1984

ELLPACK: An Evolving Problem Solving Environment

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

Report Number:
84-497

ELLPACK: AN EVOLVING PROBLEM SOLVING ENVIRONMENT

John R. Rice

CSD-TR-497
October, 1984

ELLPACK: An Evolving Problem Solving Environment

John R. Rice
Computer Science
Purdue University
West Lafayette, IN 47907
U.S.A.

Preliminary version for IFIP WG2.5 Working Conference:
Problem Solving Environments for Scientific Computing
INRIA Sophia-Antipolis Laboratory
France, 17-21 June, 1985

CSD-TR 497
October 15, 1984

Abstract

This paper describes the possible evolution of the ELLPACK system from a research tool for software evaluation to an expert system for solving elliptic problems. The third "version" of ELLPACK is presently under development with the intention of exploring the impact of parallel and vector computers on its problem solving capabilities and to introduce problem solving methods based on geometric domain mappings.

ELLPACK: 1974-1983

The original motivation for the ELLPACK system was to provide a tool for the performance evaluations of software for solving partial differential equations (PDEs). It had become clear in the work of [Houstis et al, 1975] that one needed a well organized environment for making large scale evaluations of PDE software. One aspect of this environment is a high level language for expressing the PDE problem to be solved. A second aspect is to have a framework into which one can insert problem solving modules from various sources. Prototype systems called ELLPACK 77 and ELLPACK 78 were developed and tested extensively, about 200 copies of these systems were distributed world wide. On the basis of the experience with ELLPACK 77 and 78, a new system, simply called ELLPACK, was developed. It is described in detail in the book [Rice and Boisvert, 1984]. A third version, tentatively called Vector ELLPACK is under development and described later.

THE EXISTING ELLPACK SYSTEM

The existing ELLPACK system has three principal components: a user interface, an internal structure and a set of problem solving software modules. The system is based on the principles of a software parts technology; space limitations preclude describing the system in detail so we just discuss the user interface. Figure 1 shows a

simple ELLPACK program for solving the elliptic problem:

$$u_{xx} + u_{yy} + 3u_x - 4u = e^{x+y} \sin(\pi x)$$

with boundary conditions as follows on a trapezoidal domain:

$u = x^2$	for $y = 2, 0 \le x \le 1$
$u = 1 + y/2$	for $x = 1, 0 \le y \le 2$
$u = xy/2$	on the line joining the points (1,0) and (0,−1)
$u_x = 0$	for $x = 0, -1 \le y \le 2$

The problem is solved approximately by ordinary finite differences (5 POINT STAR) and Gauss elimination (BAND GE). A table and a contour plot of the computed solution $U(x,y)$ is made.

This example illustrates the power of the ELLPACK language for simple, direct problem solving but the principal lesson learned from the prototype systems ELLPACK 77 and 78 is that one must also have considerable flexibility in combining and controlling the very high level statements in ELLPACK. ELLPACK is a Fortran based system in that it is assumed the user knows Fortran and the whole system is implemented in Fortran. Thus is was natural to make ELLPACK an actual extension of Fortran and thereby gain access to the control and data structures of Fortran. This approach has proved surprisingly powerful and ELLPACK can be used to solve a wide variety of more complex problems (e.g. nonlinear problems, systems of equation, parabolic problems). Some of the ELLPACK programs for these problems have a certain complexity, but they are still far, far simpler than equivalent programs written in Fortran from scratch.

There are over 50 problem solving modules in ELLPACK and they provide the algorithmic horsepower to solve a broad range of problems. Even so, there are many important elliptic problem solving techniques that are not represented in ELLPACK.

While ELLPACK is useful for solving "real world" problems, it must be emphasized that its primary goal is not to be a production problem solving system. It is primary a tool for research and a system for experimenting with the nature of PDE solving systems. As such, it has a different emphasis on the selection of features than one would find in a production system.

THE NEXT ELLPACK

Many avenues of development are possible for ELLPACK. One avenue selected for development relates to the following general question about problem solving environments (PSEs):

> "Can the power of new computer architectures be exploited within the very high level language framework without perturbing the user interface?"

