j = 1$ or $j = NY$) or a vertical ($i = 1$ or $i = NX$, $1 < j < NY$) boundary. We will only show the difference equations for a point on the lower horizontal boundary ($j = 1$) as the equations for the upper horizontal boundary and both the left and right vertical boundaries are completely analogous.

When constant boundary conditions ($BH_\ell = 0$) exist at the point being evaluated, $u_{\ell,i,1}$ is determined exactly from (2.2) by $u_{\ell,i,1} = \frac{CH_\ell}{AH_\ell}$. Clearly no differential equation needs to be generated in this case. However to preserve the structure of the system of ODE's we define $\frac{du_{\ell,i,1}}{dt} = 0$.

For a point on the lower boundary with changing boundary condition, approximating equations are required for all the arguments of the function (2.1). $\vec{u}_x$ and $\frac{\partial}{\partial x}\left(DH_{\ell,k}\frac{\partial u_k}{\partial x}\right)$ remain the same as defined for a point in the center of the mesh. We will therefore only define equations for $\vec{u}$, $\vec{u}_y$, and $\frac{\partial}{\partial y}\left(DV_{\ell,k}\frac{\partial u_k}{\partial y}\right)$. For the ℓ^{th} PDE at (x_i, y_j), $1 < i < NX$, $j = 1$ with $BH_\ell \neq 0$, we use the approximations as follows:

(3.3a) $$u_k(t,x,y) \approx u_{k,i,1} ,$$

(3.3b)
$$\frac{\partial u_k(t,x,y)}{\partial y} \approx \begin{cases} \dfrac{u_{k,i,2} - u_{k,i,1}}{\Delta y_1} , & BH_k = 0 , \quad k \neq \ell , \\ \dfrac{CH_k - AH_k u_{k,i,1}}{BH_k} , & BH_k \neq 0 , \end{cases}$$

(3.3c)

(3.3d)
$$\frac{\partial}{\partial y}\left[DV_{\ell,k}\,\frac{\partial u_k(t,x,y)}{\partial y}\right] \approx \begin{cases} \dfrac{1}{y_{1+\frac{1}{2}} - y_1}\left[\left(DV_{\ell,k,i,1+\frac{1}{2}} - DV_{\ell,k,j,1}\right)\left(\dfrac{u_{k,i,2} - u_{k,i,1}}{\Delta y_1}\right)\right] , \\ \text{if } BH_k = 0, \quad k \neq \ell , \\ \dfrac{1}{y_{1+\frac{1}{2}} - y_1}\left[DV_{\ell,k,i,1+\frac{1}{2}}\left(\dfrac{u_{k,i,2} - u_{k,i,1}}{\Delta y_1}\right) - DV_{\ell,k,i,1}\left(\dfrac{CH_k - AH_k u_{k,i,1}}{BH_k}\right)\right] , \\ \text{if } BH \neq 0 , \end{cases}$$

(3.3e)

where ($k = 1,2,\ldots,NPDE$), $DV_{\ell,k,i,1\pm\frac{1}{2}}$, $y_{1\pm\frac{1}{2}}$ are defined by (3.2f) with $j = 1$. We note that the approximations (3.3b) and (3.3d) are not five point centered difference equations. Clearly if $BH_k \neq 0$, the boundary condition provides an exact value to $\frac{\partial u_k}{\partial y}$ in (3.3c) and (3.3e). However for the case $BH_k = 0$, $k \neq \ell$, an approximation is needed for the argument $\frac{\partial u_k}{\partial y}$ of the function f_ℓ. Therefore these one-sided approximations (3.3b) and (3.3d) are required.

Case 3

The final case is to evaluate (2.1) at (x_i, y_j) where $i = 1$ or $i = NX$ and $j = 1$ or $j = NY$. We will again only show the equations for one case ($x = x_1$ and $y = y_1$) as the other three corners are completely analogous. Since the point is affected by both the horizontal and vertical boundary

conditions we must consider the three possibilities: 1) when both are constant, 2) when only one is constant, and 3) when neither boundary condition is constant.

The first case is when constant horizontal and vertical boundary conditions ($BH_1 = BV_1 = 0$) are present at (x_1,y_1). To most nearly approximate a realistic solution, an average of the two boundary conditions is taken. Hence we define:

$$(3.4a) \qquad u_{\ell,1,1} = \frac{1}{2}\left(\frac{CH_\ell}{AH_\ell} + \frac{CV_\ell}{AV_\ell}\right), \qquad BH_\ell = BV_\ell = 0 .$$

Consider now (x_1,y_1) with one boundary condition constant and the other changing. We assume that in a realistic sense the constant boundary condition is more apt to "dominate" the solution. Hence $u_{\ell,1,1}$ is defined in this case to be the constant boundary condition, i.e. we define:

$$(3.4b) \qquad u_{\ell,1,1} = \frac{CH_\ell}{AH_\ell}, \qquad BH_\ell = 0, \quad BV_\ell \neq 0 ,$$

$$(3.4c) \qquad u_{\ell,1,1} = \frac{CV_\ell}{AV_\ell}, \qquad BH_\ell \neq 0, \quad BV_\ell = 0 .$$

For these first two cases we consider the approximations (3.4a)-(3.4c) to be exact, so no ODE is required. As for Case 2, we set $\frac{du_{\ell,1,1}}{dt} = 0$ to preserve the structure of the system of ODE's.

The last possibility, when both boundary conditions are not constant, requires the interface to evaluate (2.1), i.e. find approximating values for $\vec{u}$, $\vec{u}_x$, $\vec{u}_y$, $\frac{\partial}{\partial x}\left(DH_{\ell,k}\frac{\partial \vec{u}}{\partial x}\right)$ and $\frac{\partial}{\partial y}\left(DV_{\ell,k}\frac{\partial \vec{u}}{\partial y}\right)$. For the ℓ^{th} PDE at $x = x_1$, $y = y_1$, $BH_\ell \neq 0$, and $BV_\ell \neq 0$ these approximations are:

$$(3.4d) \qquad u_k(t,x,y) \approx u_{k,1,1}$$

$$(3.4e) \qquad \frac{\partial u_k(t,x,y)}{\partial x} \approx \begin{cases} \dfrac{u_{k,2,1} - u_{k,1,1}}{\Delta x_1}, & BV_k = 0, \quad k \neq \ell, \\ \dfrac{CV_k - AV_k u_k}{BV_k}, & BV_k \neq 0, \end{cases}$$

$$(3.4f)\qquad \frac{\partial u_k(t,x,y)}{\partial y} \;\approx\; \begin{cases} \dfrac{u_{k,1,2} - u_{k,1,1}}{\Delta y_1}, & BH = 0,\ k \neq \ell, \\ \dfrac{CH_k - AH_k u_{k,1,1}}{BH_k}, & BH_k \neq 0, \end{cases}$$

$$(3.4g)\qquad \frac{\partial}{\partial x}\left[DH_{\ell,k}\,\frac{\partial u_k(t,x,y)}{\partial x}\right] \;\approx\; \begin{cases} \dfrac{1}{x_{1+\frac12} - x_1}\left[(DH_{\ell,k,1+\frac12,1} - DH_{\ell,k,1,1})\left(\dfrac{u_{k,2,1} - u_{k,1,1}}{\Delta x_1}\right)\right], \\ \text{if } BV_k = 0,\ k \neq \ell, \\ \dfrac{1}{x_{1+\frac12} - x_1}\left[DH_{\ell,k,1+\frac12,1}\left(\dfrac{u_{k,2,1} - u_{k,1,1}}{\Delta x_1}\right) - DH_{\ell,k,1,1}\left(\dfrac{CV_k - AV_k u_{k,1,1}}{BV_k}\right)\right], \\ \text{if } BV_k \neq 0, \end{cases}$$

$$(3.4h)\qquad \frac{\partial}{\partial y}\left[DV_{\ell,k}\,\frac{\partial u_k(t,x,y)}{\partial y}\right] \;\approx\; \begin{cases} \dfrac{1}{y_{1+\frac12} - y_1}\left[(DV_{\ell,k,1,1+\frac12} - DV_{\ell,k,1,1})\left(\dfrac{u_{k,1,2} - u_{k,1,1}}{\Delta y_1}\right)\right], \\ \text{if } BH_k = 0,\ k \neq \ell, \\ \dfrac{1}{y_{1+\frac12} - y_1}\left[DV_{\ell,k,1,1+\frac12}\left(\dfrac{u_{k,1,2} - u_{k,1,1}}{\Delta y_1}\right) - DV_{\ell,k,1,1}\left(\dfrac{CH_k - AH_k u_{k,1,1}}{BH_k}\right)\right], \\ \text{if } BH_k \neq 0, \end{cases}$$

where $DH_{\ell,k,1+\frac12,1}$, $x_{1+\frac12}$, $DV_{\ell,k,1,1+\frac12}$ and $y_{1+\frac12}$ are defined by (3.2f) with i = 1

and $j = 1$.

The motivation for handling the three possibilities differently is to keep the interface simple and yet provide the same general boundary conditions (2.2)-(2.3) at the corners. We feel that knowledge and judicious use of these three possibilities will allow the user to define boundary conditions consistent with physically realistic situations.

The spatial variable differencing results in the system of ODE's of the form:

$$\text{(3.5a)} \qquad \frac{du_{i,j}}{dt} = \tilde{f}_{\ell,i,j}\left(t, \vec{u}_{i,j}, \vec{u}_{i+1,j}, \vec{u}_{i-1,j}, \vec{u}_{i,j+1}, \vec{u}_{i,j-1}\right) ,$$

$$i = 2,3,\ldots,NX - 1, \quad j = 2,3,\ldots,NY - 1 ,$$

$$\text{(3.5b)} \qquad \frac{du_{m,j}}{dt} = \tilde{f}_{\ell,m,j}\left(t, \vec{u}_{m,j}, \vec{u}_{mn,j}, \vec{u}_{m,j+1}, \vec{u}_{m,j-1}\right) ,$$

$$(m = 1,\ mn = 2) \text{ or } (m = NX,\ mn = NX - 1), \quad j = 2,3,\ldots,NY-1 ,$$

$$\text{(3.5c)} \qquad \frac{du_{i,m}}{dt} = \tilde{f}_{\ell,i,m}\left(t, \vec{u}_{i,m}, \vec{u}_{i,mn}, \vec{u}_{i+1,m}, \vec{u}_{i-1,m}\right) ,$$

$$(m = 1,\ mn = 2) \text{ or } (m = NY,\ mn = NY - 1), \quad i = 2,3,\ldots,NX-1 ,$$

$$\text{(3.5d)} \qquad \frac{du_{m,1}}{dt} = \tilde{f}_{\ell,m,1}\left(t, \vec{u}_{m,1}, \vec{u}_{mn,1}, \vec{u}_{m,2}\right) ,$$

$$(m = 1,\ mn = 2) \text{ or } (m = NX,\ mn = NX - 1)$$

$$\text{(3.5e)} \qquad \frac{du_{m,NY}}{dt} = \tilde{f}_{\ell,m,NY}\left(t, \vec{u}_{m,NY}, \vec{u}_{mn,NY}, \vec{u}_{m,NY-1}\right) ,$$

$$(m = 1,\ mn = 2) \text{ or } (m = NX,\ mn = NX - 1)$$

for $\ell = 1,2,\ldots,NPDE$, where $\vec{u}_{i,j} = (u_{1,i,j}, u_{2,i,j}, \ldots, u_{NPDE,i,j})$.

The usefulness of the method of lines approach can now be realized because the resulting system of ODE's can be solved using the powerful and advantageous capabilities already developed in ODE integrators. Present ODE integrators automatically select step size and up to twentieth order methods to maintain stability [1, 6]. They can also solve stiff nonlinear ODE's to a user specified accuracy [1, 2, 3]. Certainly to implement these capabilities would take a considerable amount of time and effort. We refer the reader to [8] for a detailed discussion of interfacing with the ODE integrator.

In summary, there are three basic steps in the solution of a system of PDE's. First the user defines his problem and sets up a spatial mesh. Then by using five point centered differencing at each mesh point, the software interface generates a system of ODE's. These approximations depend on whether the point is an interior mesh point, a boundary mesh point, or a corner mesh point. Then the final solution is determined as the ODE integrator solves the system of ODE's.

4. THE MATRIX PROBLEM

The differencing discussed in the previous section results in a system of (NODE = NPDE x NX x NY) ODE's. The solution of nonlinear problems often requires the use of the Jacobian matrix, $J = \partial\vec{f}/\partial\vec{u} = (\partial f_i/\partial u_j)$ $(i,j = 1,2,\ldots,NODE)$ with f_i defined in (3.5) and u_j denoting the current solution for one PDE at one mesh point. Clearly this matrix can become quite large (NODE,NODE) for even relatively few mesh lines. Also note that the generation of the matrix requires a large fraction of the function calls in the solution of the problem. Therefore the efficiency of any general software for two dimensional PDE's hinges on the efficient generation of the matrix. Efficiency is related to the number of mesh points since increasing the number of mesh points increases the time of each function evaluation. Consequently since accuracy often depends on the number of

mesh points, the number of function evaluations (NFE) necessary to generate the Jacobian should be independent of the number of mesh points.

Before we can efficiently generate the Jacobian we need to know its structure. Five point differencing results in the well defined banded structure shown in Figure 1. Because it assumes the matrix has a banded structure, GEARB is especially useful for our interface. We will be primarily referring to GEARB in the remainder of our discussion.

