1990

An Interactive Symbolic-Numeric Interface to Parallel ELLPACK for Building General PDE Solvers

S. Weerawarana

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
90-1054

AN INTERACTIVE SYMBOLIC-NUMERIC INTERFACE TO PARALLEL ELLPACK FOR BUILDING GENERAL PDE SOLVERS

S. Weerawarana
E. N. Houstis
John R. Rice

CSD-TR-1054
December 1990

An Interactive Symbolic–Numeric Interface to Parallel ELLPACK for Building General PDE Solvers

S. Weerawarana, E. N. Houstis, and J. R. Rice *
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907.

November 29, 1990

Abstract

In this paper we describe an interactive symbolic-numeric interface framework (editor) to the ELLPACK partial differential equation (PDE) system for building PDE solvers for a much broader range of applications. The domain of applicability of ELLPACK and its parallel version (//ELLPACK) is restricted to second order linear elliptic boundary value problems. This editor allows the specification of nonlinear initial and boundary value PDE problems. The editor applies hybrid symbolic–numeric techniques at the PDE problem level to automatically reduce them to a sequence of linear elliptic PDEs. The result of this preprocessing is recorded in the form of an ELLPACK program. Several examples are presented to demonstrate the functionality and applicability of this interface framework, and the efficiency of the underlying solution methods.

1 Introduction

ELLPACK is a high level software environment for specifying linear elliptic second order boundary value problems and their solvers. The solvers are built out of an extensive library of modules that correspond to the various phases of the numerical processes for these types of PDE problems. A detailed description of the ELLPACK system for sequential machines is given in [Rice 85]. The system is currently being extended to accommodate PDE problems which may be nonlinear, second order in space and parabolic or hyperbolic in time. The extension also includes facilities to specify or select pairs of parallel algorithms

*This research was supported in part by AFOSR 88-0234, ARO grant DAAG29-83-K-0026, NSF grant CCF-8619817 and ESPRIT project GENESIS.

and architectures. We use an object-oriented knowledge framework (editor) interface which allow the user to specify:

- linear/nonlinear PDE boundary/initial value problems in a natural form,
- the PDE domain geometry and boundary conditions in a graphical and textual form,
- parameters such as grid and the mapping of the underlying computation to the specified machine,
- linear elliptic PDE solvers using a menu for the many choices of individual components,
- displays of performance data collected,
- visualization of the computed solution,
- displays showing the efficiency of a method with respect to a known data base of performance data.

Figure 1 shows the structure of a framework and its corresponding editor. A general discussion of the object–oriented knowledge framework methodology is presented in [Forbe 89]. Figure 2 shows part of the hierarchy of these frameworks for the //ELLPACK application, while their complete description is given in [Houstis 90b, Houstis 90a]. The remaining editors of the interface are those invoked after execution to display solutions, performance data, etc.

The output of this object–oriented software environment is a control program written in the very high level //ELLPACK language which in turn is translated into a Fortran control program. Even though the //ELLPACK control program is in a very high level language, it is quite long in typical applications, of the order of several hundred lines. The resulting Fortran control program is much longer still. See [Houstis 90c] for a complete example of //ELLPACK control program as well as a discussion of its syntax and use.

This paper is concerned with the development of the PDE specification framework which uses symbolic/numeric processing to handle the PDE problem extensions to determine some of the input functions and carry out any preprocessing required. The methodology used to handle nonlinear and time–dependent terms is presented in Section 2. The functionality of this framework and its implementation are discussed in Section 3. Finally in Section 4 we present numerical results which indicate the efficiency of the high level PDE solvers created.

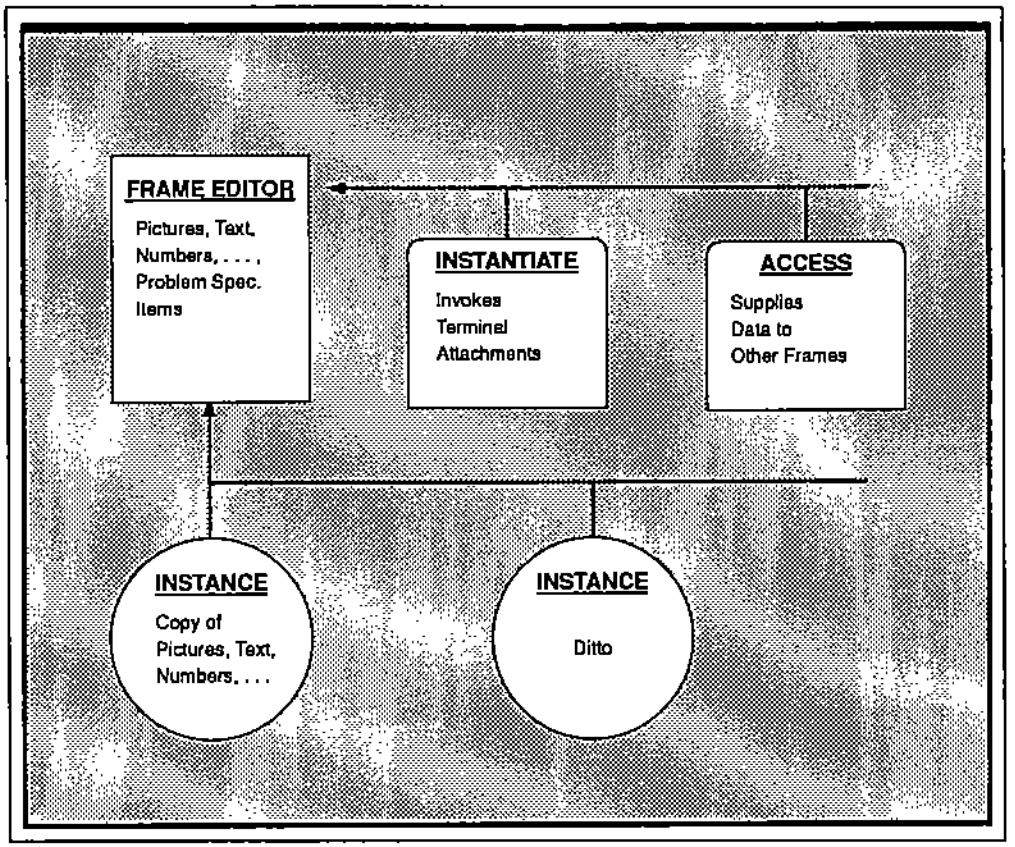


Figure 1: Structure of an Object–oriented knowledge framework in the interface.

