A Prototyping Environment for Differential Equations

TOUFIC I. BOUBEZ
Rutgers University
and
ANDY M. FRONCIONI and RICHARD L. PESKIN
Rutgers University

A system is presented to allow end users to solve nonlinear differential equations without need to write computer programs. The system treats *n*th order space (one dimensional), first order time systems with initial and/or two point boundary value specification. Users of the system need only enter the problem in direct mathematical notation, and output is automatically presented as a solution graph. The system allows the user to alter this equations, in-situ, that is to computationally steer his model. Thus the system is suited for model prototyping. Implementation is based on an object-oriented paradigm, well established and robust numerical procedures, and distributed computing to supported needed resources for numerically intensive tasks.

Categories and Subject Descriptors: D.2.m [**Software Engineering**]: Miscellaneous; D.3.2 [**Programming Languages**]: Language Classifications—*Smalltalk*; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations

General Terms: Design, Experimentation, Languages

1. INTRODUCTION

In a general scientific interface environment, an interactive tool to allow scientists the capability of experimentation with differential equations is a desirable feature. The need to enhance scientific interfaces for numerical

Editor's Note: This paper was selected as one of the two best presented at the Second International Conference on Expert Systems for Numerical Computing, held at West Lafayette, Indiana, April 22–24, 1990. The complete proceedings is published by North-Holland under the title *Intelligent Scientific Software Systems.*

This research was supported in part by the Parallel Computing Laboratory of the Center for Computer Aids For Industrial Productivity (CAIP) and in part by the National Science Foundation NSF grant EET88-14937. CAIP is supported by the New Jersey Commission on Science and Technology, Rutgers—the State University of New Jersey, and the CAIP Industrial Members.

Authors' addresses: T. I. Boubez, Department of Biomedical Engineering, and CAIP Parallel Computing Laboratory, Rutgers University, Piscataway, NJ 08855–1390, email: boubez@caip.rutgers.edu; A. M. Froncioni and R. L. Peskin, Department of Mechanical and Aerospace Engineering, and CAIP Parallel Computing Laboratory, Rutgers University, Piscataway, NJ 08855-1390.

simulation has been discussed by Peskin et al. [4, 5], where the overall concepts of our Scientific Computing Environment for Numerical Experimentation (SCENE) environment are described. More specifically, when dealing with differential equations, there is need for a tool that is capable of handling a reasonable variety of initial value, final value, and boundary value problems (including cases where boundary layers and shocks are present). Campbell [1] has built a prototype differential equation tool in Suntools. While this system has a graphical interface, its model flexibility is limited. Russo [7] has an extensive knowledge-based system capable of handling a wide class of problems, but with limited graphical and interactive interface. By constructing a differential equation system in Smalltalk-80™, we are able to combine model flexibility and complete graphical interaction capabilities, while taking full advantage of the distributed processing environment available.

In this paper, we present a differential equation (DE) tool that is designed to accept user input in the form of a problem statement (string form) of a single DE or a set of coupled DEs and the corresponding initial and boundary conditions. The output is a graphical display of the solution. User interaction with the input allows rapid change of the equations, parameter, boundary conditions, and any other parameters such as the discretization resolution and the solution domain; in effect, the user can "steer" the model.

2. NUMERICAL SOLUTIONS

In general, the nth-order differential equation

$$y^{(n)} = f\left(x, y, y^{(1)}, \ldots, y^{(n-1)}\right) \tag{1}$$

can be reduced to a set of n first-order equations

$$y_i' = g_i\left(x, y_1, \ldots, y_n\right) \qquad i = 1, \ldots, n \tag{2}$$

by using some auxiliary functions. For a set of m coupled DEs, the process is repeated for each equation, resulting in $N = n \times m$ first-order equations. This set of equations can then be solved by using any one of several methods, including shooting and relaxation. This being a prototyping environment, emphasis was placed on the robustness of the solution method, and a relaxation scheme was chosen.