PSETM, the routine we present in this paper for the generation of the Jacobian is a modification of PSETB, the routine used in GEARB for generating the matrix. PSETB uses the approximating equations,

$$(4.1) \qquad \frac{\partial f_i}{\partial u_j} \approx (f_i(u_j + re_j) - f_i(u_j))d, \ (i,j = 1,2,\ldots,NODE), \ d \text{ a small number}$$

where r is a small increment and e_j denotes the standard unit vector, to approximate entries in the Jacobian matrix. For robustness PSETB evaluates the Jacobian as a fully banded matrix (i.e. $f_i = (u_{i-ML},\ldots,u_i,\ldots,u_{i+MU})$). From Figure 1 the upper (MU) and lower (ML) band widths can be figured to be ML = MU = NPDE x (NX + 1) - 1. Clearly the Jacobian evaluation can be improved by evaluating (4.1) for only the nonzero entries in the matrix (i.e. $f_i = (u_L, u_C, u_U)$ where u_L is the NPDE entries in the lower band, u_C is the 3 x NPDE entries in the center band, and u_U is the NPDE entries in the upper band of Figure 1). We can optimize the number of function evaluations for five point differencing by simultaneously evaluating in one function call, (4.1), for all those points of the mesh in the pattern indicated in Figure 2. This is possible because f_i depends on at most one of these points. It is easily verified that six function calls (NFE = 6), regardless of the size of the mesh, is all that is required to evaluate the Jacobian for each PDE. Consequently the total number of function evaluations is NPDE x 6. PSETM can be used in conjunction with GEARB in order to optimize the Jacobian generation for five point centered differencing methods.

Figure 1

The Structure of the Jacobian Matrix

The Jacobian is a block-tridiagonal matrix of size NY blocks by NY blocks. It has the structure:

$$J = \begin{pmatrix} C_1 & U_1 & & & & & \\ L_2 & C_2 & U_2 & & & & \\ & L_3 & C_3 & U_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & & L_{MY} & C_{MY} & U_{MY} \\ & & & & & & L_{NY} & C_{NY} \end{pmatrix}$$

where MY = NY - 1. The blocks C_j, L_j and U_j in the $j^{\underline{th}}$ row of the Jacobian contain all the entries generated for the approximating ODE's (3.5) at $x = x_i$ ($i = 1,2,\ldots,NX$) and $y = y_j$. Each block is a square matrix of size NX blocks by NX blocks where C_j is block-tridiagonal and L_j and U_j are block diagonal,

$$C_j = \begin{pmatrix} d_1 & e_1 & & & & & \\ g_2 & d_2 & e_2 & & & & \\ & g_3 & d_3 & e_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & & g_{MX} & d_{MX} & e_{MX} \\ & & & & & & g_{NX} & d_{NX} \end{pmatrix}$$

$$L_j = \begin{pmatrix} a_1 & & & & & \\ & a_2 & & & & \\ & & a_3 & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & a_{MX} \\ & & & & & & a_{NX} \end{pmatrix} \quad \text{and} \quad U_j = \begin{pmatrix} b_1 & & & & & \\ & b_2 & & & & \\ & & b_3 & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & b_{MX} \\ & & & & & & b_{NX} \end{pmatrix}$$

where MX = NX - 1.

(Figure 1 continued)

All blocks contained in C_j, L_j and U_j are full NPDE by NPDE matrices. Consider the $i^{\underline{th}}$ blocks of C_j, L_j and U_j. Let $\vec{f} \equiv (f_1, f_2, \ldots, f_{NODE})$ with f_ℓ defined by (3.5), $\vec{u} \equiv (u_1, u_2, \ldots, u_{NODE})$ and $(\partial\vec{f}/\partial\vec{u})_{ii,jj}$ denote the partial of $\vec{f}$ at $x = x_i$ and $y = y_j$ with respect to $\vec{u}$ at $x = x_{ii}$ and $y = y_{jj}$ ($ii = i, i\pm1$ and $jj = j, j\pm1$). The entries contained in the $i^{\underline{th}}$ blocks of C_j, L_j and U_j are generated from the approximating ODE, f_ℓ (3.5) at $x = x_i$ and $y = y_j$, where:

$$\left(\text{ the } g_i^{\underline{th}} \text{ block of } C_j\right) = \left(\frac{\partial\vec{f}}{\partial\vec{u}}\right)_{i-1,j},$$

$$\left(\text{ the } d_i^{\underline{th}} \text{ block of } C_j\right) = \left(\frac{\partial\vec{f}}{\partial\vec{u}}\right)_{i,j},$$

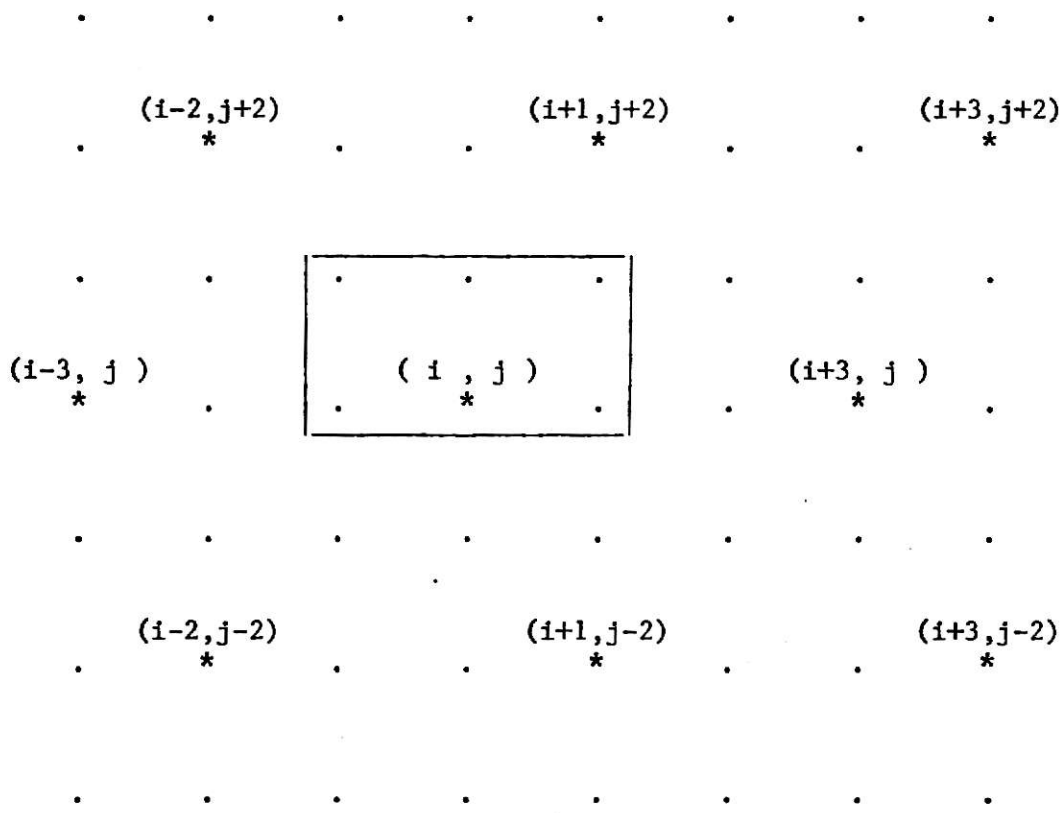
$$\left(\text{ the } e_i^{\underline{th}} \text{ block of } C_j\right) = \left(\frac{\partial\vec{f}}{\partial\vec{u}}\right)_{i+1,j},$$

$$\left(\text{ the } a_i^{\underline{th}} \text{ block of } L_j\right) = \left(\frac{\partial\vec{f}}{\partial\vec{u}}\right)_{i,j-1},$$

$$\left(\text{ the } b_i^{\underline{th}} \text{ block of } U_j\right) = \left(\frac{\partial\vec{f}}{\partial\vec{u}}\right)_{i,j+1}.$$

Figure 2

The Generation of the Jacobian



The * denotes the mesh points for which the partials (4.1) may be simultaneously evaluated. The rectangle denotes the six points requiring separate function calls to the interface.

We refer the reader to the listing of PSETM in Appendix B.

For a comparison between PSETM and PSETB, we consider the number of Jacobian entries determined by each of these routines in one function call. Since this determines the number of function calls necessary to generate the matrix, it is a useful comparison. With each function call PSETM determines $\frac{NX \times NY}{6}$ points in the Jacobian while PSETB evaluates at best $\frac{NX \times NY \times NPDE - 1}{2((NX + 1) \times NPDE) - 1) + 1}$. As it turns out, the number of points determined by PSETB is only slightly dependent on NPDE. We therefore now make some specific comparisons between PSETB and PSETM independent of NPDE. For a 3x3 spatial mesh, PSETB needs about 30% more function calls, for a 10x10 mesh it needs about 3.6 times more function calls, and for a 100x100 mesh, PSETB turns out to need over 33 times more function calls than PSETM. Clearly this is important since as the number of mesh points increases so does the time for each function evaluation. Since the time requirements are very dependent on the number of function evaluations, PSETM will significantly improve the efficiency of the generation of the Jacobian.

5. USER DEFINITION OF THE PROBLEM

This section presents the minimal effort required by the user to make use of the software interface. The problem is completely defined by a small main program and five subroutines, BNDRYH, BNDRYV, DV, DH and F. The main program defines the mesh, sets the initial conditions, provides the interface to the integrator and prints or plots all the output. The routines BNDRYH and BNDRYV define the horizontal and vertical boundaries respectively, DH and DV define the horizontal and vertical diffusion coefficients respectively, and F defines the PDE.

For the routine constructions that follow, T is the current time, X and Y define the position in the spatial mesh for the horizontal and vertical direc-

tions, and U is the current solution of the PDE's at the point defined by X and Y. We will show a detailed description of the construction of each of these routines. In each case we include the appropriate dimension statements and parameter lists.

a. The main program is constructed as follows:

```
COMMON /MESH1/ XMESH ( * )
COMMON /MESH2/ YMESH ( ** )
COMMON /MESH3/ NPDE, NX, NY
DIMENSION U0 ( *** , * , ** )
```

For the dimensions above enter the actual numerical values for * = NX, ** = NY and *** = NPDE. Define in this routine the mesh spacing (i.e. XMESH(1), XMESH(2),..., XMESH(NX) and YMESH(1), YMESH(2),..., YMESH(NY), NX $\geq$ 3, and NY $\geq$ 3) and NPDE. The initial conditions (2.4) should be defined in U0 such that $U0(\ell,i,j) = \phi_\ell(x_i,y_j)$ for (ℓ = 1,2,...,NPDE), (i = 1,2,...,NX) and (j = 1,2,...,NY). In order to interface with an integrator one usually also specifies the desired output times, the desired accuracy, the initial step size and the type of integration desired. The integrator will return the solutions to this routine.

```
STOP
END
```

b. The constructions of the boundary subroutines, BNDRYH and BNDRYV are analogous. The construction of BNDRYH is the following:

```
SUBROUTINE BNDRYH (T,X,Y,U,AH,BH,CH,NPDE)
```

```
DIMENSION U(NPDE), AH(NPDE), BH(NPDE), CH(NPDE)
```

The values for the coefficients of (2.2) are defined in this routine. X is any point of XMESH(K), K = 1,2,..., NX and Y is either YMESH(1) or YMESH(NY). Define then AH(K), BH(K), and CH(K) (K = 1,2,...,NPDE) for the lower (Y = YMESH(1)) and upper (Y = YMESH(NY)) boundaries.

```
RETURN
END
```

c. The subroutines DH and DV defining the diffusion coefficients are also analogous. We construct DH as:

```
SUBROUTINE DH (T,X,Y,U,DVALH,NPDE)
DIMENSION U(NPDE),DVALH(NPDE,NPDE)
```

Define in this routine DVALH(L,K) (L,K = 1,2,...,NPDE), the $DH_{\ell,k}$ coefficients of (2.1). X and Y denote either a boundary point or an averaged value between two mesh points (3.2f). All matrix entries of DVALH mush be defined even if they are zero.

```
RETURN
END
```

d. The subroutine F is constructed as follows:

```
SUBROUTINE F (T,X,Y,U,UX,UY,DUXX,DUYY,UDOT,NPDE)
DIMENSION U(NPDE), UX(NPDE), UY(NPDE), DUXX(NPDE,NPDE), DUYY(NPDE,NPDE),
          UDOT(NPDE)
```

In this routine, the incoming values X and Y represent the mesh point

being evaluated, UX = $\frac{\partial \vec{u}}{\partial x}$, UY = $\frac{\partial \vec{u}}{\partial y}$, DUXX = $\frac{\partial}{\partial x}$ ($DH_{\ell,k}$ $\frac{\partial \vec{u}}{\partial x}$) and DUYY = $\frac{\partial}{\partial y}$ ($DV_{\ell,k}$ $\frac{\partial \vec{u}}{\partial y}$) in approximating the arguments of f_ℓ in (2.1). The PDE's (f_ℓ, ℓ = 1,2,..., NPDE) are defined in terms of the incoming parameters in UDOT(L) (L = 1,2,..., NPDE).

```
RETURN
END
```