2 High Level Nonlinear and Time-dependent PDE Solvers

In this section we formulate a high level preprocessing methodology for solving nonlinear initial/boundary value problems of the form

$$\alpha\, u_t + \beta\, u_{tt} = F(t, x, y, z, u, u_x, u_y, u_z, u_{xx}, u_{yy}, u_{zz}, u_{xy}, u_{xz}, u_{yz}) \equiv Fu \tag{2.1}$$

including, for example, in a simpler case,

$$\begin{aligned} \alpha\, u_t + \beta\, u_{tt} &= c_0 + c_1\, u + c_2\, u_x + c_3\, u_y + c_4\, u_z + c_5\, u_{xx} + c_6\, u_{yy} + c_7\, u_{zz} + \\ &\quad c_8\, u_{xy} + c_9\, u_{xz} + c_{10}\, u_{yz} \\ &\equiv Lu \end{aligned} \tag{2.2}$$

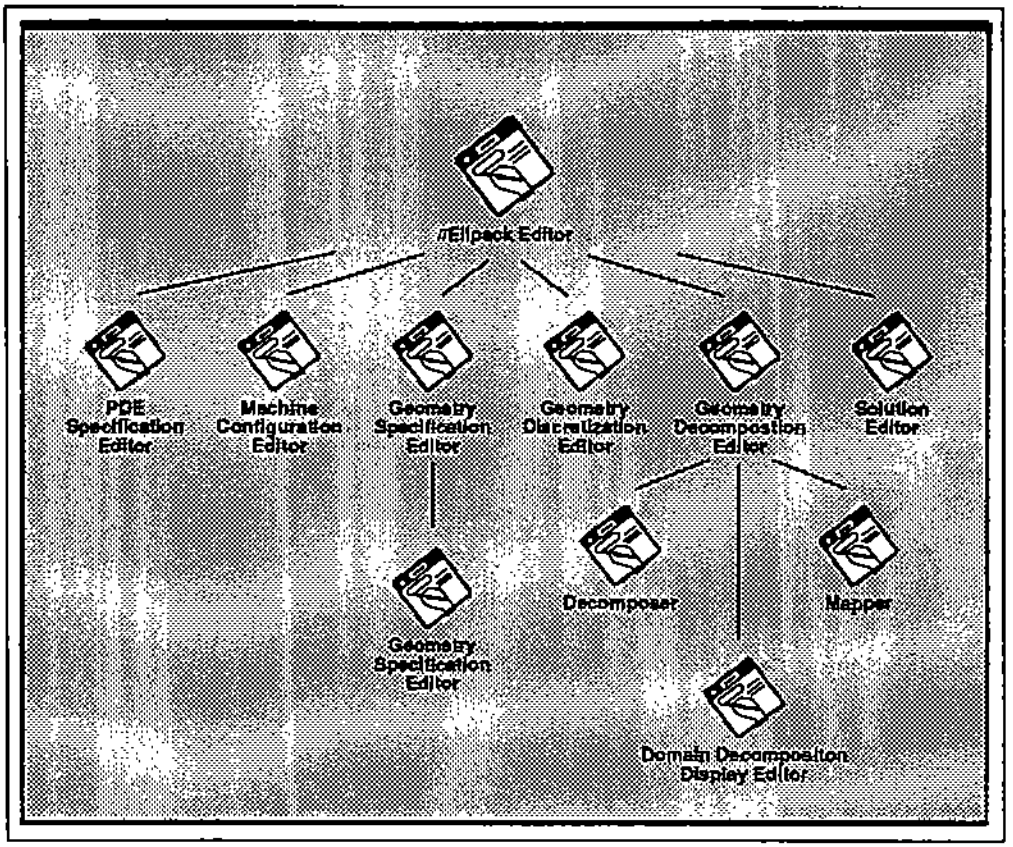


Figure 2: Hierarchy of editors in the //ELLPACK Preprocessing/Solution Interface.

Problems (2.1) and (2.2) are defined on $(0,T]\times\Omega$ with $\Omega \subset \Re^3$ and are subject to boundary conditions

$$Gu \equiv G(t,x,y,z,u,u_x,u_y,u_z) = \psi(t) \tag{2.3}$$

or, again in the simpler case,

$$Bu \equiv d_0 + d_1\, u + d_2\, u_x + d_3\, u_y = 0 \tag{2.4}$$

on the boundary of Ω, and initial conditions at $t = 0$

$$u(0,x,y,z) = \phi(x,y,z). \tag{2.5}$$

The coefficients c_i and d_i may depend on (x,y,z) and the problem remains in the class of linear PDEs that //ELLPACK currently assumes. The coefficients of L and B could be

functions of the solution u, making a semi–linear problem not in the class that //ELLPACK currently assumes. The parameters α and β are chosen to make the equation (2.1) elliptic ($\alpha = 0, \beta = 0$), parabolic ($\alpha = 1, \beta = 0$) and hyperbolic ($\alpha = 0, \beta = 1$).