For a relaxation scheme, this set of first-order equations is rewritten as a set of finite difference equations (FDEs) for each of the interior points in the discretized domain. Several discretization schemes can be used, including the forward-differencing one that is used by the tool:

$$y_i = \frac{1}{2}\left(y_{i,k+1} + y_{i,k}\right)$$

$$y_i' = \frac{1}{\Delta x}\left(y_{i,k+1} - y_{i,k}\right)$$

™ Smalltalk-80 is a registered trademark of Xerox Corp.

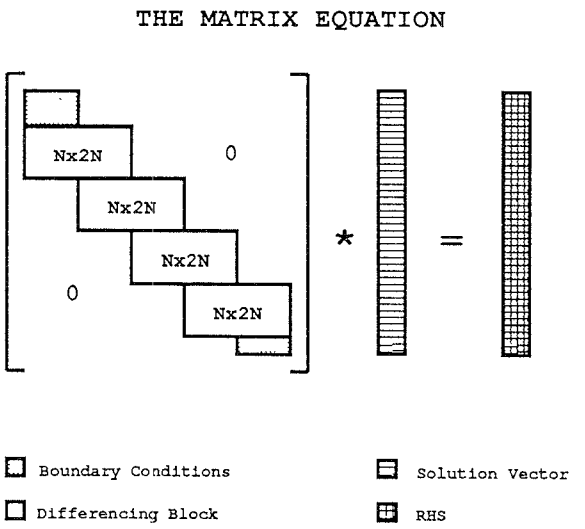


Figure 1

where $y_{i,k}$ is the value of the function y_i at the point k in the discretized domain. In vector notation, this results in a set of equations $\mathbf{E}_k$ at each point in the interior domain. These FDEs are linearized by rewriting them as a set of linear equations in the highest derivatives, taking the nonlinear terms from the previous iteration step. The solution to the FDE problem is then found by successive relaxation of an initial guess. For this purpose, the equations are expanded in a first-order Taylor series with respect to small changes Δy_k [6]:

$$\mathbf{E}_k(y_k + \Delta y_k, y_{k-1} + \Delta y_{k-1}) \approx \mathbf{E}_k(y_k, y_{k-1}) + \sum_{n=1}^{N} \frac{\partial \mathbf{E}_k}{\partial y_{n,k-1}} \Delta y_{n,k-1} + \sum_{n=1}^{N} \frac{\partial \mathbf{E}_k}{\partial y_{n,k}} \Delta y_{n,k}.$$

By setting the updated value $\mathbf{E}_k(y_k + \Delta y_k, y_{k-1} + \Delta y_{k-1})$ to be zero at the solution, an $N \times 2N$ matrix block is contributed by each internal point. A similar treatment of the boundary conditions results in a complete matrix equation if the form (Figure 1):

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3}$$

relating all the interior points in the discretized domain and incorporating the boundary conditions. This equation is more specifically written as:

$$\mathbf{A}^{(l-1)} \cdot \mathbf{x}^{(l)} = \mathbf{b}^{(l-1)} \tag{4}$$

and solved in an iteration loop, using the solution from the previous iteration $(l-1)$ to construct the matrix and solve for the next iteration (l). The solution is thus relaxed until convergence is reached.

Symbolic Processing Steps

Initial Equation

uxx + u	$dy^2/dx^2 = -y$

Order Reduction

u1 - a0 a1 + u0	$dy_1/dx = y_2$ $dy_2/dx = -y_1$

FDE Expansion

(1/DX)*(ua0-us1) - 0.5*(aa0+as1)	$\frac{1}{\Delta x}(y_{1\,k} - y_{1\,k-1}) = \frac{1}{2}(y_{2\,k} + y_{2\,k-1})$
(1/DX)*(aa0-as1) + 0.5*(ua0+us1)	$\frac{1}{\Delta x}(y_{2\,k} - y_{2\,k-1}) = -\frac{1}{2}(y_{1\,k} + y_{1\,k-1})$