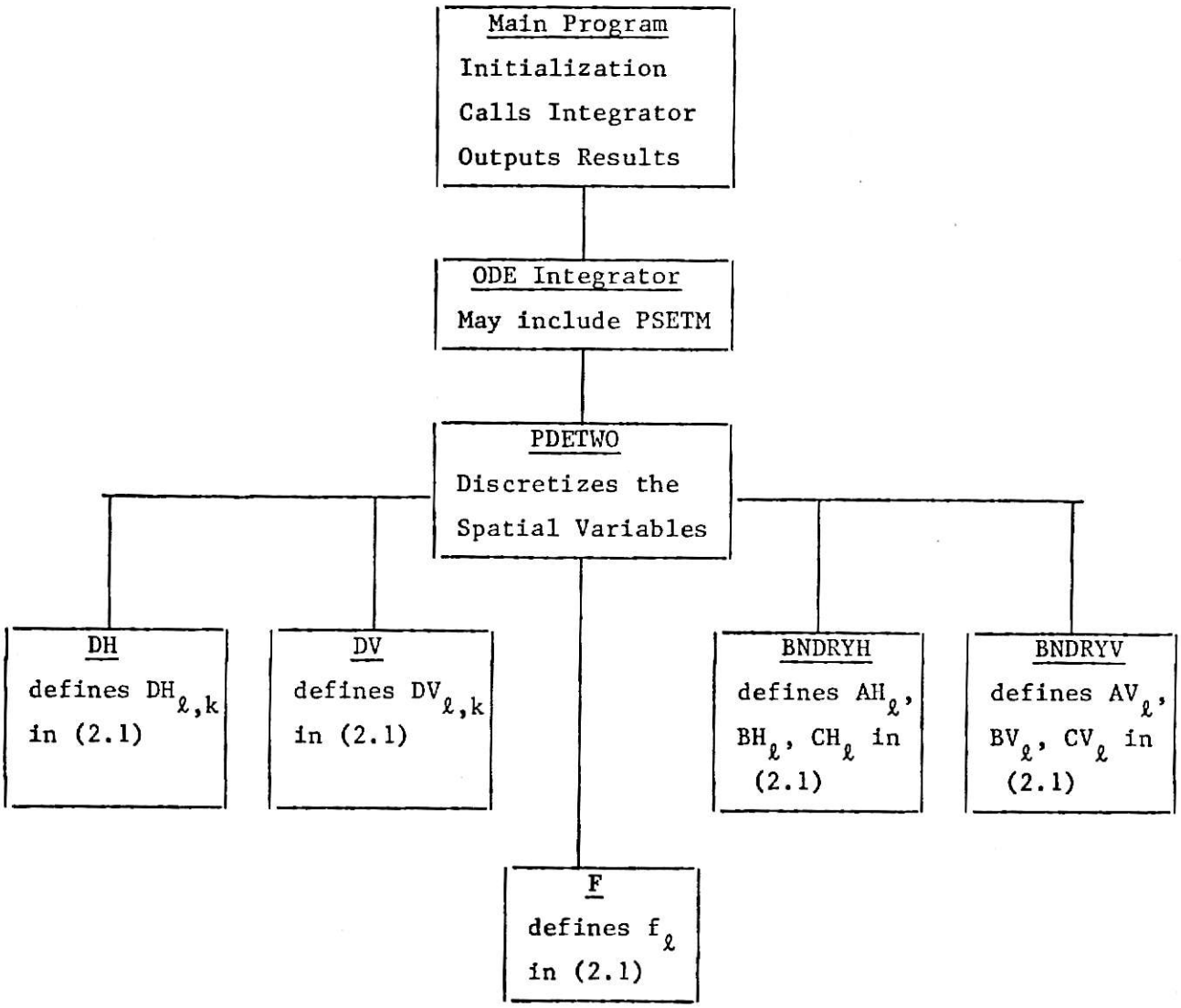
We note that there is a modification necessary in the main routine when the problem defines a constant, time dependent boundary condition. The integrator generally does not determine the solution at the time specified by the user and so it interpolates the solution back to the time specified. Therefore to obtain the solution at the constant, time dependent boundaries, the user must call the appropriate boundary routines from the main routine. An example of this case is presented in the next section along with a complete listing of the user defined routines.

Note that these six routines, BNDRYH, BNDRYV, DV, DH, F and the main program completely define the problem. We can now visualize the relationship of all the routines as shown in Figure 3. The main program initializes the solution process, passing the required parameters to the integrator and passing the necessary information to PDETWO through common blocks. The integrator consists of several routines. If PSETM is used it replaces one of these routines. The integrator then calls PDETWO which evaluates the right hand side of the semidiscrete approximating ODE system (3.5). The integrator makes calls to PDETWO for each time step and in order to evaluate the Jacobian matrix (if required). PDETWO in turn calls BNDRYH, BNDRYV, DV, DH and F in approximating (3.5). After the integrator has determined the solution for the user specified time and accuracy, it returns the solution to the main program to be printed or plotted.

For further clarification of the use of PDETWO, we provide in the next section the complete listing of the user defined routines for one of the examples presented there.

Figure 3

The Relationship of the Routines



6. NUMERICAL EXAMPLES

It has proven to be a very difficult task to find good test problems for our software interface. Because of the infant state of general purpose software for two dimensional PDE's, few good test problems are available. For our situation a good test problem would be a moderately difficult PDE or system of PDE's for which the actual solution is known. In this section we present examples which we feel will at least demonstrate the potential usefulness of our software interface.

The first example we present is an Elliptic PDE which represents a parallel plate heated by a nearly oblong object. The PDE is:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - 6xye^x e^y(xy + x + y - 3)$$

with boundary conditions

$$u(t,x,y) = 0$$

and initial conditions

$$\phi(0,x,y) = 0 .$$

For this problem the actual solution is $3e^x e^y(x - x^2)(y - y^2)$. By integrating to a steady state solution we obtain the solution to the original problem since the problem's solution is independent of time. We chose the region to be $0 \leq x \leq 1$ and $0 \leq y \leq 1$ with a uniformly spaced 5 x 5 spatial mesh. The initial time step is 10^{-7} and the error tolerance is 10^{-6}. Using a modified Newton's method in the ODE integrator, the results presented in the following table were obtained. TIME is the specified output time. NSTEPS is the total number of steps taken by the integrator to reach the specified time. NFE is the

total number of steps taken by the integrator to reach the specified time. NFE is the total number of calls to PDETWO required and NJE is the number of Jacobian evaluations. ERROR indicates the maximum error over the region.

TIME	NSTEPS	NFE	NJE	ERROR
.001	7	36	2	.294
.01	19	94	5	.439
.1	50	165	7	.099
1.0	108	301	13	.016
10.0	129	377	17	.016

As seen from the ERROR column, the solution has reached a steady state by a time of 1.0 . The observed error is larger than the error tolerance because the error tolerance affects only the error in the time integration. Therefore the observed error is due to the spatial discretization. This error could be reduced by choosing a finer mesh. This is demonstrated by using a uniformly spaced 10x10 mesh in place of the 5x5 mesh. The results from this change are as follows:

TIME	NSTEPS	NFE	NJE	ERROR
.001	9	82	3	.548
.01	28	161	5	.468
.1	70	276	8	.081
1.0	130	474	14	.0034
10.0	143	560	17	.0034

Note that the error is reduced by approximately 1/4 indicating that the error is resulting from the spatial discretization.

We have also solved a Hyperbolic PDE with our interface. The following PDE represents the flow of a chemical reaction through a parallel plate. The equation is:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - 2500\ x\ (1 - x)\ \frac{\partial u}{\partial y} + 100\ u^2$$

with boundary conditions

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial y} = 0$$

and initial conditions

$$\phi\ (0,x,y) = 0.0 \qquad \text{when } x \neq 0$$

$$\phi\ (0,0,y) = 1.0\ .$$

This PDE represents the flow of the reaction in the y direction. It is derived assuming (i) no diffusion in the y direction, (ii) the reactor length is infinitely long, (iii) the coefficients of $\frac{\partial^2 u}{\partial x^2}$ and u^2 are constant and (iv) the coefficient of $\frac{\partial u}{\partial y}$ is a function of x alone.

We used the region $0 \leq x \leq 1$ and $0 \leq y \leq 1$ to obtain our solution. We chose a spatial mesh with 10 uniformly space intervals in the x-direction (NX=11) and 30 uniformly spaced intervals in the y-direction (NY=31). The error tolerance was 10^{-4} and the initial step size was 10^{-6}. We solved this problem using a modified Newton's method and using functional iteration in the ODE integrator. Our results are as follows:

ITERATION

TIME	NSTEPS	NFE
.1E-6	1	2
.1E-5	4	5
.1E-4	5	6
.1E-3	10	14

NEWTON'S

TIME	NSTEPS	NFE	NJE
.1E-6	1	9	1
.1E-5	4	20	2
.1E-4	5	22	2
.1E-3	11	41	3

The transient nature of the solution causes the error to be determined by the time step. It also causes the Jacobian to be outdated quickly. Consequently

Newton's method proved to be less efficient than functional iteration. We note that the output times are small but they are relative to the speed of the reaction. For this equation, the wave had moved approximately 1/4 of the way across the mesh by the final output time.

Our final example is a problem that has been contrived to demonstrate the use and some of the capabilities of our software. Since we know the actual solution, the capabilities of our software interface can be more easily understood. The problem is:

$$\frac{\partial u_1}{\partial t} = \frac{\partial}{\partial x}\left(u_2 \frac{\partial u_1}{\partial x}\right) + \frac{\partial}{\partial y}\left(u_2 \frac{\partial u_1}{\partial y}\right) + \frac{\partial}{\partial x}\left(u_1 \frac{\partial u_2}{\partial x}\right) - 4e^{-t}y\frac{\partial u_2}{\partial x} - y\frac{\partial u_1}{\partial y}$$

$$\frac{\partial u_2}{\partial t} = \frac{\partial}{\partial x}\left(u_1^2 \frac{\partial u_1}{\partial x}\right) + \frac{\partial}{\partial y}\left(u_2 \frac{\partial u_2}{\partial y}\right) - 2xy^2e^{-2t}\frac{\partial u_1}{\partial x} - 2e^{-t}(u_2 + 2e^{-t}y^2) - u_2$$

with vertical boundary conditions

$$u_1(t,0,y) = 0 \ , \qquad \frac{\partial u_2}{\partial x} = 2xe^{-t} \quad \text{at } x = 0 \ ,$$

$$u_1 + \frac{\partial u_1}{\partial x} = 2e^{-t}y \ , \qquad u_2 = e^{-t}(x^2 + y^2) \quad \text{at } x = 1 \ ,$$

horizontal boundary conditions

$$\frac{\partial u_1}{\partial y} = e^{-t}x \ , \qquad u_2 + \frac{\partial u_2}{\partial y} = e^{-t}x^2 \quad \text{at } y = 0 \ ,$$

$$u_1 = e^{-t}xy \ , \qquad \frac{\partial u_2}{\partial y} = 2ye^{-t} \quad \text{at } y = 1 \ ,$$

and initial conditions

$$u_1(0,x,y) = xy \ , \qquad u_2(0,x,y) = x^2 + y^2 \ .$$

The actual solution is:

$$u_1 = e^{-t}xy \ , \qquad u_2 = e^{-t}(x^2 + y^2) \ .$$

Note that this problem is a coupled system of nonlinear PDE's having Neuman, Dirichlet and mixed boundary conditions. We have solved this problem using a modified Newton's method with a uniform 5x5 spatial mesh over the region $0 \leq x \leq 1$ and $0 \leq y \leq 1$ with an initial step size of 10^{-7} and an error tolerance for the time integration of 10^{-7}. We obtained the following results where NFE(PSETB) indicates the number of function evaluations if PSETB in GEARB is used and NFE(PSETM) indicates the number of function evaluations if PSETM is used.

TIME	NSTEPS	NFE(PSETB)	NFE(PSETM)	NJE	ERROR U(1)	ERROR U(2)
.1E-4	4	55	35	2	.749E-5	.134E-4
.1E-3	8	84	53	3	.748E-4	.134E-3
.1E-2	15	142	89	5	.734E-3	.132E-2
.1E-1	37	316	197	11	.608E-2	.118E-1
.1E 0	142	1056	662	36	.202E-1	.555E-1

In this table, the ERROR columns indicate the maximum error found in the interior of the mesh. Since the solution of u_1 and u_2 increases with x and y, this maximum error was found at the mesh point (4,4). Clearly, since the initial conditions are defined consistent with the original problem, the error for small time (up to .1E-2) is dependent on the time truncation error. With a large enough time (.1E-1) the spatial discretization error can be observed. We did not pursue this since we observed this error in the first example.