2.1 Nonlinear PDE solvers

The traditional way to handle nonlinear PDE problems numerically is to discretize them first with an appropriate method and then solve the corresponding nonlinear algebraic equations. A non–traditional alternative is to apply a nonlinear methodology directly on the continuous PDE equations and boundary conditions. This approach reduces the original nonlinear PDE problem to a sequence of linear PDEs which must be solved by iteration starting with some initial guess. Under certain general assumptions we can show that the two approaches are equivalent. Another advantage of the second approach is the ability to utilize existing linear PDE solvers without any modification. Its disadvantage is that the user has to define and implement the nonlinear process at the program control (PDE operator) level and carry out the symbolic processing required. Rice has tested this methodology in [Rice 83, Rice 85] by constructing special ELLPACK templates which implement nonlinear and time dependent high level solvers by embedding ELLPACK code into Fortran. In order to separate the user from the PDE problem specification and PDE preprocessing/solution phases, we have developed a knowledge based editor which automatically generates these templates from given specifications and carries out the needed symbolic processing. The //ELLPACK system already has a natural interface to MAXIMA and //ELLPACK and now implements two of the most often used nonlinear approaches, the Picard and Newton's iterations. Their formulation at the PDE problem level has appeared in [Rice 83a, 83b]. For completeness, we present it here.

2.1.1 Picard iteration for nonlinear problems

In order to apply Picard iteration, one rewrites the equations $F(u) = 0$ and $G(u) = 0$ into the form $L(u)u = f(u)$, and $B(u)u = g(u)$ and then uses the iteration

$$L(u_k)\, u_{k+1} = f(u_k),$$

$$B(u_k)\, u_{k+1} = g(u_k).$$

Note that $L(u)\, u$ is the operator $L(u)$ applied to the function u and not an ordinary multiplication. The coefficients of L and the function f can be nonlinear functions of the partial derivatives of u. The coefficients of the boundary operator B and function g can depend on u. The attractiveness of this method is its simplicity. Its weakness is that convergence can not be predicted a priori. Picard's nonlinear process can be described as follows:

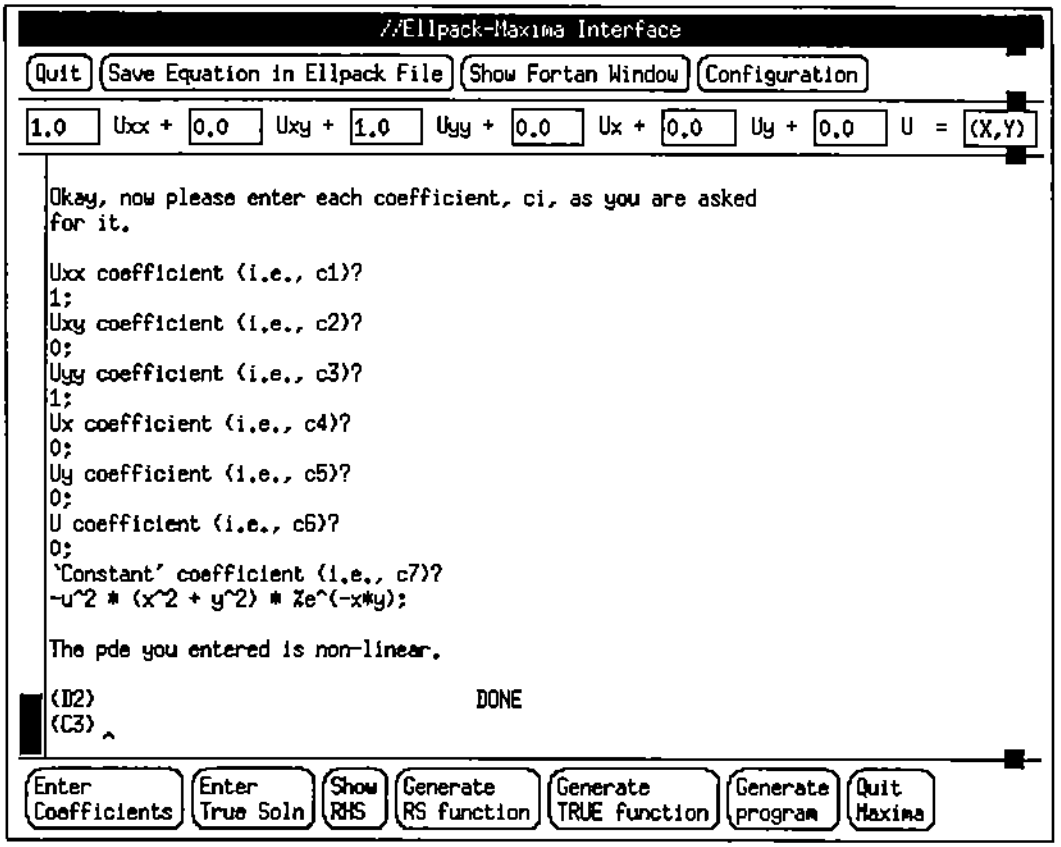


Figure 3: A semi–linear PDE problem specified in the PDE specification editor. This display occurs in the //ELLPACK–MAXIMA part of this editor.

Guess $u(0)$
repeat
    Solve $L(u_k)u_{k+1} = f(u_k)$, and $B(u_k)u_{k+1} = g(u_k)$
    Set $u_{k+1} := u_k$
until converged.

Next we consider some examples to demonstrate the formulation of this method in the //ELLPACK environment. Figure 3 shows the PDE specification editor for the problem

$$u_{xx} + u_{yy} = u^2(x^2 + y^2)e^{-xy} \tag{2.6}$$

```
OPTIONS.
     l1time = .true.
     clockwise = .true.
     xplot3d

DECLARATIONS.
     parameter (tol=0.01)
     parameter (niters=10.0)

EQUATION.
     UXX+UYY = U(X,Y)""2*(X""2+Y""2)*EXP(-X*Y)

BOUNDARY.
 enter a boundary here.

GRID.
 enter a grid here.

TRIPLE.
     set (u = zero)

FORTRAN. +both.
     I1LEVL = 1
     do 20 I = 1,niters

DISCRETIZATION.
 enter discretization method here.

INDEXING.
 enter indexing method here.

SOLUTION.
 enter solution method here.

OUTPUT.
     MAX (ERROR)

FORTRAN. +both.
 test for convergence:
        if (R1NRM2 .lt. tol) then
        go to 30
       endif
       I1LEVL = 0
  20  continue
     print *, 'failed to converge!'
     go to 50
  30  continue
     print *, 'converged in ', I, ' iterations'
  50  continue

END.
```

Figure 4: Template of a //ELLPACK program for Picard's method for (2.6), (2.7). Later stage of the solution process may add greatly to this program.

defined on the unit square subject to boundary conditions

$$u = u^2 + g \tag{2.7}$$

where g is a function that makes $u = e^{xy}$ the true solution of the boundary value problem. The Picard iteration template generated by this interface editor is shown in Figure 4. In spite of the simplicity of Picard's method, it is not always very effective. The number of required iterations for convergence can be very large.