Matrix Block

$$\begin{bmatrix} -(1/DX) & -0.5 & (1/DX) & -0.5 \\ 0.5 & -(1/DX) & 0.5 & (1/DX) \end{bmatrix} \begin{bmatrix} us1 \\ as1 \\ ua0 \\ aa0 \end{bmatrix} = \begin{bmatrix} -E_1 \\ -E_2 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{1}{\Delta x} & -\frac{1}{2} & \frac{1}{\Delta x} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{\Delta x} & \frac{1}{2} & \frac{1}{\Delta x} \end{bmatrix} \begin{bmatrix} y_{1\,k-1} \\ y_{2,k-1} \\ y_{1,k} \\ y_{2,k} \end{bmatrix} = \begin{bmatrix} -E_1 \\ -E_2 \end{bmatrix}$$

Solution

Automatic Script Generation — Numerical Program Implementation

Figure 2

3. NUMERICAL SOLUTIONS WITH THE DE TOOL

In standard numerical methods, the equations are usually reduced and prepared beforehand, and all the algebraic steps described in the previous section are performed by the user. The computer is only used in the final solution steps that require number crunching (matrix solutions and relaxation iterations). The purpose of using our prototyping environment is to automate the initial stages as well as the numerical computation steps, so that an equation is processed from string form to a solution plot, while still allowing user interaction during the solution process. The following steps are shown in Figure 2.

The most important user system interface step is the initial one, that of problem formulation. The problem equation has to be entered, parsed, and processed. Subclassing from Smalltalk classes, a string expression can easily be converted into an equation by using a scanner and supplying it with the necessary token dictionaries. The equation $u_{xx} = -u$, for example, is entered as: 'uxx + u' and, when scanned, produces a new instance of EquationList, a subclass of Array, having the value (uxx + u). In the process, a set of higher-order derivative variables (a, b, c, . . .) are defined such that a = du/dx, b = da/dx, etc., and serve as intermediate variables.

The m EquationList instances representing the nth-order equations are recursively scanned to reduce their order, and the above substitutions are

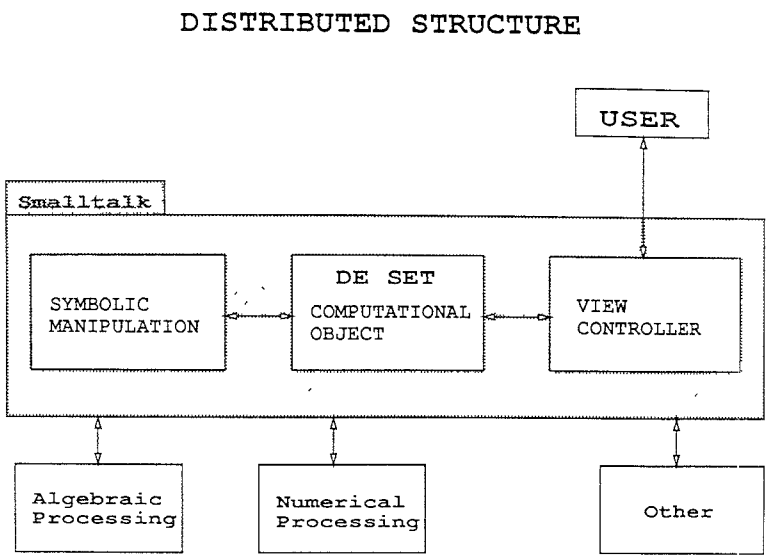


Figure 3

applied, producing N first-order equations. The FDEs are then produced by performing the following substitutions from another Smalltalk Dictionary instance:

```
 u → 0.5 * ((u at:(p + 1)) + (u at.p))
ux → (1 / deltaX) * ((u at:(p + 1)) − (u at:p))
 a → 0.5 * ((a at:(p + 1)) + (a at:p))
ax → (1 / deltaX) * ((a at:(p + 1)) − (a at:p))
   ⋮
```

The resulting set of difference equations is linearized and written in the form given by Eq. 4. To effect this process, the system utilizes an 'expert' algebraic manipulation tool, e.g., Maple [2], residing on a remote machine to expand the resulting equations into their Taylor series and produce the matrix blocks. A back-end program is then produced and sent to a back-end machine where it is invoked via distributed computing. The resulting solution vector is then downloaded back and graphically displayed in the Smalltalk environment.