It is obvious that ELLPACK can be made to operate in any architecture that supports Fortran. But a PDE solving system such as ELLPACK has efficient execution as one of its important performance criteria because these problems tend to consume

```
*****************************************************
*                                                   *
*  EXAMPLE ELLPACK PROGRAM 3.B1                     *
*                                                   *
*      THIS PROBLEM HAS MIXED BOUNDARY CONDITONS.   *
*      THE PROGRAM COMPARES TWO DISTINCT METHODS FOR *
*      SOLVING THE SAME PROBLEM.                    *
*                                                   *
*****************************************************

OPTIONS.     LEVEL=1 $ TIME

EQUATION. UXX + (1.0+Y**2)*UYY  -  UX  -  (1.0+Y**2)*UY  =  F(X,Y)

BOUNDARY.     - U +  UX = 0.          ON  X=1.
                U       = TRUE(X,Y)   ON  Y=0.
                U +  UX = 2.0*EXP(Y)  ON  X=0.
                U       = TRUE(X,Y)   ON  Y=1.

GRID.        4 X POINTS  $  5 Y POINTS
OUT.         MAX(TRUE)

DIS.         5 POINT STAR
SOL.         LINPACK BAND
OUT.         TABLE(U) $ MAX(ERROR,7,9)

DIS.         HERMITE COLLOCATION
SOL.         BAND.GE
OUT.         TABLE(U) $ MAX(ERROR,7,9)

SUBPROGRAMS.
      FUNCTION TRUE(X,Y)
C        THE STANDARD ELLPACK TRUE SOLUTION FUNCTION (IF KNOWN)
         TRUE = EXP(X+Y) + ((X*(X-1.0))**2)*ALOG(1.0+Y**2)
         RETURN
      END
      FUNCTION F(X,Y)
C        CONSTRUCT F SO TRUE IS AS GIVEN
         F = ALOG(1.0+Y**2) * (2.0 + X*(-14.0 + X*(18.0 - 4.0*X)))
     A       + 2.0*((X*(X - 1.0))**2)*(1.0-Y-2.0*Y**2/(1.0+Y**2))
         RETURN
      END
```

Figure 1. A simple ELLPACK program for solving a linear, second order elliptic problem on a trapezoidal domain.

large amounts of computing power. So ELLPACK must do more than operate, it must operate efficiently.

It is important to identify the proper comparisons to be made in evaluating a PSE. It is not appropriate to compare the execution time performance with the fastest possible program for a problem; the latter would merely print the answer out. Rather, one should compare the execution time with the normal program that one would expect outside the PSE. In scientific computing today, this means using Fortran plus a reasonably large software library.

My conjecture is that ELLPACK already provides faster solutions in the ordinary case of sequential machine architectures. The typical user does gain by exploiting specific problem characteristics in his Fortran code, but he loses by not knowing how to implement (or select) his basic algorithm as well as the expert who prepared the ELLPACK module. This assumes that the ELLPACK module is not in the software library available, which is usually the case. The expert, of course, will be able to produce faster running programs than ELLPACK generates.

I further conjecture that PSEs will have a much larger advantage for more complex machines (e.g. vector machines or multi-processors). There is some quantitative evidence to support this in the paper [Umetani et al, 1984]. A simplistic condensation of these results is to say that a PSE for solving PDEs requires a tenth of the code and then runs three times as fast as Fortran code which has been both automtically vectorized and then hand optimized in a normal manner. This large advantage may surprise some, but its source is simple. The higher level statements in a PSE allow one to identify much more easily the inherent parallelism in the computation. Once this is true, an automatic code generation approach can outperform the usual process of expressing the computation in a sequential manner and then trying to reconstitute that into a parallel or vector form.

A second avenue selected for ELLPACK development is to incorporate geometric domain mapping methods into it. These methods have been very powerful over the past 150 years in the hands of clever scientists. They pose new challenges for scientific computing because they manipulate geometric mappings. The powerful graphics based PSE offers new potential for exploiting these methods. There are two related techniques here:

A. *Grid adaption*: This is the process of defining a grid (a partition of the domain) that reflects the nature of the problem being solved. Both grid refinement and changes of coordinates (as in the moving finite element method) are included in this technique.

B. *Domain regularization*: This is the process of mapping one domain onto another which is more "regular", i.e. more amendable to the use of particular problem solving methods. Classical examples of this technique are the use of conformal mappings and polar coordinates.

The availability of efficient methods for solving general PDEs removes some of the classical constraints on the mappings to be allowed. Thus, they need merely be reasonably smooth rather than conformal or define orthogonal coordinate systems. In many instances useful mappings can be generated cheaply, such as with blending functions [Gordon and Thiel, 1984] to map a four sided domain onto a square. These techniques are discussed in some detail in [Houstis and Rice, 1984] from the point of view of domain regularization to allow the exploitation of vector architecture.

Figure 2 shows two examples of these techniques. In Figure 2A a mapping of a rectangle into itself is made which (i) preserves the logically rectangular coordinate system, and (ii) refines the grid in the area where the PDE solution oscillates more. The key to the success of this approach is to choose the proper function to define the new grid and to have an efficient algorithm to use in equidistributing the grid with respect to this function. This mapping is computed by equidistributing the residual in a "coarse" solution with a finer grid. A heuristic, multigrid like method (due to C. Ribbens and J. Rice) is used.