We also provide a complete listing of the user defined routines required to interface with our software interface. It is as follows:

```
C
C          THIS IS A LISTING OF ALL THE USER DEFINED ROUTINES NECESSARY
C   TO SOLVE THE THIRD EXAMPLE.
C
C          COMMON BLOCK GEAR9 PROVIDES INFORMATION FROM THE INTEGRATOR.
C
C   HUSED    IS THE SIZE OF TIME STEP USED IN THE INTEGRATOR.
C   NSTEP    IS THE NUMBER OF STEPS USED BY THE INTEGRATOR TO OBTAIN THE
C            SOLUTION AT THE SPECIFIED OUTPUT TIME.
C   NFE      IS THE TOTAL NUMBER OF FUNCTION EVALUATIONS REQUIRED.
C   NJE      IS THE TOTAL NUMBER OF JACOBIAN EVALUATIONS REQUIRED.
C
C          THE VARIABLES USED ARE AS FOLLOWS..
C
C   NX       IS THE NUMBER OF HORIZONTAL MESH POINTS.
C   NY       IS THE NUMBER OF VERTICAL MESH POINTS.
C   MF       DETERMINES THE TIME INTEGRATION METHOD.
C   NPDE     IS THE NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C   INDEX=1  INDICATES THIS IS THE FIRST CALL TO THE INTEGRATOR.
C   N        IS THE TOTAL NUMBER OF ODE*S GENERATED (N=NX*NY*NPDE).
C   TO       IS THE INITIAL TIME.
C   H        IS THE INITIAL TIME STEP.
C   EPS      IS THE ERROR TOLERANCE.
C   U0       IS THE CURRENT SOLUTION AT THE MESH POINTS.
C   AU1,AU2  ARE THE ACTUAL VALUES OF THE SOLUTION AT THE MESH POINTS.
C   E1,E2    ARE THE ERRORS BETWEEN THE CALCULATED VALUE AND THE ACTUAL
C            VALUE AT THE MESH POINTS.
C   XMESH    CONTAINS THE MESH SPACING IN THE HORIZONTAL DIRECTION.
C   YMESH    CONTAINS THE MESH SPACING IN THE VERTICAL DIRECTION.
C
      DIMENSION U0(2,5,5),AU1(5,5),AU2(5,5),E1(5,5),E2(5,5)
      DIMENSION ALPHAH(2),BETAH(2),GAMMAH(2),ALPHAV(2),BETAV(2),
     *          GAMMAV(2)
      COMMON /MESH1/ XMESH(5)
      COMMON /MESH2/ YMESH(5)
      COMMON /MESH3/ NPDE,NX,NY
      COMMON /GEAR9/ HUSED,NQUSED,NSTEP,NFE,NJE
      READ (5,10) NX,NY,MF,NPDE,INDEX
   10 FORMAT (5I3)
C
C  DEFINE THE MESH SPACING.
C
      DX = 1.D+00/(NX-1.D+00)
      DO 20 K=1,NX
        XMESH(K) = (K-1)*DX
        YMESH(K) = XMESH(K)
   20   CONTINUE
C
C  DEFINE THE UPPER AND LOWER BAND WIDTHS FOR THE JACOBIAN.
C
      MU=NPDE*(NX+1)-1
      ML=MU
      READ (5,30) N,TO,H,EPS
   30 FORMAT ( I3,F7.2,E7.1,E7.1)
C
C  DEFINE THE INITIAL CONDITIONS.
C
      DO 40 J2=1,NY
        DO 40 J3=1,NX
          U0(1,J3,J2) = XMESH(J3)*YMESH(J2)
```

```
            UO(2,J3,J2) = XMESH(J3)*XMESH(J3) + YMESH(J2)*YMESH(J2)
   40       CONTINUE
            WRITE (6,50) N,TO,H,EPS,MF,UC
   50 FORMAT (3H N ,I2,4H TO ,F9.2,3H H ,E8.1,5H EPS ,E8.1,4H MF ,
           * I2,4H UC ,(/1H ,10E11.2))
C
C  SPECIFY THE OUTPUT TIME.
C
   55 READ (5,60,END=180) TOUT
   60 FORMAT (F11.7)
            WRITE (6,70) TOUT
   70 FORMAT (6H TOUT ,E15.6)
C
C  CALL THE INTEGRATOR.
C
            CALL DRIVEB (N,TO,H,UO,TOUT,EPS,MF,INDEX,ML,MU)
            WRITE (6,80) HUSED,NQUSED,NSTEP,NFE,NJE
   80 FORMAT (7H HUSED ,E10.4,7H ORDER ,I3,7H NSTEP ,I6,5H NFE ,I5,
           * 5H NJE ,I5)
C
C  DEFINE THE CORRECT SOLUTIONS FOR CONSTANT TIME DEPENDENT BOUNDARIES.
C
            DO 82 J1=2,NX
              CALL BNDRYH(T,X(J1),Y(NY),U(1,J1,NY),ALPHAH,BETAH,GAMMAH,NPDE)
              CALL BNDRYV(T,X(NX),Y(J1),U(1,NX,J1),ALPHAV, BETAV,GAMMAV,NPDE)
              DO 82 K1=1,NPDE
                IF (BETAV(K1) .EQ. 0.0) U(K1,NX,J1)=GAMMAV(K1)/ALPHAV(K1)
                IF (BETAH(K1) .EQ. 0.0) U(K1,J1,NY)=GAMMAH(K1)/ALPHAH(K1)
   82       CONTINUE
C
C  PRINT THE RESULTS.
C
            WRITE (6,90)
   90 FORMAT (/9H U VALUES)
            WRITE (6,100) (((UO(IX,IY,IZ),IY=1,NX),IZ=1,NY),IX=1,2)
  100 FORMAT ( 1H,10E13.6)
C
C  DETERMINE THE ACTUAL SOLUTION.
C
            DO 110 K2=1,NY
              DO 110 K1=1,NX
                DET = DEXP(-TOUT)
                AU1(K1,K2) = DET*XMESH(K1)*YMESH(K2)
                AU2(K1,K2) = DET*(XMESH(K1)*XMESH(K1)+YMESH(K2)*YMESH(K2))
  110       CONTINUE
            WRITE (6,120)
  120 FORMAT (23H ACTUAL SOLUTION FOR U1 )
            WRITE (6,100) AU1
            WRITE (6,130)
  130 FORMAT (23H ACTUAL SOLUTION FOR U2 )
            WRITE (6,100) AU2
C
C  DETERMINE THE ERROR.
C
            DO 140 K2=1,NY
              DO 140 K1=1,NX
                E1(K1,K2)=UC(1,K1,K2)-AU1(K1,K2)
                E2(K1,K2)=UO(2,K1,K2)-AU2(K1,K2)
  140       CONTINUE
            WRITE (6,150)
```

```
  150 FORMAT (/13H ERROR FOR U1 )
      WRITE (6,100) E1
      WRITE (6,160)
  160 FORMAT (/13H ERROR FOR U2 )
      WRITE (6,100) E2
  170 CONTINUE
      GO TO 55
  180 STOP
      END


      SUBROUTINE BNDRYV (T,X,Y,U,AV,BV,CV,NPDE)
C
C   DEFINE THE VERTICAL BOUNDARIES.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION U(NPDE),AV(NPDE),BV(NPDE),CV(NPDE)
      IF (X .GT. .5) GOTO 10
      AV(1) = 1.D+00
      BV(1) = 0.0
      CV(1) = 0.0
      AV(2) = 0.0
      BV(2) = 1.D+00
      CV(2) = 0.0
      RETURN
   10 CONTINUE
      AV(1) = 1.D+00
      BV(1) = 1.0D+00
      CV(1) = DEXP(-T)*Y*2.D+00
      AV(2) = 1.D+00
      BV(2) = 0.0
      CV(2) = DEXP(-T)*(1.D+00 + Y*Y)
      RETURN
      END


      SUBROUTINE BNDRYH (T,X,Y,U,AH,BH,CH,NPDE)
C
C   DEFINE THE HORIZONTAL BOUNDARIES.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION U(NPDE),AH(NPDE),BH(NPDE),CH(NPDE)
      IF ( Y .GT. .5) GO TO 10
      AH(1) = 0.0
      BH(1) = 1.D+00
      CH(1) = DEXP(-T)*X
      AH(2) = 1.D+00
      BH(2) = 1.D+00
      CH(2) = DEXP(-T)*X*X
      RETURN
   10 CONTINUE
      AH(1) = 1.D+00
      BH(1) = 0.0
      CH(1) = DEXP(-T)*X*Y
      AH(2) = 0.0
      BH(2) = 1.D+00
      CH(2) = 2.D+00*Y*DEXP(-T)
      RETURN
      END
```

```
      SUBROUTINE DH (T,X,Y,U,DVAL,NPDE)
C
C   DEFINE THE HORIZONTAL DIFFUSION COEFFICIENTS.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION U(NPDE),DVAL(NPDE,NPDE)
      DVAL(1,1) = U(2)
      DVAL(1,2) = U(1)
      DVAL(2,1) = 0.0
      DVAL(2,2) = U(1)*U(1)
      RETURN
      END


      SUBROUTINE DV (T,X,Y,U,DVAL,NPDE)
C
C   DEFINE THE VERTICAL DIFFUSION COEFFICIENTS.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION U(NPDE),DVAL(NPDE,NPDE)
      DVAL(1,1) = U(2)
      DVAL(1,2) = 0.0
      DVAL(2,1) = 0.0
      DVAL(2,2) = U(2)
      RETURN
      END


      SUBROUTINE F(T,X,Y,U,UX,UY,DUXX,DUYY,UDOT,NPDE)
      IMPLICIT REAL*8 (A-H,O-Z)
C
C   DEFINE THE PDE*S.
C
      DIMENSION U(NPDE),UX(NPDE),UY(NPDE),DUXX(NPDE,NPDE),
     * DUYY(NPDE,NPDE),UDOT(NPDE)
      DET = DEXP(-T)
      UDOT(1) = DUXX(1,1)+DUYY(1,1)+DUXX(1,2)-4.D+00*DET*Y*UX(2)-Y*UY(1)
      UDOT(2) = DUXX(2,2)+DUYY(2,2)-2.*X*Y*Y*DET*DET*UX(1)-2.*DET*(U(2)
     *            +2.*DET*Y*Y)-U(2)
      RETURN
      END
```

7. COMMENTS AND OBSERVATIONS

The numerical method of lines has been used for some time now and has proven to be powerful, versatile and easy to use. The successes obtained from the implementation of the method of lines developed by Sincovec and Madsen [8], certainly increases our confidence in this approach. In light of their success, we have developed and presented an extension of their software to solve PDE's with two independent spatial variables as well as an independent variable in time.

The infant state of general purpose software for two dimensional PDE's has made testing of our interface very difficult. The wide class of PDE's which the interface is capable of solving, also makes it difficult to completely test. Since few good test problems for two dimensional PDE's are available, we have solved a limited number of PDE's using our software interface. However these PDE's encompass a representative set of the possible problems allowed by our interface. Also since our interface represents an extension of proven methods, we have confidence in its validity.

The numerical solution of two dimensional PDE's depends heavily on the efficiency of the methods used to solve them. We have already made comments concerning the Jacobian generation in Section 4. We will now determine in some meaningful sense, the space and time requirements of the solution process. Since no other general software exists no comparison of these requirements to other methods can be made.

The time requirements are largely determined by the number of function evaluations (i.e. the number of times the interface, PDETWO is called). Each time the approximating ODE's for a spatial mesh point (all PDE's) are generated by PDETWO, it requires one evaluation of DH, one evaluation of DV and one evaluation of F. Of course for boundary points, it also requires eval-

uations of BNDRYH or BNDRYV or both. To approximate UX, UY, DUXX and DUYY, the interface makes approximately (5 + 6 x NPDE) multiplications in addition to the above mentioned evaluations. Time requirements will then be primarily based on NPDE, the size of the mesh and the complexity of the diffusion coefficients. Clearly as NPDE gets large the multiplications in the interface will take a large fraction of the time. Recall that the efficient generation of the Jacobian in PSETM requires six function calls to PDETWO per PDE (NFE = 6 x NPDE). These calls increase considerably the time requirements in obtaining a solution to a problem. To alleviate the time constraints, the user may choose an alternative. One alternative is for the user to explicitly define the matrix in another subroutine. This approach may reduce the time requirements. However it is not always easy to define the matrix and unfortunately this approach requires the user to completely understand our interface. The other approach is to use functional iteration and avoid the matrix problem completely. However functional iteration methods often require very small time steps in order to maintain stability and accuracy. This makes them inadequate for many problems.

The storage requirement is largely determined by the storage for the interface and the storage for the integrator. The space requirement of the interface is primarily for three arrays, DVALS(NPDE,NPDE,NX,2), U(NODE) and UDOT(NODE) (Recall that NODE = NPDE x NX x NY). This is small in comparison with the requirements of the integrator. GEARB, for instance, requires six arrays of dimension NODE, Y(NODE,M) (M = 2,3,...,13, depending on the order of time integration method used), plus the Jacobian which requires NODE x (3 x ML + 1) locations (ML = (NX + 1)NPDE - 1). The Jacobian storage is somewhat deceptive since ML x NODE locations are required for pivoting. Of the remaining (2 x ML + 1) x NODE locations at most 5(NPDE x NODE) - 2(NX x NPDE)

of the Jacobian entries are nonzero. The remaining locations fill in during the factorization of the Jacobian matrix by Gaussian elimination. The user should note that the width of the band in the Jacobian is dependent on NX and independent of NY. Clearly then to conserve storage, the mesh should be defined and the problem oriented so that NX $\leq$ NY.

To make the storage requirements clearer, we present two examples. For a 10x10 spatial mesh with one PDE the interface requires about 220 locations. Assuming Y(NODE,6), the integrator uses about 4300 locations, 3100 of which are for the Jacobian. One thousand of these are reserved for pivoting and at most 480 of the Jacobian entries are nonzero. For a 10x10 mesh with 5 NPDE the interface requires approximately 1500 locations and the integrator approximately 87,500, of which 81,500 are for the Jacobian. Of the 81,500 for the Jacobian, 27,000 are required for pivoting but no more than 12,400 are actually nonzero entries. Clearly, the direct solution of the Jacobian is the primary factor in the space requirements for the solution of two dimensional PDE's.

Since the direct solution of the Jacobian may have large storage requirements, the user could consider the alternative of using iterative methods to solve the matrix problem [16]. The advantage of these methods is that they require only the nonzero entries of the Jacobian in solving the matrix. However these methods require additional knowledge about the iteration parameters (i.e. convergence tests, when to update, etc.). The user could also use functional iteration and completely eliminate the storage for the Jacobian. However as pointed out before, functional iteration may require small time steps to maintain stability. It is also worth pointing out that if there is an iterative method for solving the matrix, which is particularly well suited to the problem being solved, the user may also use that method. With that approach, the routines in the integrator which solve the matrix by Gaussian elimination would be replaced by the routines for the iterative method.

In considering the time and space tradeoffs for the solution of a system of PDE's, it has been our experience that time tends to be a greater restriction on the solution than space. As noted previously, the time requirement stems directly from the solution of the Jacobian matrix. With the development of much faster methods for solving the matrix problem, the combination of our interface, the integrator and the faster methods for solving the matrix would be a very robust package. The interface will do all the work of discretizing the spatial variables to form approximating ODE's. Then the integrator with the help of a fast method for solving the matrix problem could determine the solution for a large class of PDE's. So, even in view of its limitations, we feel that the software interface presented in this paper is a significant first step in the development of robust general software for two dimensional PDE's.