Figure 3 shows the logical organization and the distributed structure of the DE tool. As shown, the control structure resides in the Smalltalk environment, with communication channels to back-end algebraic and numerical processors. Other channels can be opened as needed. The Model-View-Controller (MVC) paradigm in Smalltalk allows the user to open a view on the computational object (in this case the DE problem) and interact with the controller to perform the visualization and steering tasks, using graphical tools available in the SCENE system [5].

Figure 4 shows the logical organization of an instance of the Smalltalk CoupledDE class, the main class used in the DE tool. The instance variables

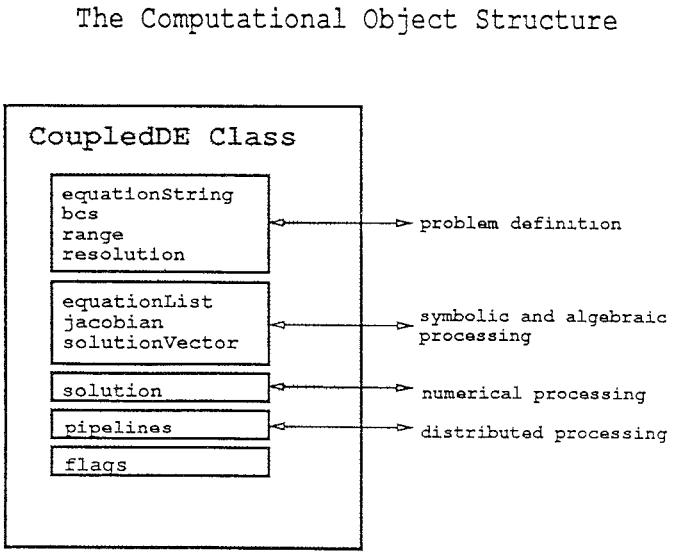


Figure 4

are grouped into five different utility groups. The first set of variables is used locally for defining the problem. This includes the equation string, the boundary conditions, the function domain, and the discretization resolution. The next utility group is used in the symbolic and algebraic processing part of the process and holds the symbolic equation list and the jacobian and the symbolic solution vector variables. These variables, along with the numerical solution variable, are communicated to the back-end machines through the pipeline and socket variables. Finally, some flag variables are needed for synchronization.

4. THE PROTOTYPING ENVIRONMENT

The object-oriented Smalltalk environment allows the DE tool to provide the user with several features for prototyping and for computational steering of the solution. These are provided as menu options. The user can alter parameters, boundary conditions, and even change the equations during the solution process. Some of the most important menu options are:

newProb allows the user to delete the current problem, if any, and to start a completely new problem definition.

changeBCs allows the user to change the number and values of the boundary conditions. The number of boundary conditions has to be consistent with the order of the problem.

changeEquation the whole equation, or any number of parameters can be changed through this option. This is a very impor-

	tant feature for computational steering, as previously mentioned. The user is also given the option of keeping the last solution obtained as an initial guess for the new problem.
changeDomain	allows the user to change the problem domain.
changeResolution	allows the user to change the discretization resolution.
remoteSolve	sends the command to solve the numerical problem on the remote machine. The user is asked to supply a tolerance for convergence. As a safety check, the user is informed whenever the number of iterations exceeds a prespecified number, and a course of action is requested.
remoteIterate	performs a number of iteration steps towards the solution. This option can be invoked when the problem statement (resolution, boundary conditions, parameters) can cause the solution not to converge. By performing a limited number of iterations, the tendency for oscillation can be detected, and the problem statement corrected accordingly.
selectSolution	will display the selected solution.
showAll	will display all the solutions simultaneously.
phasePlot	displays a phase plot of any two chosen solutions.