Figure 2B shows the use of blending functions to map a domain with four curved sides into a rectangle. An important open question here is to devise "cheap" and "smooth" mappings of N-sided domains onto rectangles. Alternatively, one can also exploit domain substructuring to map a more complex domain onto the union of rectangular domains. A powerful PSE incorporating these techniques will allow a user to combine such mappings in clever ways to exploit the particular nature of an application.

Note in both examples shown in Figure 2 that a simple Poisson problem on the original domain would be transformed into a general linear elliptic problem on the new domain.

A PRODUCTION ELLPACK

An ELLPACK-like system for production use would have three (at least) facilities not now considered for inclusion. All three facilities reflect the fact that real-world problems are often complicated. Or perhaps it reflects the fact that most of the simpler real-world problems have already been solved. These facilities are to provide the ability to handle:

(i) simultaneous PDEs,

(ii) multiple domains and interfaces,

(iii) general three-dimensional domains.

The first two of these are certainly not easy, but there seems to be a higher ratio of routine work to interesting research here than in some other areas. The geometric difficulties with general 3D domains are formidable. No one has developed a methodology that provides smoothness, reasonable efficiency, flexibility and generality for defining and manipulating such domains. However, this problem is not really one of PDEs so perhaps we can foist it off on some other research community. At this time, the most effective way to handle 3D geometry seems to be through constructions using a set of cannonical shapes (e.g., tetrahedrons, cubes, pieces of spheres, cylinders, etc.)

If one did incorporate these facilities into a nice PSE, then one would have a truly powerful facility. It would require supercomputer power because one would almost immediately be trying iteration schemes for nonlinear interface conditions (or some such thing) in general 3D geometry.

AN EXPERT SYSTEM FOR ELLIPTIC PROBLEMS

ELLPACK is naturally evolving toward an expert system for elliptic problems and serious analysis of the requirements is now being made. The three essential