2.1.2 The Newton's iteration for solving nonlinear problems

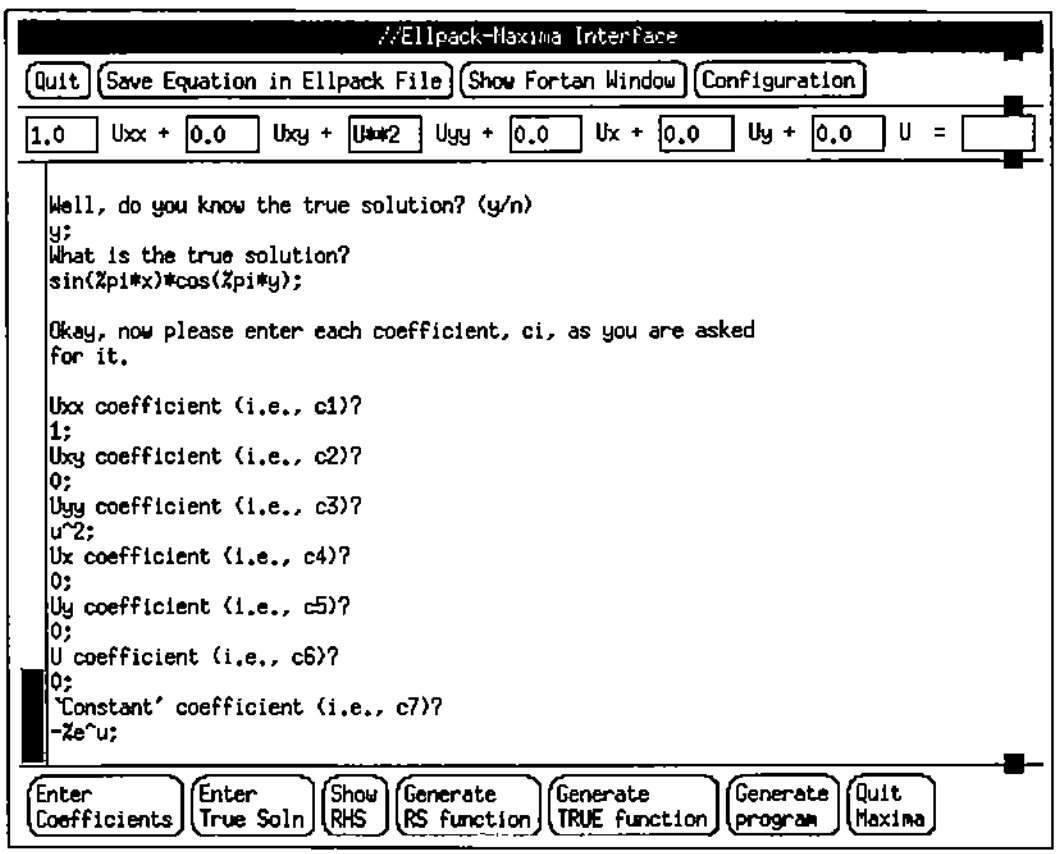
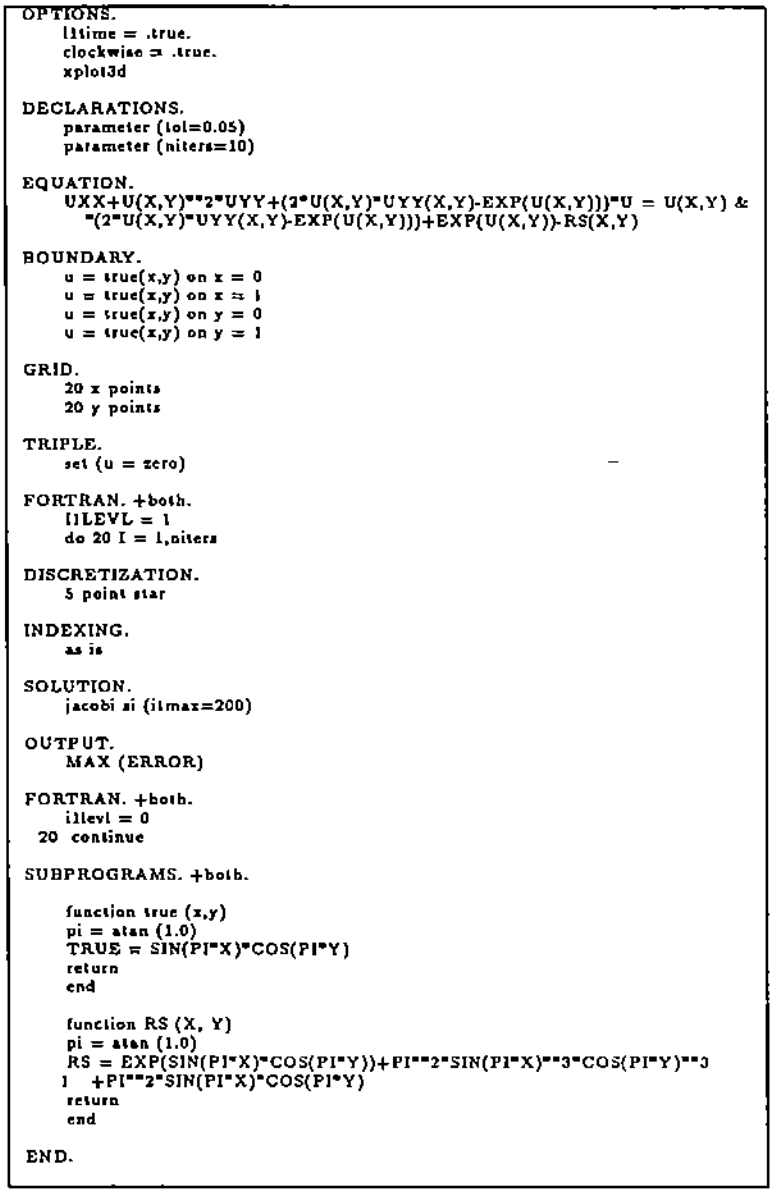