REFERENCES

1. Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, N. J. 1971.

2. Hindmarsh, A. C., "GEARB: Solution of Ordinary Differential Equations Having Banded Jacobian", UCID-30059, Lawrence Livermore Laboratory, May 1973.

3. Hindmarsh, A. C., "GEAR: Ordinary Differential Equation System Solver", UCID-30001 Rev. 2, Lawrence Livermore Laboratory, August 1972.

4. Hutula, D. N. and B. E. Wiancko, "MATUS: A Three-Dimensional Finite Element Program for Small-Strain Elastic Analysis", WAPD-TM-1081, Bettis Atomic Power Lab, March 1973.

5. Krogh, F. T., "An Integrator Design", Jet Propulsion Laboratory Tech. Memo. 278, Califormia Institute of Technology, Pasadena, CA, 1971.

6. Krogh, F. T., "Algorithms for Changing the Step Size", SIAM Journal on Numerical Analysis, Vol. 10, No. 5, October 1973, pp. 949-965.

7. Madsen, N. K. and R. F. Sincovec, "The Numerical Method of Lines for the Solution of Nonlinear Partial Differential Equations", UCRL-75142, Lawrence Livermore Laboratory, September 1973; also to appear in S.I.A.M. Journal on Numerical Analysis.

8. Madsen, N. K. and R. F. Sincovec, "Software for Nonlinear Partial Differential Equations", ACM Transactions on Mathematical Software, Vol. 1, No. 3, September 1975, pp. 232-260.

9. Maechen, G. and S. Sack, "The TENSOR Code", in Methods in Computational Physics, Vol. 3, B. Alder, S. Fernbach, and M. Rotenberg, eds., Academic Press, New York, 1964.

10. Marcal, P. V., "On General Purpose Programs for Finite Element Analysis", Numerical Solution of Partial Differential Equations-II, SYNSPADE 1970, B. Hubbard, Editor, Academic Press, New York, 1971.

11. Shampine, L. F. and M. K. Gordon, Computer Solution of Ordinary Differential Equations: Initial Value Problems, to be published by W. H. Freeman and Co., San Francisco, CA.

12. Snyder, L. J., "Two-Phase Reservoir Flow Calculations", Society of Petroleum Engineers Journal, June 1969, pp. 170-182.

13. Varga, R. S., Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, N. J., 1962.

14. Wilkins, M. L., "Calculation of Elastic-Plastic Flow", Lawrence Livermore Laboratory, Report UCRL-7322, Rev. 1, 1969.

15. Yasinsky, J. B., M. Natelson, and L. A. Hageman, "TWIGL - A Program to Solve the Two-Dimensional, Two-Group, Space-Time Neutron Diffusion Equations with Temperature Feedback", WAPD-TM-743, Bettis Atomic Power Lab, 1968.