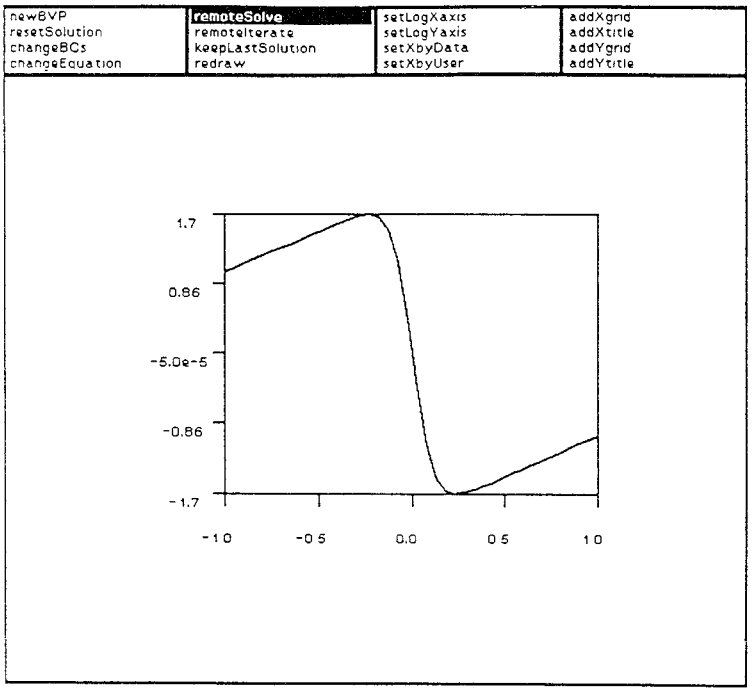
The environment also allows two types of error handling. In the first type, computational errors are intercepted, and the user is informed with an appropriate message. Any of the variables can then be examined and the problem reformulated accordingly. In the second type, a user-controlled cap is placed on the number of relaxation iterations to be performed. If this number is reached, as in the case of oscillations or nonconvergence, the user is informed, and the latest state of the solution is displayed. The solution process can then be resumed, redirected, or restarted with a revised initial guess.

5. RESULTS

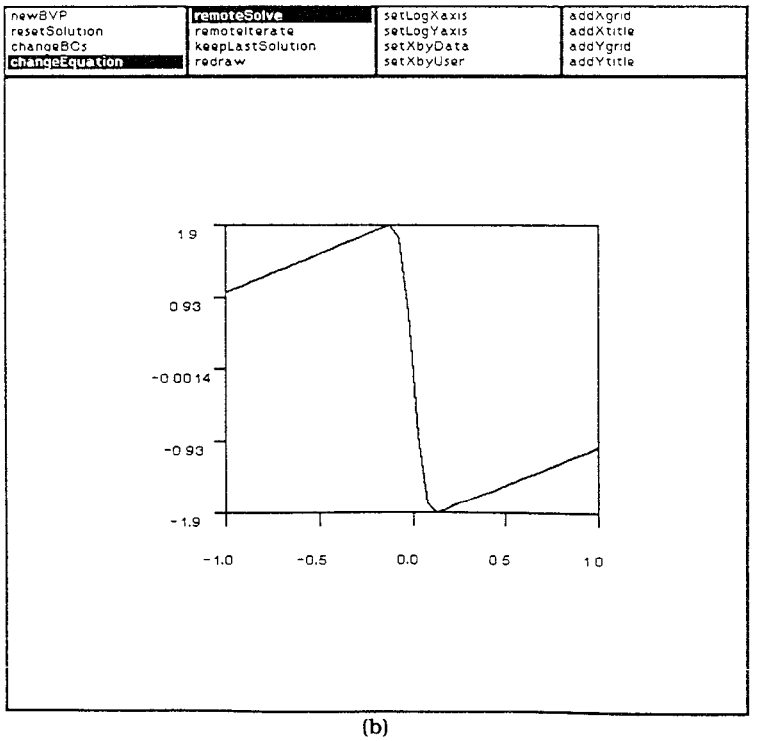
The DE tool was tested on a number of boundary-, initial- and final-value problems with good results. The use of a relaxation method lessens the problem of Gibbs' phenomena associated with resolving sharp shock-type problems. For example, the following singularly perturbed problem:

equation: $0.1 * uxx - (u * ux) + u = 0$
BC's: $u(-1) = 1;\ u(1) = -1$
resolution: 40 points

was tested on the system. The problem was solved and graphical results presented in approximately 17 seconds (using Pro-Matlab [7] running on an Ardent Titan I as the back-end processor), using 7 relaxation iterations. As is evident from Figure 5a, no Gibbs' oscillation is present in the final solution.


newBVP
resetSolution
changeBCs
changeEquation
remoteSolve
remoteIterate
keepLastSolution
redraw
setLogXaxis
setLogYaxis
setXbyData
setXbyUser
addXgrid
addXtitle
addYgrid
addYtitle
1.7
0.86
-5.0e-5
-0.86
-1.7
-1.0
-0.5
0.0
0.5
1.0

(a)


newBVP
resetSolution
changeBCs
changeEquation
remoteSolve
remoteIterate
keepLastSolution
redraw
setLogXaxis
setLogYaxis
setXbyData
setXbyUser
addXgrid
addXtitle
addYgrid
addYtitle
1.9
0.93
-0.0014
-0.93
-1.9
-1.0
-0.5
0.0
0.5
1.0

(b)

Figure 5

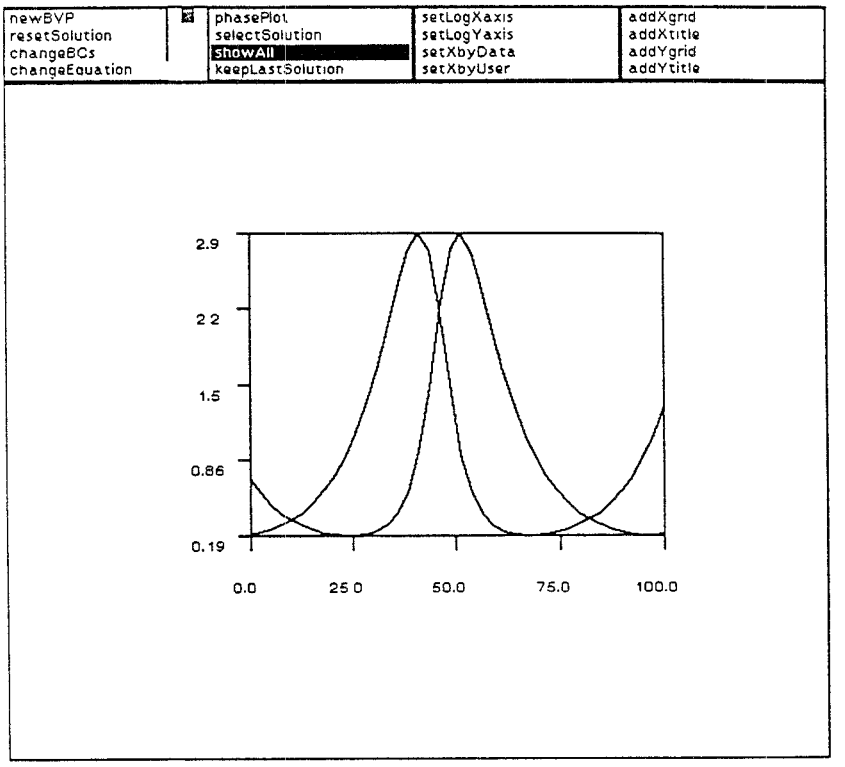


Figure 6

In addition, the shock region is properly represented. As an example of the use of some of the option, the **changeEquation** option was used to change the high-order parameter from 0.1 to 0.05, and a new solution is now obtained in only 13 seconds and 6 additional iterations, by taking the previous solution as initial guess for the new problem (Figure 5b).