Figure 5: The nonlinear problem (2.8) in the PDE specification editor in preparation for applying Newton's method.



```
OPTIONS.
     [1time = .true.
     clockwise = .true.
     xplot3d

DECLARATIONS.
     parameter (tol=0.05)
     parameter (niters=10)

EQUATION.
     UXX+U(X,Y)**2*UYY+(2*U(X,Y)*UYY(X,Y)-EXP(U(X,Y)))*U = U(X,Y) &
       *(2*U(X,Y)*UYY(X,Y)-EXP(U(X,Y)))+EXP(U(X,Y))-RS(X,Y)

BOUNDARY.
     u = true(x,y) on x = 0
     u = true(x,y) on x = 1
     u = true(x,y) on y = 0
     u = true(x,y) on y = 1

GRID.
     20 x points
     20 y points

TRIPLE.
     set (u = zero)

FORTRAN. +both.
     [1LEVL = 1
     do 20 I = 1,niters

DISCRETIZATION.
     5 point star

INDEXING.
     as is

SOLUTION.
     jacobi si (itmax=200)

OUTPUT.
     MAX (ERROR)

FORTRAN. +both.
     illevl = 0
  20 continue

SUBPROGRAMS. +both.

     function true (x,y)
     pi = atan (1.0)
     TRUE = SIN(PI*X)*COS(PI*Y)
     return
     end

     function RS (X, Y)
     pi = atan (1.0)
     RS = EXP(SIN(PI*X)*COS(PI*Y))+PI**2*SIN(PI*X)**3*COS(PI*Y)**3
    1   +PI**2*SIN(PI*X)*COS(PI*Y)
     return
     end

END.
```

Figure 6: Template of a //ELLPACK program for Newton's method applied to (2.8). Internally, the system has created the linearized PDE problem which is solved repeatedly in the Fortran DO-loop 20.

A superior alternative to Picard iteration is the well known Newton's method. For its formulation we consider the general PDE problem (2.1), (2.3) with $\alpha = 0 = \beta$. The idea of the method is to approximate $F(u) = 0$ and $G(u) = 0$ with their linear counter parts

$$\begin{aligned} F(u_0) + F'(u_0)(u_1 - u_0) &= 0 \\ G(u_0) + G'(u_0)(u_1 - u_0) &= 0 \end{aligned}$$

and then iteratively solve these linear problems. The linear counterparts are the Fréchet derivatives of the operators F and G with respect the the function u and its derivatives. While the mathematical foundations of such differentiation is complex, its mechanics are similar to ordinary differentiation. The corresponding symbolic/numeric process that implements Newton's method can be described as follows:

Compute Fre'chet derivatives $L(u), B(u)$ of $F(u)$ and $G(u)$
repeat
    Solve $L(u_0)u = -(F(u_0) - L(u_0)u_0)$, and $B(u_0)u = -(G(u_0) - B(u_0)u_0)$
    Set $u_0 := u$
until converged.

To illustrate the application of this method consider the problem

$$F(u) = u_{xx} + u^2\, u_{yy} - e^u - f = 0 \tag{2.8}$$

defined on the unit square and subject to boundary conditions

$$G(u) = u + u^2\, u_y - g = 0 \tag{2.9}$$

on $x = 0$ and

$$G(u) = u - g' = 0 \tag{2.10}$$

on the rest of the boundary. The functions f, g, and g' are selected such that $u(x, y) = sin(x)\, cos(y)$ is the true solution. Figure 5 displays this PDE problem in the PDE specification editor and Figure 6 shows the template generated for Newton's method from the PDE specification framework.

2.2 Time semi–discretization solvers

In this section we consider a semi-discretization procedure for solving initial/boundary value problems. The procedure can be viewed as opposite to method of lines, since the time discretization is done first and the original problem is reduced to a sequence of linear

or nonlinear time–independent PDEs on the various time levels. In the case of a parabolic PDE ($\alpha = 1, \beta = 0$) and Crank–Nicholson discretization, equation (2.1) is reduced to

$$u(t) - u(t - \Delta t) = \frac{\Delta t}{2} \left\{ F(u) \mid_{u=u(t)} + F(u) \mid_{u=u(t-\Delta t)} \right\}. \tag{2.11}$$

Note that we have suppressed the space variables and derivatives of u in the above equation. Assuming that the solution and its derivatives are known at the $t - \Delta t$ level, then the nonlinear PDE with respect to $u(t)$ is solved over the domain Ω with boundary conditions

$$G(u(t)) = \psi(t). \tag{2.12}$$

To illustrate this approach, we consider the equation

$$u_t = \nu \frac{(\phi(u)u_x)_x}{u^n} + \nu \frac{(\phi(u)u_y)_y}{u^n} - f(u)u_x - g(u)u_y + h \tag{2.13}$$

defined on $(0,T]\times$(unit square) subject to Dirichlet boundary conditions

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

on the boundary of Ω and initial conditions $u = true$.

The equation is parametrized with respect to ν, ϕ, n, f, and g. The function h is chosen so that a given function $true(t,x,y)$ is the solution of the PDE problem. Appendix A shows the template for solving (2.13) with $\nu = 1$, $\phi(u) = e^{-u}$, $n = 2$, $f = g = u^2$ and $true(t,x,y) = t + x + y$.