16. Young, D., Iterative Solution of Large Linear Systems, Academic Press, New York and London, 1971.

```
C
C
C                              APPENDIX A
C
C
      SUBROUTINE PDETWO (N,T,U,UDOT)
      IMPLICIT REAL*8 (A-H,O-Z)
C
C       PDETWO IS AN INTERFACE DESIGNED TO BE USED IN CONJUNCTION WITH AN
C   ODE INTEGRATOR (SUCH AS GEARB) TO SOLVE A SYSTEM OF TWO-DIMENSIONAL
C   PDE*S.  IT IS AN EXTENSION OF PDEONE, AN INTERFACE DEVELOPED BY
C   SINCOVEC AND MADSEN USED FOR THE SOLUTION OF ONE-DIMENSIONAL
C   SYSTEMS OF PDE*S.
C
C     PDETWO USES FIVE POINT CENTERED DIFFERENCING TO CONVERT A TWO-
C   DIMENSIONAL SYSTEM OF PDE*S TO A SYSTEM OF ODE*S OF THE FORM,
C   UDOT = F(T,X,Y,U).
C      PDETWO IS CALLED BY THE INTEGRATOR (STIFFB IN GEARB) TO GENERATE
C   THE SYSTEM OF ODE*S TO BE SOLVED.  IF A JACOBIAN MATRIX IS REQUIRED
C   PDETWO IS ALSO CALLED IN ORDER TO GENERATE THIS MATRIX (PSETB IN
C   GEARB OR PSETM THE MODIFIED VERSION OF PSETB MAKE THESE CALLS).
C
C                          REFERENCES
C
C       1.  A. C. HINDMARSH, GEAR.. ORDINARY DIFFERENTIAL EQUATION
C             SYSTEM SOLVER, UCID-30001 REV. 3, LAWRENCE LIVERMORE
C             LABORATORY, P.O.BOX 808, LIVERMORE, CA 94550, DEC. 1974.
C
C       2.  A. C. HINDMARSH, GEARB.. SOLUTION OF ORDINARY
C             DIFFERENTIAL EQUATIONS HAVING BANDED JACOBIAN,
C             UCID-30059 REV. 1, L.L.L., MARCH 1975.
C
C       3.  R.F. SINCOVEC AND N.K. MADSEN, SOFTWARE FOR NONLINEAR
C             PARTIAL DIFFERENTIAL EQUATIONS, ACM TRANSACTIONS ON MATH-
C             EMATICAL SOFTWARE, SEPTEMBER 1975, PP. 232-269.
C
C       4.  R.F. SINCOVEC AND N.K. MADSEN, ALGORITHM 494 PDEONE,
C             SOLUTIONS OF SYSTEMS OF PARTIAL DIFFERENTIAL EQUATIONS,
C             ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, SEPTEMBER 1975,
C             PP. 262-263.
C
C  THE INPUT PARAMETERS ..
C
C  N                THE NUMBER OF ODE*S GENERATED BY PDETWO AND PASSED
C                  TO THE INTEGRATOR.  N IS EQUAL TO NX*NY*NPDE (SEE THE
C                  VARIABLE EXPLANATIONS BELOW).
C  T                THE CURRENT TIME.
C  U                AN ARRAY OF SIZE NPDE BY NX BY NY CONTAINING THE
C                  CURRENT SOLUTION VALUES FOR ALL THE MESH POINTS.
C
C  THE OUTPUT PARAMETER IS..
C
C  UDOT             THE SYSTEM OF NPDE BY NX BY NY ODE'S PASSED TO THE
C                  INTEGRATOR.
C
C  THE VARIABLES ..
C
C  DVALVS           SAVES THE DIFFUSION COEFFICIENTS FOR FUTURE
C                  EVALUATIONS.
C  DVALH            RETURNS FROM DH THE HORIZONTAL DIFFUSION COEFFI-
```

```
C                    CIENTS AND CONTAINS DUXX ON CALLS TO F.
C                    (SEE EXPLANATION OF DH AND F).
C   DVALV             RETURNS FROM DV THE VERTICAL DIFFUSION COEFFI-
C                    CIENTS AND CONTAINS DUYY ON CALLS TO F.
C                    (SEE EXPLANATION OF DV AND F).
C   ALPHAH,BETAH,
C     GAMMAH          CONTAIN THE HORIZONTAL BOUNDARY COEFFICIENTS
C                    PASSED TO PDETWO FROM BNDRYH (SEE EXPLANATION OF
C                    BNDRYH).
C   ALPHAV,BETAV,
C     GAMMAV          CONTAIN THE VERTICAL BOUNDARY COEFFICIENTS
C                    PASSED TO PDETWO FROM BNDRYV (SEE EXPLANATION OF
C                    BNDRYV).
C   UX,UY             STORE THE DU/DX AND DU/DY EVALUATIONS RESPECTIVELY.
C   DXI,DXIR,DXIC     STORE THE HORIZONTAL MESH SPACING.
C   XAVG              STORES THE AVERAGE BETWEEN TWO MESH POINTS ON THE
C                    HORIZONTAL AXIS.
C   UDVA,UDVB,
C     UDHR,UDHL       STORE THE DIFFERENCING FOR APPROXIMATING DUXX
C                    AND DUYY.
C   UAVGH,UAVGV       CONTAIN U AVERAGES FOR APPROXIMATING THE DIFFUSION
C                    COEFFICIENTS AT THE MESH MID-POINTS.
C
C   THE COMMON BLOCK VARIABLES..
C
C   X        (MESH1)  IS THE MESH SPACING IN THE HORIZONTAL DIRECTION.
C   Y        (MESH2)  IS THE MESH SPACING IN THE VERTICAL DIRECTION.
C   NPDE     (MESH3)  IS THE NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C   NY                IS THE NUMBER OF MESH POINTS IN THE VERTICAL
C                    DIRECTION.
C   NX                IS THE NUMBER OF MESH POINTS IN THE HORIZONTAL
C                    DIRECTION.
C
C   THE USER DEFINED ROUTINES CALLED BY PDETWO..
C
C
C   BNDRYH (T,X,Y,U,ALPHAH,BETAH,GAMMAH,NPDE) DEFINES THE COEFFICIENTS,
C                    ALPHAH,BETAH AND GAMMAH USED TO GENERATE THE HORIZON-
C                    TAL BOUNDARY CONDITION ALPHAH*U+BETAH*UY = GAMMAH
C                    IN PDETWO.
C   BNDRYV (T,X,Y,U,ALPHAV,GETAV,GAMMAV,NPDE) DEFINES THE COEFFICIENTS,
C                    ALPHAV,GETAV AND GAMMAV USED TO GENERATE THE VERTI-
C                    CAL BOUNDARY CONDITION ALPHAV*U+BETAV*UY = GAMMAV
C                    IN PDETWO.
C   DH (T,X,Y,U,DVALH,NPDE)  DEFINES THE HORIZONTAL DIFFUSION COEFFI-
C                    CIENTS DVALH USED IN APPROXIMATING THE TERM DUXX (THE
C                    TERM INVOLVING THE SECOND PARTIAL OF U AND THE
C                    DIFFUSION COEFFICIENT WITH RESPECT TO X).
C   DV (T,X,Y,U,DVALV,NPDE) DEFINES THE VERTICAL DIFFUSION COEFFI-
C                    CIENTS DVALV USED IN APPROXIMATING THE TERM DUYY (THE
C                    TERM INVOLVING THE SECOND PARTIAL OF U AND THE
C                    DIFFUSION COEFFICIENT WITH RESPECT TO Y).
C   F (T,X,Y,U,UX,UY,DVALH,DVALV,UDOT) DEFINES THE PDE*S TO BE SOLVED IN
C                    TERMS OF X,Y,U,UX (DU/DX),UY(DU/DY),DVALH(DUXX) AND
C                    DVALV(DUYY) AND RETURNS THE ODE*S, UDOT, ASSOCIATED
C                    WITH THE MESH POINT (X,Y).
C
C
C   THE USER MUST INCLUDE IN PDETWO A DIMENSION STATEMENT AS FOLLOWS..
C
```

```
C      DIMENSION DVALVS (NPDE,NPDE,NX,2),DVALV (NPDE,NPDE),
C     * DVALH (NPDE,NPDE,2),U (NPDE,NX,NY),X (NX),Y (NY),
C     * ALPHAH (NPDE),BETAH (NPDE),GAMMAH (NPDE), ALPHAV (NPDE),
C     * BETAV (NPDE),GAMMAV (NPDE),UX (NPDE),UY (NPDE), DXI (NX),
C     * DXIR (NX),DXIC (NX),XAVG (NX),UDVA (NPDE,NX),UDHR (NPDE),
C     * UDVB (NPDE),UDHL (NPDE),UAVGH (NPDE),UAVGV (NPDE),
C     * UDOT (NPDE,NX,NY)
C
C   WHERE THE ACTUAL NUMERICAL VALUES MUST BE SUBSTITUTED IN FOR NPDE,
C   NX AND NY.
C
C
C  DETERMINE THE MESH SPACING ALONG THE HORIZONTAL AXIS
C
       DXI(1)=1./(X(2)-X(1))
       DXIR(1)=DXI(1)
       DXIC(1)=2*DXI(1)
       XAVG(1)=.5*(X(2)+X(1))
       ILIM=NX-1
       ILIMY=NY-1
       DO 10 I=2,ILIM
         XAVG(I)=.5*(X(I+1)+X(I))
         DXI(I)=1./(X(I+1)-X(I-1))
         DXIR(I)=1./(X(I+1)-X(I))
   10    DXIC(I)=2*DXI(I)
      DXI(NX)=DXIR(ILIM)
      DXIC(NX)=2*DXI(NX)
C
C  THE FOLLOWING LOOP DETERMINES THE U APPROXIMATIONS FOR THE BOTTOM
C  AND TOP BOUNDARIES.  IC HAS THE VALUE OF 1 FOR THE TOP AND 2 FOR THE
C  BOTTOM
C
       DO 410 IC=1,2
         IF (IC .EQ. 1) GO TO 20
         M=1
         IS=1
         MN=2
         GO TO 30
   20    M=NY
         MN=ILIMY
         IS=-1
   30    DYI=1./(Y(MN)-Y(M))*IS
         DYIA=DYI
         DYIC=2*DYI
         YAVG=.5*(Y(M)+Y(MN))
C
C  DETERMINE THE BOUNDARY CONDITIONS (LEFT CORNER)
C
         CALL BNDRYH (T,X(2),Y(M),U(1,2,M),ALPHAH,BETAH,GAMMAH,NPDE)
         CALL BNDRYV (T,X(1),Y(MN),U(1,1,MN),ALPHAV,BETAV,GAMMAV,NPDE)
         DO 35 K=1,NPDE
           IF (BETAH(K) .EQ. 0.0) U(K,2,M)=GAMMAH(K)/ALPHAH(K)
           IF (BETAV(K) .EQ. 0.0) U(K,1,MN)=GAMMAV(K)/ALPHAV(K)
   35      CONTINUE
         CALL BNDRYH (T,X(1),Y(M),U(1,1,M),ALPHAH,BETAH,GAMMAH,NPDE)
         CALL BNDRYV (T,X(1),Y(M),U(1,1,M),ALPHAV,BETAV,GAMMAV,NPDE)
         ITEST=0
         DO 70 K=1,NPDE
           IF (BETAV(K) .NE. 0.0 .AND. BETAH(K) .NE. 0.0) GO TO 70
           IF (BETAH(K) .EQ. 0.0 .AND. BETAV(K) .NE. 0.0) GO TO 40
```

```
            IF (BETAH(K) .NE. 0.0 .AND. BETAV(K) .EQ. 0.0) GO TO 50
            U(K,1,M)=(GAMMAH(K)/ALPHAH(K)+GAMMAV(K)/ALPHAV(K))*.5
            ITEST=ITEST+1
            GO TO 60
   40       U(K,1,M)=GAMMAH(K)/ALPHAH(K)
            GO TO 60
   50       U(K,1,M)=GAMMAV(K)/ALPHAV(K)
   60           CONTINUE
   70       CONTINUE
         IF (ITEST .EQ. NPDE) GO TO 130
         CALL BNDRYH (T,X(1),Y(M),U(1,1,M),ALPHAH,BETAH,GAMMAH,NPDE)
         CALL BNDRYV (T,X(1),Y(M),U(1,1,M),ALPHAV,BETAV,GAMMAV,NPDE)
C
C EVALUATE THE D COEFFICIENTS (LEFT CORNER)
C
   80    CALL DH (T,X(1),Y(M),U(1,1,M),DVALH,NPDE)
         CALL DV (T,X(1),Y(M),U(1,1,M),DVALV,NPDE)
C
C EVALUATE DU/DX AND DU/DY (LEFT CORNER)
C
         DO 120 K=1,NPDE
            IF (BETAV(K) .NE. 0.0) GO TO 90
            UX(K)=DXI(1)*(U(K,2,M)-U(K,1,M))
            GO TO 100
   90       UX(K)=(GAMMAV(K)-ALPHAV(K)*U(K,1,M))/BETAV(K)
  100       IF (BETAH(K) .NE. 0.0) GO TO 110
            UY(K)=DYI*(U(K,1,MN)-U(K,1,M))*IS
            GO TO 120
  110       UY(K)=(GAMMAH(K)-ALPHAH(K)*U(K,1,M))/BETAH(K)
  120       CONTINUE
C
C CALCULATE THE U AVERAGES (LEFT CORNER)
C
  130    DO 140 K=1,NPDE
            UAVGH(K)=.5*(U(K,2,M)+U(K,1,M))
            UAVGV(K)=.5*(U(K,1,M)+U(K,1,MN))
            UDHR(K)=(U(K,2,M)-U(K,1,M))*DXIR(1)
            UDVA(K,1)=(U(K,1,MN)-U(K,1,M))*DYIA*IS
  140       CONTINUE
C
C CALCULATE THE D COEFFICIENTS AT THE MIDPOINTS OF THE INTERVALS
C BETWEEN THE LEFT CORNER AND THE NEIGHBORING POINTS
C
         CALL DV (T,X(1),YAVG,UAVGV,DVALVS(1,1,1,IC),NPDE)
         CALL DH (T,XAVG(1),Y(M),UAVGH,DVALH(1,1,2),NPDE)
         IF (ITEST .EQ. NPDE) GO TO 160
C
C CALCULATE DUXX AND DUYY (LEFT CORNER)
C
         DO 150 L=1,NPDE
           DO 150 K=1,NPDE
             DVALH(K,L,1)=DXIC(1)*(DVALH(K,L,2)*UDHR(L)-DVALH(K,L,1)
     *                        *UX(L))
             DVALV(K,L)=DYIC*(DVALVS(K,L,1,IC)*UDVA(L,1)-DVALV(K,L)
     *                        *UY(L))*IS
  150        CONTINUE
C
C EVALUATE THE RIGHT SIDE OF PDE (LEFT CORNER)
C
         CALL F (T,X(1),Y(M),U(1,1,M),UX,UY,DVALH,DVALV,UDOT(1,1,M),
```

```
     *           NPDE)
C
C SET UDOT = 0 FOR KNOWN BOUNDARY CONDITIONS
C
  160    DO 170 K=1,NPDE
           IF (BETAH(K) .EQ. 0.0 .OR. BETAV(K) .EQ. 0.0) UDOT(K,1,M)=0.0
  170      CONTINUE
C
C EVALUATE THE RIGHT CORNER BOUNDARY CONDITIONS
C
         CALL BNDRYH (T,X(NX),Y(M),U(1,NX,M),ALPHAH,BETAH,GAMMAH,NPDE)
         CALL BNDRYV (T,X(NX),Y(M),U(1,NX,M),ALPHAV,BETAV,GAMMAV,NPDE)
         ITEST = 0
         DO 210 K=1,NPDE
           IF (BETAV(K) .NE. 0.0 .AND. BETAH(K) .NE. 0.0) GO TO 210
           IF (BETAH(K) .EQ. 0.0 .AND. BETAV(K) .NE. 0.0) GO TO 180
           IF (BETAH(K) .NE. 0.0 .AND. BETAV(K) .EQ. 0.0) GO TO 190
           U(K,NX,M)=(GAMMAV(K)/ALPHAV(K)+GAMMAH(K)/ALPHAH(K))*.5
               ITEST=ITEST+1
           GO TO 200
  180      U(K,NX,M)=GAMMAH(K)/ALPHAH(K)
           GO TO 200
  190      U(K,NX,M)=GAMMAV(K)/ALPHAV(K)
  200        CONTINUE
  210      CONTINUE
         IBCK=1
         IFWD=2
C
C LOOP ON THE HORIZONTAL BOUNDARY MESH POINTS FROM X(2) TO X(NX-1)
C
         DO 300 I=2,ILIM
           K=IBCK
           IBCK=IFWD
           IFWD=K
C
C EVALUATE THE HORIZONTAL BOUNDARY CONDITIONS