Another example tried is the predator-prey problem given by

equation set: $ux - 0.1 * u + 0.1 * u * v = 0$
$vx + 0.1 * v - 0.1 * u * v = 0$
BC's: $u(0) = 0.2;\ v(0) = 0.7$
resolution: 40 points

A plot of the two solutions in given in Figure 6. The problem was solved in 105 seconds, necessitating 18 iterations.

Similarly, good results were obtained for other nonlinear shock- and smooth-type DEs, giving us full confidence in the results obtained during prototyping.

It is important to note that, although the relaxation method just described uses a uniform grid on the solution domain, the DE tool includes the necessary structures for nonuniform gridding. All the functions are instances of the TwoPointFunction class, which incorporates domain knowledge, such

as variable mesh density information. The domain knowledge will be used by a proposed extension [8], which will provide initial guess and mesh density data through nonlinear singular perturbation analysis. In addition, the variable gridding structures will be useful in other adaptive gridding techniques.

6. CONCLUSIONS

This paper presents a DE solution environment for prototyping and computational steering. This tool has proved to be robust and correct in solving a number of difficult DE systems, in particular, a singularly perturbed, nonlinear ordinary differential equation. The tool allows both visualization and computational steering of solutions for DE problems.

Several extensions to the system are being implemented. As mentioned in the previous section, an expert system is also being incorporated to implement domain decomposition and high-order initial estimates for singular perturbation problems [8]. Since convergence in the vicinity of a root is quadratic, a good initial guess will result in faster convergence and might prevent oscillations and inaccurate results [6]. A plot of the error will also be provided, to within the user-provided tolerance value. In addition, the system has recently been modified to handle first-order time PDEs and will be upgraded to nth-order time PDEs.

REFERENCES

1. CAMPBELL, J. R., AND MCGAVRAN, L. P. An integrated distributed processing interface for supercomputers and workstations. Submitted to *ASE 89, Applications of Supercomputers in Engineering* (Southhampton Univ., UK, Sept 5-7, 1989).
2. CHAR, B W., GEDDES, K. O., GONNET, G H , MONAGAN, M. B , AND WATT, S M *MAPLE Reference Manual.* Univ of Waterloo, WATCOM Publications Ltd., Waterloo, Canada, 1988
3. MOLER, C., LITTLE, J., AND BANGERT, S. *Pro-Matlab Users' Guide.* The MathWorks, Inc , 1987.
4. PESKIN, R. L., WALTHER, S. S , AND FRONCIONI, A. M. Smalltalk—The next generation scientific computing interface?. *Math. Comput. Simul. 31*, (1989), 371-381.
5. PESKIN, R. L., WALTHER, S. S., FRONCIONI, A. M., AND BOUBEZ, T. I. Incremental visualization as a strategy for computational steering. *IBM J. Res. Dev.* To be published, 1992.
6. PRESS, W H., FLANNERY, B. P , TEUKOLSKY, S A., AND VETTERLING, W. T. *Numerical Recipes.* Cambridge University Press, 1988.
7. RUSSO, M. F. Automatic generation of parallel programs using nonlinear singular perturbation theory. Ph D. Thesis, Rutgers Univ , 1989
8. RUSSO, M F., AND PESKIN, R. L. Automatically identifying the asymptotic behavior of nonlinear singularly-perturbed boundary-value problems *J Autom. Reasoning* To appear.

Received March 1990; revised February 1991; accepted July 1991