3 A Symbolic / Numeric Interface for nonlinear / time-dependent PDEs

We have developed an interface to the symbolic computing system MAXIMA[1] [MACSYMA 77] to implement some of the transformations noted above. We briefly describe the user interface to the PDE specification editor, PDE manipulation methodologies and their specification, and the interface between //ELLPACK and MAXIMA. The PDE specification editor is the software that deals with obtaining the PDE operator for the //ELLPACK programming environment.

[1] MAXIMA is the AKCL version of Macsyma.

3.1 User Specification of the PDE Operator

The PDE specification editor consists of a window providing access to MAXIMA, an equation display/edit panel above it, and a button panel below it. There are several additional buttons above the equation display panel which are used for configuration, quitting and saving, see Figure 3. The control panel below the MAXIMA window consists of buttons to enter the PDE operator, to generate the //ELLPACK program and buttons to allow the user to enter corrected information and obtain partial results. For example, the *show right hand side function* button allows the user to view the correction term added to force a certain solution. Also, Fortran code for the PDE coefficient functions may be viewed.

When the system prompts for input, the user is expected to provide information in legal MAXIMA syntax via the MAXIMA access window. We are in the process of adding an *input wrapper* for MAXIMA which will allow one to enter expressions in a more 'natural' form and not deal with any syntax peculiarities of MAXIMA.

The PDE operator is specified by either entering the coefficients of the operator L or by defining an implicit relationship between them. If the PDE is specified by entering coefficients, one starts by pressing the *enter coefficients* button. The MAXIMA function which implements this action prompts the user for each coefficient of the PDE operator and saves them in internal variables for later use by other functions. It also sends these expressions to the equation display/edit panel for display. The user may edit the display panel directly as well.

Given the PDE operator, the system identifies the type of the operator (i.e., time–dependent, parabolic, hyperbolic, nonlinear) and sets appropriate global flags. We also provide a convenient method to force a given function to be the true solution. For example, if we wanted to check whether the linearization process is working correctly, we could request that the PDE operator be perturbed to force a certain solution. Then, by visualizing the computed solution using //ELLPACK's solution visualization editor, we can verify correctness of the solution process.

3.2 Linearization and Time Discretization Methodology

Once the operator has been specified, a //ELLPACK program can be generated to solve the linearized elliptic PDE. This is done by pressing the *generate program* button. If the operator is found to be nonlinear, then it is linearized in the manner outlined in Section 2.1.2 using MAXIMA's symbolic differentiation capability. If the operator is time–dependent, then the time derivatives are discretized using one of several methods (for example, Crank–Nicholson discretization).

For nonlinear PDEs, several parameters control the solution process and the user is asked to enter values for these during the //ELLPACK program generation stage. We

currently require the user to specify a tolerance (to be used to check whether a satisfactory solution has been found), the maximum number of iterations to perform before aborting, and finally which norm to use to check for convergence. We plan to integrate knowledge–based assistance into this editor to automatically provide values for these parameters based on experience and other empirical data, if the user so desires.

For time–dependent PDEs, we allow the user to select one of several time discretization methods although it is always possible to define any discretization scheme using directly MAXIMA's symbolic manipulation capabilities. As //ELLPACK's data structures only provide convenient mechanisms for 2–stage time–discretizations, we currently limit the user to them. We are examining convenient implementation methods for k–stage techniques also. The other parameters that are involved with the time–discretization stage are the starting time (t_0), the solution u_0 at time t_0, the ending time (t_{end}), and the time step. As with nonlinear PDEs, we expect to integrate knowledge–based assistance to select the time step needed to obtain some requested tolerance.

Once all the necessary parameters are specified, a //ELLPACK program is generated. For nonlinear problems, this program iteratively improves an initial guess until some convergence criteria has been met. For time–dependent problems, the program steps along the time axis solving an elliptic problem at each step. If the operator is both time–dependent and nonlinear, then the outer loop iterates over time while the inner loop iterates to solve the linearized elliptic PDE at each time step. The generated program is a //ELLPACK program which includes all necessary Fortran code. For example, Fortran functions are generated to compute derivatives of the initial conditions.

3.3 //ELLPACK – MAXIMA Interface

MAXIMA is a large, Lisp-based, interactive system that expects the user to type in expressions to be evaluated and printed. MAXIMA can be programmed in a high level language [MACSYMA 77] and in Lisp. In the //ELLPACK environment, MAXIMA runs as a separate process, possibly on a different host machine. The processes are connected via three sockets [Leffler 86]; one to the standard input stream of MAXIMA, one to the standard output and standard error streams, and the other to a special connection to pass messages between MAXIMA and //ELLPACK. The MAXIMA functions we have implemented communicate with //ELLPACK using this third connection in a simple protocol. In the current prototype implementation, the user enters expressions directly to MAXIMA. A button panel allows the user to invoke the needed MAXIMA functions conveniently. For example, to enter the coefficients of the PDE operator, the user presses the *enter coefficients* button. MAXIMA has a simple Fortran code generation capabilities to generate Fortran code for expressions. We have extended this into a rudimentary code generation capability to generate the Fortran code we need (including loops, control structures and

subprograms), using "print" statements. Since this technique does not allow one to manipulate the generated code, we will be integrating an automatic code generation system, GENCRAY [Weerawarana 89], to provide a much more flexible code generation capability.