C
           CALL BNDRYH (T,X(I+1),Y(M),U(1,I+1,M),ALPHAH,BETAH,GAMMAH,
     *                  NPDE)
           DO 215 K=1,NPDE
             IF (BETAH(K) .EQ. 0.0) U(K,I+1,M)=GAMMAH(K)/ALPHAH(K)
  215        CONTINUE
           CALL BNDRYH (T,X(I),Y(M),U(1,I,M),ALPHAH,BETAH,GAMMAH,NPDE)
           ITESTR=0
           DO 220 K=1,NPDE
             IF (BETAH(K) .NE. 0.0) GO TO 220
             U(K,I,M)=GAMMAH(K)/ALPHAH(K)
             ITESTR=ITESTR+1
  220        CONTINUE
           IF (ITESTR .EQ. NPDE) GO TO 260
           IF (ITESTR .EQ. 0.0) GO TO 230
           CALL BNDRYH (T,X(I),Y(M),U(1,I,M),ALPHAH,BETAH,GAMMAH,NPDE)
C
C CALCULATE DU/DX AND DU/DY (HORIZONTAL BOUNDARY)
C
  230      DO 250 K=1,NPDE
             UX(K)=(U(K,I+1,M)-U(K,I-1,M))*DXI(I)
             IF (BETAH(K) .NE. 0.0) GO TO 240
             UY(K)=(U(K,I,MN)-U(K,I,M))*DYI*IS
             GO TO 250
```

```
  240       UY(K)=(GAMMAH(K)-ALPHAH(K)*L(K,I,M))/BETAH(K)
  250       CONTINUE
C
C DETERMINE U AVERAGE (HORIZONTAL BOUNDARY)
C
  260     DO 270 K=1,NPDE
            UAVGV(K)=(U(K,I,M)+U(K,I,MN))*.5
            UAVGH(K)=(U(K,I+1,M)+U(K,I,M))*.5
            UDHL(K)=UDHR(K)
            UDHR(K)=(U(K,I+1,M)-U(K,I,M))*DXIR(I)
            UDVA(K,I)=(U(K,I,MN)-U(K,I,M))*DYIA*IS
  270       CONTINUE
C
C EVALUATE THE D COEFFICIENTS (HORIZONTAL BOUNDARY)
C
          CALL DV (T,X(I),YAVG,UAVGV,DVALVS(1,1,I,IC),NPDE)
          CALL DH (T,XAVG(I),Y(M),UAVGH,DVALH(1,1,IFWD),NPDE)
          IF (ITESTR .EQ. NPDE) GO TO 290
          CALL DV (T,X(I),Y(M),U(1,I,M),DVALV,NPDE)
C
C EVALUATE DUXX AND DUYY (HORIZONTAL BOUNDARY)
C
          DO 280 L=1,NPDE
            DO 280 K=1,NPDE
              DVALH(K,L,IBCK)=DXIC(I)*(DVALH(K,L,IFWD)*UDHR(L)-
     *                                 DVALH(K,L,IBCK)*UDHL(L))
              DVALV(K,L)=DYIC*(DVALVS(K,L,I,IC)*UDVA(L,I)-DVALV(K,L)
     *                                 *UY(L))*IS
  280         CONTINUE
C
C EVALUATE THE RIGHT SIDE OF THE PDE (HORIZONTAL BOUNDARY)
C
          CALL F (T,X(I),Y(M),U(1,I,M),UX,UY,DVALH(1,1,IBCK),
     *             DVALV,UDOT(1,I,M),NPDE)
C
C SET UDOT = 0 FOR KNOWN BOUNDARY CONDITIONS
C
  290   DO 300 K=1,NPDE
          IF (BETAH(K) .EQ. 0.0) UDOT(K,I,M)=0.0
  300     CONTINUE
C
C COMPLETE EVALUATING THE RIGHT CORNER
C
        CALL BNDRYV (T,X(NX),Y(MN),U(1,NX,MN),ALPHAV,BETAV,GAMMAV,NPDE)
        DO 305 K=1,NPDE
          IF (BETAV(K) .EQ. 0.0) U(K,NX,MN)=GAMMAV(K)/ALPHAV(K)
  305     CONTINUE
        IF (ITEST .EQ. NPDE) GO TO 360
        CALL BNDRYH (T,X(NX),Y(M),U(1,NX,M),ALPHAH,BETAH,GAMMAH,NPDE)
        CALL BNDRYV (T,X(NX),Y(M),U(1,NX,M),ALPHAV,BETAV,GAMMAV,NPDE)
C
C EVALUATE THE D COEFFICIENTS (RIGHT CORNER)
C
  310   CALL DH (T,X(NX),Y(M),U(1,NX,M),DVALH(1,1,IBCK),NPDE)
        CALL DV (T,X(NX),Y(M),U(1,NX,M),DVALV,NPDE)
C
C EVALUATE DU/DX AND DU/DY (RIGHT CORNER)
C
        DO 350 K=1,NPDE
          IF (BETAV(K) .NE. 0.0) GO TO 320
```

```
          UX(K)=DXI(NX)*(U(K,NX,M)-U(K,ILIM,M))
          GO TO 330
  320     UX(K)=(GAMMAV(K)-ALPHAV(K)*U(K,NX,M))/BETAV(K)
  330     IF (BETAH(K) .NE. 0.0) GO TO 340
          UY(K)=DYI*(U(K,NX,MN)-U(K,NX,M))*IS
          GO TO 350
  340     UY(K)=(GAMMAH(K)-ALPHAH(K)*U(K,NX,M))/BETAH(K)
  350     CONTINUE
C
C EVALUATE THE VERTICAL U AVERAGE (RIGHT CORNER)
C
  360   DO 370 K=1,NPDE
          UAVGV(K)=(U(K,NX,M)+U(K,NX,MN))*.5
          UDVA(K,NX)=(U(K,NX,MN)-U(K,NX,M))*DYI*IS
  370     CONTINUE
C
C EVALUATE THE VERTICAL D COEFFICIENT (ABOVE THE RIGHT CORNER)
C
        CALL DV (T,X(NX),YAVG,UAVGV,DVALVS(1,1,NX,IC),NPDE)
        IF (ITEST .EQ. NPDE) GO TO 390
C
C EVALUATE DUXX AND DUYY (RIGHT CORNER)
C
        DO 380 L=1,NPDE
          DO 380 K=1,NPDE
            DVALH(K,L,IBCK)=DXIC(NX)*(DVALH(K,L,IBCK)*UX(L)-
     *                      DVALH(K,L,IFWD)*UDHR(L))
            DVALV(K,L)=DYIC*(DVALVS(K,L,NX,IC)*UDVA(L,NX)-DVALV(K,L)
     *                      *UY(L))*IS
  380       CONTINUE
C
C EVALUATE THE RIGHT SIDE OF THE PDE (RIGHT CORNER)
C
        CALL F (T,X(NX),Y(M),U(1,NX,M),UX,UY,DVALH(1,1,IBCK),DVALV,
     *          UDOT(1,NX,M),NPDE)
  390   DO 400 K=1,NPDE
          IF (BETAH(K) .EQ. 0.0 .OR. BETAV(K) .EQ. 0.0) UDOT(K,NX,M)=0.0
  400     CONTINUE
  410   CONTINUE
C
C DETERMINE THE U VALUES FOR THE J-TH ROW
C IC IS USED TO DETERMINE IF THE LAST ROW IS TO BE DETERMINED, IF SO THE
C D COEFFICIENTS HAVE ALREADY BEEN DETERMINED FROM THE BOUNDARY SOLVED
C EARLIER SO IT IS A SPECIAL CASE
C
       IC=2
       DO 780 J=2,ILIMY
         IF (J .NE. ILIMY) GO TO 500
         IC=1
  500    DYI=1./(Y(J+1)-Y(J-1))
         DYIC=2*DYI
         DYIB=DYIA
         DYIA=1./(Y(J+1)-Y(J))
         YAVG=(Y(J+1)+Y(J))*.5
C
C
C DETERMINE THE LEFT BOUNDARY (J-TH ROW)
C
        IF (IC .EQ. 1) GO TO 506
        CALL BNDRYV (T,X(1),Y(J+1),U(1,1,J+1),ALPHAV,BETAV,GAMMAV,NPDE)
```

```
      DO 505 K=1,NPDE
        IF (BETAV(K) .EQ. 0.0) U(1,1,I+1)=GAMMAV(K)/ALPHAV(K)
505     CONTINUE
506   CONTINUE
      CALL BNDRYV (T,X(1),Y(J),U(1,1,J),ALPHAV,BETAV,GAMMAV,NPDE)
      ITESTJ=0
      DO 510 K=1,NPDE
        IF (BETAV(K) .NE. 0.0) GO TO 510
        U(K,1,J)=GAMMAV(K)/ALPHAV(K)
        ITESTJ=ITESTJ+1
510     CONTINUE
      IF (ITESTJ .EQ. NPDE) GO TO 560
      IF (ITESTJ .EQ. 0) GO TO 520
      CALL BNDRYV (T,X(1),Y(J),U(1,1,J),ALPHAV,BETAV,GAMMAV,NPDE)
C
C EVALUATE DU/DX AND DU/DY (LEFT BOUNDARY,J-TH ROW)
C
520   DO 550 K=1,NPDE
        IF (BETAV(K) .NE. 0.0) GO TO 530
        UX(K)=(U(K,2,J)-U(K,1,J))*DXIR(1)
        GO TO 540
530     UX(K)=(GAMMAV(K)-ALPHAV(K)*U(K,1,J))/BETAV(K)
540     UY(K)=(U(K,1,J+1)-U(K,1,J-1))*DYI
550     CONTINUE
C
C EVALUATE THE HORIZONTAL U AVERAGE (J-TH ROW)
C
560   DO 570 K=1,NPDE
        UAVGH(K)=(U(K,2,J)+U(K,1,J))*.5
        UDHR(K)=(U(K,2,J)-U(K,1,J))*DXIR(1)
        UDVB(K)=UDVA(K,1)
        UDVA(K,1)=(U(K,1,J+1)-U(K,1,J))*DYIA
570     CONTINUE
C
C EVALUATE THE D COEFFICIENTS (LEFT BOUNDARY,J-TH ROW)
C
      DO 580 L=1,NPDE
        DO 580 K=1,NPDE
          DVALV(K,L)=DVALVS(K,L,1,2)
580       CONTINUE
      CALL DH (T,X(1),Y(J),U(1,1,J),DVALH,NPDE)
      CALL DH (T,XAVG(1),Y(J),UAVGH,DVALH(1,1,2),NPDE)
      IF (IC .EQ. 1) GO TO 590
C
C EVALUATE THE VERTICAL U AVERAGE (J-TH ROW)
C
      DO 585 K=1,NPDE
        UAVGV(K)=(U(K,1,J+1)+U(K,1,J))*.5
585     CONTINUE
      CALL DV (T,X(1),YAVG,UAVGV,DVALVS(1,1,1,2),NPDE)
C
C EVALUATE DUXX AND DUYY (LEFT BOUNDARY,J-TH ROW)
C
590   IF (ITESTJ .EQ. NPDE) GO TO 610
      DO 600 L=1,NPDE
        DO 600 K=1,NPDE
          DVALH(K,L,1)=DXIC(1)*(DVALH(K,L,2)*UDHR(L)-
     *                  DVALH(K,L,1)*UX(L))
          DVALV(K,L)=DYIC*(DVALVS(K,L,1,IC)*UDVA(L,1)-DVALV(K,L)
     *                  *UDVB(L))
```

```
  600       CONTINUE
C
C EVALUATE THE RIGHT SIDE OF THE PDE (LEFT BOUNDARY,J-TH ROW)
C
      CALL F (T,X(1),Y(J),U(1,1,J),UX,UY,DVALH,DVALV,UDOT(1,1,J),NPDE)
C
C SET UDOT = 0 FOR KNOWN LEFT BOUNDARY CONDITIONS
C
  610 DO 620 K=1,NPDE
        IF (BETAV(K) .EQ. 0.0) UDOT(K,1,J)=0.0
  620   CONTINUE
C
C EVALUATE THE RIGHT BOUNDARY FOR THE J-TH ROW
C
      CALL BNDRYV (T,X(NX),Y(J),U(1,NX,J),ALPHAV,BETAV,GAMMAV,NPDE)
      ITESTJ=0
      DO 630 K=1,NPDE
        IF (BETAV(K) .NE. 0.0) GO TO 630
        U(K,NX,J)=GAMMAV(K)/ALPHAV(K)
        ITESTJ=ITESTJ+1
  630   CONTINUE
C
C LOOP TO EVALUATE U IN THE CENTER OF THE GRID (2<I<NX-1,2<J<NY-1)
C
      IBCK=1
      IFWD=2
      DO 680 I=2,ILIM
        K=IBCK
        IBCK=IFWD
        IFWD=K
C
C CALCULATE THE HORIZONTAL U AVERAGE, DU/DX AND DU/DY
C (I-TH POINT OF THE J-TH ROW)
C
        DO 640 K=1,NPDE
          UAVGH(K)=(U(K,I+1,J)+U(K,I,J))*.5
          UX(K)=(U(K,I+1,J)-U(K,I-1,J))*DXI(I)
          UY(K)=(U(K,I,J+1)-U(K,I,J-1))*DYI
  640     CONTINUE
C
C  EVALUATE THE D COEFFICIENTS (I-TH POINT OF THE J-TH ROW)
C
        DO 650 L=1,NPDE
          DO 650 K=1,NPDE
            DVALV(K,L)=DVALVS(K,L,I,2)
  650       CONTINUE
        CALL DH (T,XAVG(I),Y(J),UAVGH,DVALH(1,1,IFWD),NPDE)
        IF (IC .EQ. 1) GO TO 660
        DO 655 K=1,NPDE
          UAVGV(K)=(U(K,I,J+1)+U(K,I,J))*.5
  655     CONTINUE
        CALL DV (T,X(I),YAVG,UAVGV,DVALVS(1,1,I,2),NPDE)
C
C EVALUATE DUXX AND DUYY (I-TH POINT OF THE J-TH ROW)
C
  660   DO 670 L=1,NPDE
          UDHL(L)=UDHR(L)
          UDHR(L)=(U(L,I+1,J)-U(L,I,J))*DXIR(I)
          UDVB(L)=UDVA(L,I)
          UDVA(L,I)=(U(L,I,J+1)-U(L,I,J))*DYIA
```

```
            DO 670 K=1,NPDE
              DVALH(K,L,IBCK)=DXIC(I)*(DVALH(K,L,IFWD)*UDHR(L)-
     *                         DVALH(K,L,IBCK)*UDHL(L))
              DVALV(K,L)=DYIC*(DVALVS(K,L,I,IC)*UDVA(L,I)-DVALV(K,L)
     *                         *UDVB(L))
  670         CONTINUE
C
C EVALUATE THE RIGHT SIDE OF THE PDE (I-TH POINT OF THE J-TH ROW)
C
            CALL F (T,X(I),Y(J),U(1,I,J),UX,UY,DVALH(1,1,IBCK),DVALV,
     *              UDOT(1,I,J),NPDE)
  680     CONTINUE
C
C EVALUATE THE RIGHT BOUNDARY FOR THE J-TH ROW
C
          IF (IC .EQ. 1) GO TO 686
          CALL BNDRYV (T,X(NX),Y(J+1),U(1,NX,J+1),ALPHAV,BETAV,GAMMAV,
     *                 NPDE)
          DO 685 K=1,NPDE
            IF (BETAV(K) .EQ. 0.0) U(K,NX,J+1)=GAMMAV(K)/ALPHAV(K)
  685       CONTINUE
  686     CONTINUE
          IF (ITESTJ .EQ. NPDE) GO TO 730
          IF (ITESTJ .EQ. 0) GO TO 690
          CALL BNDRYV (T,X(NX),Y(J),U(1,NX,J),ALPHAV,BETAV,GAMMAV,NPDE)
C
C EVALUATE DU/DX AND DU/DY (RIGHT BOUNDARY,J-TH ROW)
C
  690     DO 710 K=1,NPDE
            UY(K)=(U(K,NX,J+1)-U(K,NX,J-1))*DYI
            IF (BETAV(K) .NE. 0.0) GO TO 700
            UX(K)=(U(K,NX,J)-U(K,ILIM,J))*DXIR(ILIM)
            GO TO 710
  700       UX(K)=(GAMMAV(K)-ALPHAV(K)*U(K,NX,J))/BETAV(K)
  710       CONTINUE
C
C EVALUATE THE D COEFFICIENTS (RIGHT BOUNDARY,J-TH ROW)
C
          DO 720 L=1,NPDE
            DO 720 K=1,NPDE
              DVALV(K,L)=DVALVS(K,L,NX,2)
  720         CONTINUE
          CALL DH (T,X(NX),Y(J),U(1,NX,J),DVALH(1,1,IBCK),NPDE)
  730     DO 740 K=1,NPDE
            UAVGV(K)=(U(K,NX,J+1)+U(K,NX,J))*.5
            UDVB(K)=UDVA(K,NX)
            UDVA(K,NX)=(U(K,NX,J+1)-U(K,NX,J))*DYIA
  740       CONTINUE
          IF (IC .EQ. 1) GO TO 750
          CALL DV (T,X(NX),YAVG,UAVGV,DVALVS(1,1,NX,2),NPDE)
  750     IF (ITESTJ .EQ. NPDE) GO TO 770
C
C EVALUATE DUXX AND DUYY (RIGHT BOUNDARY,J-TH ROW)
C
          DO 760 L=1,NPDE
            DO 760 K=1,NPDE
              DVALH(K,L,IBCK)=DXIC(NX)*(DVALH(K,L,IBCK)*UX(L)-
     *                          DVALH(K,L,IFWD)*UDHR(L))
              DVALV(K,L)=DYIC*(DVALVS(K,L,NX,IC)*UDVA(L,NX)-DVALV(K,L)
     *                         *UDVB(L))
```

```
  760       CONTINUE
C
C EVALUATE THE RIGHT SIDE OF THE PDE (RIGHT BOUNDARY,J-TH ROW)
C
        CALL F (T,X(NX),Y(J),U(1,NX,J),UX,UY,DVALH(1,1,IBCK),DVALV,
     *          UDOT(1,NX,J),NPDE)
C
C SET UDOT = 0 FOR KNOWN BOUNDARY VALUES
C
  770 DO 780 K=1,NPDE
        IF (BETAV(K) .EQ. 0.0) UDOT(K,NX,J)=0.0
  780   CONTINUE
      RETURN
      END
```