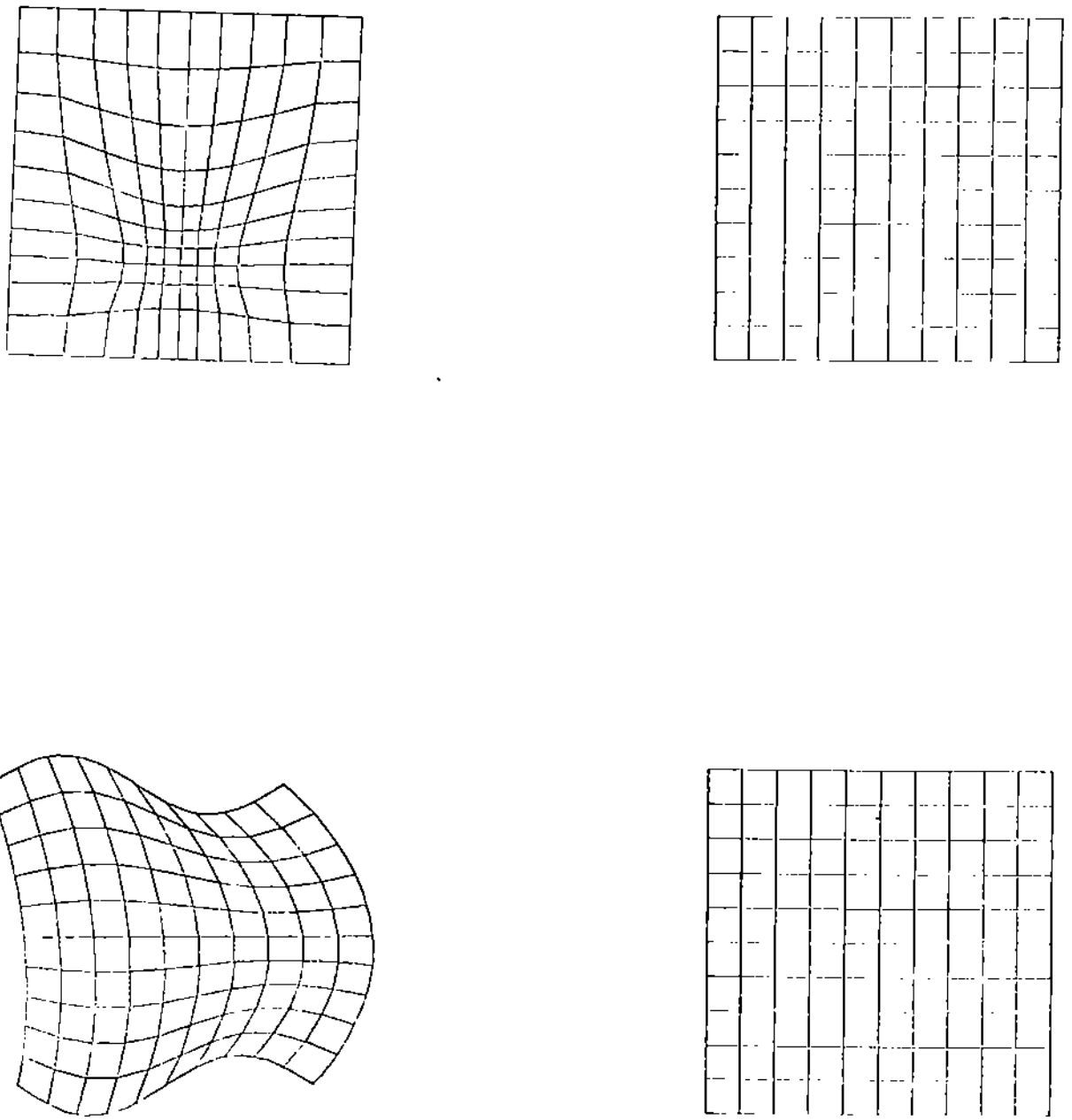

Figure 2. Domain mapping techniques: (A) Introducing a coordinate system into a square which refines the grid appropriately, (B) Introducing a coordinate system into a curved domain by blending which makes it rectangular.

requirements for such a system are:

(i) expertise

(ii) symbolic analysis of the problem

(iii) a decision mechanism

The ELLPACK project was started because of doubts about the validity of the "expert's" knowledge about the best ways to solve PDEs. ELLPACK is embedded in a system [Boisvert et al, 1979] to gather data about the problem solving power of software modules. The project has confirmed the shortcomings of the experts' knowledge, but it has also convinced me that, as little as the experts know, they know a lot more than the typical scientists. Thus, constructing an expert system for PDEs is a viable project even while recognizing that substantial efforts are also needed in enlarging the domain of the experts' knowledge.

The existing ELLPACK already does a small amount of symbolic processing of the problem, for example, it sets a switch automatically when the PDE has constant coefficients. Much more extensive preprocessing, both symbolic and numerical, is needed for an expert system. The symbolic preprocessing is used both to select the methods to use and, within a given method, to tailor the calculation to the particular problem at hand. Thus, the expert system should discover that a PDE has a forcing function with a 1/x singularity at a corner of the boundary. It should also discover that the u_x and u coefficients are constants to be evaluated only once and the u_y coefficient is missing so that no terms involving its coefficient are ever included in a method.

While symbolic preprocessing can give a lot of information about the elliptic problem, there are some places where numerical preprocessing is appropriate. A simplified scenario is as follows: The system is presented with a PDE whose symbolic appearance is ordinary. The user gives no guidance, so the system chooses a course grid and computes a tentative solution. The error estimates applied to this solution indicate that no accuracy has been obtained. The next step is to numerically explore all the coefficient functions of the PDE and its boundary conditions. If one or two of them oscillate rapidly somewhere, or has a sharp peak somewhere, then one has an important clue as to the next step to be taken to solve the problem.

Finally, there is the question of decision mechanism. Initially, expert systems are likely to start out using a "decision tree". The system developer feels that the system can reach a good decision by measuring certain things and branching on the basis of the values. If one path leads to a deadend, then various backtracking and restarting schemes can be used. While the number of decision variables is moderate (say a hundred or so), their management is not a cruicial issue and the decision tree can be implemented in the PDE's natural computing environment. As the size and expertise of the system grows, there comes a time when one must become more organized about these matters and add a "decision-rule processing" component to the PSE.

REFERENCES

1. Boisvert, R.F., Houstis, E.N., and Rice, J.R., A system for performance evaluation of partial differential equations software, IEEE Trans. Software Eng., 5

(1979) 418-425.

2. Gordon, W.J., and Thiel, L.C., Transfinite interpolation techniques for exact a priori matching of boundary conditions for elliptic problems, *J.* Numer. Math. Engng., (1984).

3. Houstis, E.N., Lynch, R.E., Papatheodorou, T.S., and Rice, J.R., Development, evaluation and selection of methods for elliptic partial differential equations, Ann. Assoc. Calcul. Analog. 11 (1975) 98-105.

4. Houstis, E.N., and Rice, J.R., Vector ELLPACK: Domain mappings and parallel geometric discretization, in *Advances in Computer Methods for Partial Differential Equations* V. (Visnevetsky and Stepleman, eds.), IMACS, Rutgers University (1984) 195-198.

5. Rice, J.R., and Boisvert, R.E., *Solving Problems using ELLPACK*, (Springer-Verlag, New York, 1985).

6. Umetani, Y., Tsuji, M., Iwasawa, K., and Hiranyama, H., DEQSOL: A numerical simulation language for vector/parallel processors (1984).