4 Numerical Examples

In this section we present the solutions of the PDE equations (2.6), (2.8), and (2.13) with Dirichlet boundary conditions. Tables 1 and 2 indicate the convergence of the Picard and Newton's iteration methods for equations (2.6) and (2.8), respectively. In Table 3 we present the space discretization error at several time levels after 2,4, and 6 Newton iterations. In all examples we used five–point star to discretize the linearized PDE equations in space and Jacobi–SI iteration to solve the resulting linear equations.

iterations	$\|error\|_\infty$
1	6.983E-02
2	5.133E-03
3	2.285E-04
4	1.506E-04
5	1.301E-04
10	1.320E-04

Table 1: The convergence of Picard's method for PDE equation (2.6) defined on the unit square with Dirichlet boundary conditions and true solution $u(x, y) = e^{xy}$.

iterations	$\|error\|_\infty$
1	7.341E-01
2	5.255E-01
3	1.011E-01
4	1.462E-02
5	7.782E-04
6	7.079E-04
10	7.073E-04

Table 2: The convergence of Newton's method for PDE equation (2.8) defined on the unit square with Dirichlet boundary conditions and true solution $u(x,y) = sin(\pi\, x)\, cos(\pi\, y)$.

time level	$\|error\|_\infty$ after k Newton iterations			
	k = 1	k = 2	k = 3	k = 5
Δt	3.576E-07	5.960E-07	4.768E-07	4.172E07
$2\Delta t$	4.172E-07	5.960E-07	5.960E-07	4.768E-07
$3\Delta t$	5.960E-07	5.960E-07	5.960E-07	5.960E-07
$5\Delta t$	5.960E-07	7.152E-07	7.152E-07	5.960E-07
$10\Delta t$	9.536E-07	1.192E-06	1.192E-06	1.430E-06
$50\Delta t$	1.192E-05	1.096E-05	1.001E-05	1.049E-05

Table 3: The Crank–Nicholson/5–point star numerical solution of PDE equation (2.13) defined on the unit square with Dirichlet boundary conditions and true solution $u(x,y) = t + x + y$. The computation uses $\Delta x = \Delta y = \frac{1}{10}$, and $\Delta t = 0.1$.

References

[Forbe 89] Forbe, B. W. R., Rusell, A. D., and Stremer, S. F., "Object–oriented Knowledge Frameworks", *Engineering with Computers*, Vol. 5, 1989, pp. 79–89.

[Houstis 90a] Houstis, E. N., Rice, J. R., Chrisochoides, N. P., Karanthanasis, H. C., Papachiou, P. N., Samartzis, M. K., Vavalis, E. A., and Wang, Ko Yang, "Parallel (//) ELLPACK PDE solving system", CAPO technical report CER–89–20, Department of Computer Sciences, Purdue University, October 1989.

[Houstis 90b] Houstis, E. N., Rice, J. R., Chrisochoides, N. P., Karanthanasis, H. C., Papachiou, P. N., Samartzis, M. K., Vavalis, E. A., Wang, Ko Yang, and Weerawarana, S., "//ELLPACK: A Numerical Simulation Programming Environment for Parallel MIMD Machines", *Proceeding of the 1990 International Conference on Supercomputing*, ACM Press, New York, USA, 1990.

[Houstis 90c] Houstis, E. N., Rice, J. R., Chrisochoides, N. P., Kim, S. B., Ku, T., Wang, K. Y., and Weerawarana, S., *//ELLPACK User's Guide*, CSD–TR–1039, Department of Computer Sciences, Purdue University, December 1990.

[Leffler 86] Leffler, Samuel J., et. al., *An Advanced 4.3BSD Interprocess Communication Tutorial*, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1986.

[MACSYMA 77] *MACSYMA Reference Manual*, Version 9, The Mathlab Group, Laboratory for Computer Science, M. I. T., Cambridge, MA, 1977.

[Rice 83] Rice, John R., "Building elliptic problem solvers with ELLPACK", *Elliptic Problem Solvers II*, G. Birkhoff and A. Schoerstadt, editors, Academic Press, 1983, pp. 3–27.

[Rice 85] Rice, John R., and Boisvert, Ronald F., *Solving Elliptic Problems Using ELLPACK*, Springer Series in Computational Mathematics 2, Springer–Verlag, New York, USA, 1985.

[Wang 88] Wang, Paul S., *An Introduction to Berkeley UNIX*, Wadsworth Publishing Company, Belmont, California, 1988.

[Weerawarana 89] Weerawarana, Sanjiva and Wang, Paul S., "GENCRAY: A Portable Code Generator for CRAY Fortran", *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, ACM Press, New York, 1990.