APPENDIX B

```
      SUBROUTINE PSETM (Y,NO,CON,MITER,IER)
C
C
C       PSETM IS SPECIFICALLY DESIGNED TO REPLACE THE ROUTINE PSETB
C  USED IN GEARB.  IT IS INTENDED TO BE USED WITH PDETWO WHICH USES
C  FIVE POINT CENTERED DIFFERENCING TO GENERATE A SYSTEM OF ODE*S.
C  PSETM OPTIMIZES THE NUMBER OF FUNCTION EVALUATIONS NEEDED TO
C  GENERATE THE JACOBIAN MATRIX.
C
C       PSETM IS CALLED BY STIFFB IN GEARB WHEN THE INTEGRATION METHOD
C  (MITER=1 OR 2) REQUIRES A JACOBIAN MATRIX.  IT COMPUTES AND PROCESSES
C  THE MATRIX P=I-H*EL(1)*J, WHERE J IS THE JACOBIAN. J IS CALCULATED BY
C  USING A USER-SUPPLIED ROUTINE PDB (MITER = 1) OR BY USING FINITE
C  DIFFERENCING (MITER = 2).  IF MITER = 1, PSETM GENERATES THE JACOBIAN
C  WITH A USER SUPPLIED ROUTINE.  THIS METHOD IS USUALLY AVOIDED SINCE
C  IT REQUIRES THE USER TO COMPLETELY UNDERSTAND PDETWO.  IF MITER = 2,
C  J IS APPROXIMATED BY (UDOT(Y+R)-UDOT(Y))*D.  UDOT(Y) IS SUPPLIED
C  BY STIFFB THROUGH THE COMMON BLOCK GEAR4.  D IS A SMALL NUMBER
C  CALCULATED IN PSETM.  PSETM DETERMINES THE MESH POINTS FOR WHICH
C  UDOT(Y+R) IS EVALUATED IN EACH CALL TO PDETWO.  AFTER THE CALL TO
C  PDETWO, THE ASSOCIATED ENTRIES IN THE JACOBIAN ARE THEN EVALUATED.
C  PSETM USES SIX CALLS TO PDETWO FOR EACH PDE IN ORDER TO GENERATE THE
C  JACOBIAN.  EACH CALL DETERMINES UDOT(Y+R) FOR THE MESH POINTS IN
C  THE FOLLOWING PATTERN..
C
C                  O X X O X X O
C                  X X X X X X X
C                  X O X X O X X
C                  X X X X X X X
C                  O X X O X X O
C
C  WHERE O REPRESENTS THE POINTS FOR WHICH UDOT(Y+R) IS EVALUATED.
C  ASSOCIATED WITH EACH MESH POINT ( O OR X ) IS A SOLUTION FOR EACH
C  OF THE PDE*S.  THEREFORE TO COMPLETELY EVALUATE J, THE PATTERN
C  SHOWN ABOVE MUST BE EVALUATED FOR EACH PDE.
C  THE TOTAL NUMBER OF FUNCTION EVALUATIONS FOR PSETM IS THEN NPDE*6
C  AS OPPOSED TO MW CALLS BY PSETB (SEE MW EXPLANATION BELOW).
C
C                       REFERENCES
C
C       1.  A. C. HINDMARSH, GEAR.. ORDINARY DIFFERENTIAL EQUATION
C             SYSTEM SOLVER, UCID-30001 REV. 3, LAWRENCE LIVERMORE
C             LABORATORY, P.O.BOX 808, LIVERMORE, CA 94550, DEC. 1974.
C
C       2.  A. C. HINDMARSH, GEARB.. SOLUTION OF ORDINARY
C             DIFFERENTIAL EQUATIONS HAVING BANDED JACOBIAN,
C             UCID-30059 REV. 1, L.L.L., MARCH 1975.
C
C  THE INPUT PARAMETERS..
C
C  Y                 CONTAINS THE CURRENT SOLUTIONS OF THE ORDINARY
C                   DIFFERENTIAL EQUATIONS.
C  NO                IS THE NUMBER OF ORDINARY DIFFERENTIAL EQUATIONS.
C  CON               IS THE CONSTANT (-H*EL(1)).
C  MITER             INDICATES THE TYPE OF ITERATION METHOD BEING USED
C                   BY THE INTEGRATOR.
```

```
C                    MITER = 0 MEANS FUNCTIONAL ITERATION.
C                    MITER = 1 MEANS THE CHORD METHOD WITH AN ANALYTIC
C                              JACOBIAN SUPPLIED IN THE USER DEFINED
C                              ROUTINE PDB.
C                    MITER = 2 MEANS THE CHORD METHOD WITH THE JACOBIAN
C                              CALCULATED IN PSETM.
C                    MITER = 3 MEANS THE CHORD METHOD WITH THE JACOBIAN
C                              REPLACED BY A DIAGONAL APPROXIMATION
C                              BASED ON A DIRECTIONAL DERIVATIVE.
C                              THIS METHOD IS IN GEARB, BUT IS NOT USEFUL
C                              IN SOLVING PDE*S.
C   IER                 IS AN ERROR INDICATOR USED IN THE ROUTINE DECB.
C
C   THE VARIABLES ..
C
C   R                    IS A SMALL INCREMENT USED IN EVALUATING THE
C                       FUNCTION NEAR THE CURRENT SOLUTION Y ( I.E.
C                       UDOT(Y+R)).
C   RO                   IS A LOWER BOUND ON THE SIZE OF R.
C   D                    IS A SMALL NUMBER USED TO APPROXIMATE THE
C                       DERIVATIVE.
C
C   THE COMMON BLOCK VARIABLES ..
C
C   UROUND  (GEAR1)      IS THE UNIT ROUNDOFF OF THE MACHINE.
C   YMAX    (GEAR2)      IS THE MAXIMUM Y VALUES, WHICH ARE USED TO SCALE
C                       THE VALUE IN R AND FOR ERROR CONTROL IN GEARB.
C   ERROR   (GEAR3)      IS A TEMPORY STORE FOR Y VALUES.
C   SAVE1   (GEAR4)      IS THE VALUE OF UDOT(Y+R) PASSED BACK FROM PDETWO.
C   SAVE2   (GEAR5)      IS THE CURRENT VALUE OF UDOT(Y) PASSED FROM STIFFB.
C   PW      (GEAR6)      STORES J AND RETURNS P WHEN RETURNING TO STIFFB.
C   MW      (GEAR8)      IS NO - 1.
C   EPSJ                 IS SQRT(UROUND).
C   MW                   IS THE BAND WIDTH OF THE JACOBIAN (ML+MU+1).
C   ML                   IS THE WIDTH OF THE LOWER HALF OF THE BAND.  ML =
C                       (NX + 1) * NPDE - 1).
C   MU                   IS THE WIDTH OF THE UPPER HALF OF THE BAND.  MU =
C                       (NX + 1) * NPDE - 1).
C                       MW,ML AND MU ARE USED IN THE DIRECT SOLUTION OF J IN
C                       DECB.
C   NPDE    (MESH3)      IS THE NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C   NY                   IS THE NUMBER OF MESH POINTS IN THE VERTICAL
C                       DIRECTION.
C   NX                   IS THE NUMBER OF MESH POINTS IN THE HORIZONTAL
C                       DIRECTION.
C
C  THE ROUTINES CALLED BY PSETM..
C
C   PDB(N,T,Y,PW,NO,ML,MU) IS A USER DEFINED ROUTINE WHICH DEFINES J IN
C                       PW IF MITER = 1.
C   DECB(NO,N,ML,MU,PW,IPIV,IER) COMPUTES THE LU DECOMPOSITION OF J USED
C                       FOR THE DIRECT SOLUTION OF J.
C   PDETWO (N,T,Y,SAVE) IS THE INTERFACE WHICH CONVERTS A TWO
C                       DIMENSIONAL SYSTEM OF PDE*S INTO A SYSTEM OF ODE*S.
C
C
C   NOTE:  IF THE USER WANTS TO KNOW THE NUMBER OF FUNCTION EVALUATIONS
C   WHEN USING PSETM HE MUST PROVIDE HIS OWN COUNTER AS NFE PROVIDED IN
C   COMMON BLOCK GEAR9 IS NO LONGER CORRECT.
C
```

```
C
      INTEGER NO,MITER,IER,N,IDUMMY,IPIV,ML,MU,MW,NM1,NCML,NOW,
     *   I,J,KMAX,K,J1,JJ,I1,I2,II
      DOUBLE PRECISION Y,CON,T,H,DUMMY,URCUND,YMAX,ERRCR,
     *   SAVE1,SAVE2,PW,EPSJ,D,RO,YM,R
      DIMENSION Y(NC,1)
      COMMON /GEAR1/ T,H,DUMMY(3),URCUND,N,IDUMMY(3)
      COMMON /GEAR2/ YMAX(1)
      COMMON /GEAR3/ ERRCR(1)
      COMMON /GEAR4/ SAVE1(1)
      COMMON /GEAR5/ SAVE2(1)
      COMMON /GEAR6/ PW(1600)
      COMMON /GEAR7/ IPIV(1)
      COMMON /GEAR8/ EPSJ,ML,MU,MW,NM1,NCML,NOW
      COMMON /PRT/ IPNT
      COMMON /MESH3/ NPDE,NX,NY
C
C  IF MITER = 1, CALL PDB TO GENERATE THE JACOBIAN
C
      IF (MITER .EQ. 2) GO TO 20
      CALL PDB (N, T, Y, PW, NO, ML, MU)
      DO 10 I = 1,NOW
 10     PW(I) = PW(I)*CON
      GO TO 230
C
C  IF MITER = 2, APPROXIMATE J WITH FINITE DIFFERENCING
C
 20   D = 0.D0
      DO 30 I = 1,N
 30     D = D + SAVE2(I)**2
      RO = DABS(H)*DSQRT(D)*1.D03*URCUND
C
C  THE Y(,1) VALUES ARE SAVED TEMPORARILY IN ERRCR.
C
      DO 40 I = 1,N
 40     ERROR(I) = Y(I,1)
C
C  UP TO THIS POINT, PSETB AND PSETM ARE IDENTICAL.
C
      IN3=3*NPDE
      IN2M=2*NPDE-1
      ISTART = IN2M*NO
      NN=NX*NPDE
      NNM=NM1*NPDE
      ID=(NPDE-1)*NC
      IA=NN*NO
      NROW2=NN*2
      NP=NO*(NROW2+IN2M)
      DO 45 II=1,NP
   45   PW(II)=0.0
C
C  EVALUATE THE MESH FOR EACH PDE.
C
      DO 160 IPDE=1,NPDE
        IPDEM=IPDE-1
        IPIN=NM1*IPDEM
C
C  PROVIDE THE SIX FUNCTION EVALUATIONS NECESSARY FOR EACH PDE.
C
        DO 150 K4=1,2
```

```
            DO 140 K3=1,IN3,NPDE
              IS=1
              IF (K3 .EQ. 2*NPDE+1) IS=-2
              I1=IPDEM+K3+(K4-1)*NN
              I2=NN*K4
              ISV=ISTART
C
C   SELECT THE MESH POINTS FOR WHICH UDOT(Y+R) IS TO BE EVALUATED.
C
              DO 60 K2=K4,NY,2
                DO 50 K1=I1,I2,IN3
                  R = DMAX1(EPSJ*YMAX(K1),RO)
                  Y(K1,1)=Y(K1,1)+R
                  ISV=ISV+1
                  PW(ISV)=K1
   50             CONTINUE
                I1=I1+NROW2+IS*NPDE
                I2=I2+NROW2
   60           IS=-IS
C
C   EVALUATE UDOT(Y+R)
C
         CALL DIFFUN (N,T,Y,SAVE1)
C
C   GENERATE THE JACOBIAN BY EVALUATING (UDOT(Y+R)-UDOT(Y))*D.
C   THE ENTRIES IN THE JACOBIAN ARE FIRST DETERMINED FOR THE CENTER BAND,
C   THEN THE UPPER BAND AND FINALLY THE LOWER BAND.  THIS IS REPEATED
C   FOR ALL PDE*S.  THE FINAL STEP THEN IS TO ZERO THE APPROPRIATE
C   ENTRIES CAUSED BY THE VERTICAL BOUNDARIES.
C
              J1=ISTART+1
              DO 130 IC=J1,ISV
                JJ=PW(IC)
                R=DMAX1(EPSJ*YMAX(JJ),RO)
                D=CON/R
                II=IA+ID+JJ+IPIN
                IF (JJ .LE. NPDE) GO TO 70
                II=II+NNM
                I1=JJ-IPDEM-NPDE
                I2=I1+IN3-1
                IF (JJ .GT. NC-NPDE) I2=I2-NPDE
                GO TO 80
   70           I1=1
                I2=2*NPDE
C
C   GENERATE ENTRIES FOR THE MIDDLE BAND
C
   80           DO 90 I=I1,I2
                  PW(II)=(SAVE1(I)-SAVE2(I))*D
   90             II=II-NM1
                IF (JJ .GT. NC-NN) GO TO 110
                II=ID+JJ+NN+IPIN
                I1=JJ-IPDEM+NN
                I2=I1+NPDE-1
C
C   GENERATE ENTRIES FOR THE UPPER BAND
C
                DO 100 I=I1,I2
                  PW(II)=(SAVE1(I)-SAVE2(I))*D
  100             II=II-NM1
```

```
              IF (JJ .LE. NN) GO TO 130
  110         II=ID+2*IA+JJ-NN+IPIN
              I1=JJ-IPDEM-NN
              I2=I1+NPDE-1
C
C  GENERATE ENTRIES FOR THE LOWER BAND.
C
              DO 120 I=I1,I2
                PW(II)=(SAVE1(I)-SAVE2(I))*C
  120           II=II-NM1
  130           Y(JJ,1)=ERROR(JJ)
                ERROR(JJ) = 0.D0
  140       CONTINUE
  150     CONTINUE
  160   CONTINUE
        ISN=-NO
        DO 200 I4=1,2
          II2=IA+ID+1
          DO 190 I3=1,IN2M
            II2=II2+ISN
            III=II2
            DO 180 I2=2,NY
              III=III+NN
              II0=III+I3-1
C
C ZERO APPROPRIATE ENTRIES IN THE CENTER BAND
C
              DO 170 II=III,II0
  170         PW(II)=0.0
  180         CONTINUE
  190       CONTINUE
        ISN=NM1
  200   CONTINUE
        I1=ISTART+1
        I2=I1+.34*NO
        DO 220 II=I1,I2
  220   PW(II)=0.0
C
C  THE REMAINDER OF PSETM IS THE SAME AS PSETB.
C  ADD THE IDENTITY MATRIX.
C
  230 DO 240 I = 1,N
  240   PW(NOML+I) = PW(NCML+I) + 1.DO
C
C  DO THE LU DECOMPOSITION ON P.
C
        CALL DECB (NO, N, ML, MU, PW, IPIV, IER)
        RETURN
        END
```

GENERAL SOFTWARE FOR TWO DIMENSIONAL PARTIAL DIFFERENTIAL EQUATIONS

BY

David Kennett Melgaard

B. S., New Mexico State University, 1970

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

1976

GENERAL SOFTWARE FOR TWO DIMENSIONAL PARTIAL DIFFERENTIAL EQUATIONS

David K. Melgaard, Kansas State University

ABSTRACT

The numerical solution of partial differential equations (PDE's) is generally not only a complicated process but is highly problem dependent as well, requiring the long and difficult task of developing new software for each problem. Recently, easy to use, portable and efficient general software for the solution of one dimensional PDE's has become available. In this paper, we present a software interface, which is an extension of this existing software, to solve PDE's with one dimension in time and two spatial dimensions. The software interface employs the method of lines technique whereby centered differencing of the spatial variables results in a system of time dependent ordinary differential equations (ODE's) which are then solved using one of the already developed robust ODE integrators. Included in this paper will be a detailed description of the use of the interface. We also discuss the efficient generation of the Jacobian matrix. Representative examples will be given to demonstrate the use, validity, capabilities, and limitations of the software interface.