A //ELLPACK program template for solving (2.13)

```
OPTIONS.
      l1time = .true.
      clockwise = .true.
      xplot3d

DECLARATIONS.
      real tol
      integer niters
      common /saveu/ unkn($i1ngrx,$i1ngry)

GLOBAL.
      real t, deltat, alpha, tstart, tstop
      integer nstep, nsteps
      common /timedep/ t,deltat, nstep, alpha

EQUATION.
      ALPHA*EXP(-U(X,Y))/U(X,Y)**2*UXX+ALPHA*EXP(-U(X,Y))/U(X,Y)**2*UYY+ &
         ALPHA*(-2*EXP(-U(X,Y))*UX(X,Y)/U(X,Y)**2-U(X,Y)**2)*UX+ALPHA*(- &
         2*EXP(-U(X,Y))*UY(X,Y)/U(X,Y)**2-U(X,Y)**2)*UY+(ALPHA*(-2*(EXP( &
         -U(X,Y))*UYY(X,Y)-EXP(-U(X,Y))*UY(X,Y)**2)/U(X,Y)**3+(EXP(-U(X, &
         Y))*UY(X,Y)**2-EXP(-U(X,Y))*UYY(X,Y))/U(X,Y)**2-2*U(X,Y)*UY(X,Y &
         )-2*(EXP(-U(X,Y))*UXX(X,Y)-EXP(-U(X,Y))*UX(X,Y)**2)/U(X,Y)**3+( &
         EXP(-U(X,Y))*UX(X,Y)**2-EXP(-U(X,Y))*UXX(X,Y))/U(X,Y)**2-2*U(X, &
         Y)*UX(X,Y))-1/DELTAT)*U = EXP(-U(X,Y))*((-ALPHA*U(X,Y)-2*ALPHA) &
         *UYY(X,Y)+(ALPHA*U(X,Y)+ALPHA)*UY(X,Y)**2-2*ALPHA*U(X,Y)**4*EXP &
         (U(X,Y))*UY(X,Y)+(-ALPHA*U(X,Y)-2*ALPHA)*UXX(X,Y)+(ALPHA*U(X,Y) &
         +ALPHA)*UX(X,Y)**2-2*ALPHA*U(X,Y)**4*EXP(U(X,Y))*UX(X,Y)+PDERS( &
         X,Y)*U(X,Y)**2*EXP(U(X,Y))+ALPHA*FORCE(X,Y)*U(X,Y)**2*EXP(U(X,Y &
         )))/U(X,Y)**2

BOUNDARY.
      u=true(x,y) on x=0
      u=true(x,y) on x=1
      u=true(x,y) on y=0
      u=true(x,y) on y=1

GRID.
```

```
      10 x points
      10 y points

TRIPLE.
      set (u = t0sol)

FORTRAN.
      call save (r1tabl, i1ngrx*i1ngry)

      l1cstc = .false.
      alpha = 0.5
      tstart = 0.0
      tstop = 5.0
      deltat = 0.1
      nsteps = int ((tstop-tstart)/deltat + 0.5)
      deltat = (tstop - tstart)/nsteps

      do 100 nstep = 1, nsteps
      t = tstart + nstep*deltat

TRIPLE.
      set u by blending

FORTRAN.
      niters = 10
      tol = 0.005
      do 300 i = 1, niters
      print *, 'time=',t,' , iteration=', i

DISCRETIZATION.
      5 point star

INDEXING.
      as is

SOLUTION.
      band ge

OUTPUT.
```

```
      max (error)

FORTRAN.
      i1levl = 0

*     test for convergence
      if (R1NRMI .lt. tol) then
        go to 301
      endif
 300  continue
      print *, 'failed to converge!'
      go to 302
 301  continue
      print *, 'converged in ', i, ' iterations.'
 302  continue

FORTRAN.
      call q35pvl
      call save (r1tabl, i1ngrx*i1ngry)
 100  continue

SUBPROGRAMS. +both.

      subroutine save (arr, len)
      real arr(1)
      common /saveu/ unkn($i1ngrx*$i1ngry)
      do 101 i = 1,len
        unkn(i) = arr(i)
 101  continue
      return
      end

      function PDERS(x,y)
      real t, deltat, alpha, tstart, tstop
      external u1
      integer nstep, nsteps
      common /timedep/ t,deltat, nstep, alpha
      t = t - deltat
      if (nstep .eq. 1) then
```

```
   PDERS = (ALPHA-1.0)*(-U0(X,Y,5)*U0(X,Y,6)**2-U0(X,Y,4)*U0(X,Y,6)
1     **2+(U0(X,Y,3)*EXP(-U0(X,Y,6))-U0(X,Y,5)**2*EXP(-U0(X,Y,6)))/
2     U0(X,Y,6)**2+(U0(X,Y,1)*EXP(-U0(X,Y,6))-U0(X,Y,4)**2*EXP(-U0(
3     X,Y,6)))/U0(X,Y,6)**2-FORCE(X,Y))-U0(X,Y,6)/DELTAT
 else
   PDERS = (ALPHA-1.0)*((EXP(-U1(X,Y,6))*U1(X,Y,3)-EXP(-U1(X,Y,6))*
1     U1(X,Y,5)**2)/U1(X,Y,6)**2-U1(X,Y,6)**2*U1(X,Y,5)+(EXP(-U1(X,
2     Y,6))*U1(X,Y,1)-EXP(-U1(X,Y,6))*U1(X,Y,4)**2)/U1(X,Y,6)**2-U1
3     (X,Y,6)**2*U1(X,Y,4)-FORCE(X,Y))-U1(X,Y,6)/DELTAT
 endif
 t = t + deltat
 return
 end

 function U0(x,y,ideriv)
 real t, deltat, alpha, tstart, tstop
 integer nstep, nsteps
 common /timedep/ t,deltat, nstep, alpha
 if (ideriv .eq. 1) then
   U0 = 0
 else if (ideriv .eq. 2) then
   U0 = 0
 else if (ideriv .eq. 3) then
   U0 = 0
 else if (ideriv .eq. 4) then
   U0 = 1
 else if (ideriv .eq. 5) then
   U0 = 1
 else if (ideriv .eq. 6) then
   U0 = Y+X
 endif
 return
 end

 function TOSOL (x,y)
 TOSOL = U0(X,Y,6)
 return
 end
```

```
      function u1(x,y,ideriv)
      common /saveu/unkn($i1ngrx*$i1ngry)
      u1 = r1qd2i(x, y, unkn, ideriv)
      return
      end

      function FORCE(x,y)
      real tol
      integer niters
      real t, deltat, alpha, tstart, tstop
      integer nstep, nsteps
      common /timedep/ t,deltat, nstep, alpha
      FORCE = -EXP(-Y-X-T)*(2*Y**4*EXP(Y+X+T)+8*X*Y**3*EXP(Y+X+T)+8*T*Y*
     1   *3*EXP(Y+X+T)+12*X**2*Y**2*EXP(Y+X+T)+24*T*X*Y**2*EXP(Y+X+T)+12
     2   *T**2*Y**2*EXP(Y+X+T)+Y**2*EXP(Y+X+T)+8*X**3*Y*EXP(Y+X+T)+24*T*
     3   X**2*Y*EXP(Y+X+T)+24*T**2*X*Y*EXP(Y+X+T)+2*X*Y*EXP(Y+X+T)+8*T**
     4   3*Y*EXP(Y+X+T)+2*T*Y*EXP(Y+X+T)+2*X**4*EXP(Y+X+T)+8*T*X**3*EXP(
     5   Y+X+T)+12*T**2*X**2*EXP(Y+X+T)+X**2*EXP(Y+X+T)+8*T**3*X*EXP(Y+X
     6   +T)+2*T*X*EXP(Y+X+T)+2*T**4*EXP(Y+X+T)+T**2*EXP(Y+X+T)+2)/(Y+X+
     7   T)**2
      return
      end

      function true (x,y)
      real tol
      integer niters
      real t, deltat, alpha, tstart, tstop
      integer nstep, nsteps
      common /timedep/ t,deltat, nstep, alpha
      TRUE = Y+X+T
      return
      end

END.
```