

Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

AN INTERACTIVE GRAPHICAL SYSTEM FOR
THE APPROXIMATION OF PARTIAL DIFFERENTIAL
EQUATIONS

by

Paul S. LaFata

Technical Report # 138

October 1971

AN INTERACTIVE GRAPHICAL SYSTEM FOR THE APPROXIMATION OF PARTIAL DIFFERENTIAL EQUATIONS

by

Paul S. LaFata

ABSTRACT

An interactive graphical system which can be used to approximate bivariate functions or to obtain approximate solutions to partial differential equations is described. The function or differential equation is entered symbolically, so the system may be used by someone with no programming experience. The basic mathematical technique is to assume the approximate solution is a linear combination of user selected functions, and to use linear programming to determine the coefficients of these functions. The approximate solution and error information can be graphically displayed in several different ways, and based on these results various parameters may be modified on line so as to obtain a better approximation.

This research was supported in part by the National Science Foundation under grant GJ-0362.

1. Introduction

Interactive graphical systems have proven to be powerful tools in the area of numerical analysis. This has been demonstrated by a number of general purpose mathematical analysis systems and also by some more specific systems for the approximation of data, functions, and differential equations [6,9].

In this paper we describe an interactive graphical system, PDE, which allows us to approximate functions of two variables and to obtain approximate solutions to Partial Differential Equations. This system was developed as an extension to two previous systems which allow us to solve data fitting problems [7], and approximate univariate functions or obtain approximate solutions to ordinary differential equation boundary value problems [13]. In all of these cases the user may impose certain auxiliary conditions on the approximate solution, such as convexity. The basic mathematical technique is to assume a solution of the form $v(\alpha, x) = \sum_{i=1}^m \alpha_i \phi_i(x)$, and to use linear programming (LP) to determine the coefficients α_i of the user selected approximating functions $\phi_i(x)$, $i = 1, \dots, m$. This technique has been discussed by a number of authors for the approximation of functions and approximate solution of ordinary differential equations [10,2], and also for the approximate solution of certain types of partial differ-

ential equations [11,12].

The system PDE is implemented on the Univac 1108 and Adage AGT/10 computers, which are connected together via a high speed data channel. The problem is specified at the Adage, with the differential equation and boundary conditions being entered symbolically so the system may be used by a scientist or engineer with no programming experience. The domain for the problem may be specified to be any polygonal domain, and is not required to be simply connected. For the approximating functions $\phi_i(x)$, PDE uses a tensor product of spline functions with the user selecting the degree and number of basis splines to be used. This offers an excellent compromise between generality and ease of use in this type of interactive program [1,5]. The linear program is generated and solved on the 1108, giving the coefficients of the spline approximation to the solution. Contour plots of this approximate solution and related error information may then be displayed at the user's option. If the results obtained are not satisfactory, he may then modify his earlier choices and repeat the process.

In Section 2 we briefly give a mathematical description of the approximation problem.

Section 3 contains a description of the system PDE as it would appear to a user. The final section contains some examples of partial

differential equations which have been approximated using PDE. Elliptic and hyperbolic problems are approximated on domains of various shapes, and we see the effect that modifying certain parameters has on these approximations. A more detailed description of PDE which includes examples of parabolic problems is contained in [8].

2. Mathematical Formulation

We now consider the following class of second order boundary value problems given by

$$L_k[u] = f_k(x, y) \text{ on } D_k, \quad k = 0, 1, \dots, n \quad (2.1)$$

where $D = \bigcup_k D_k$ is a closed and bounded domain in E^2 , the L_k are linear differential operators defined below, the $f_k(x, y)$ are analytic functions on D_k , and we assume that there exists a solution $u(x, y) \in C[D]$. We define

$$\begin{aligned} L_0[u] \equiv & a_{00}(x, y)u + a_{01}(x, y)u_x + a_{02}(x, y)u_y \\ & + a_{03}(x, y)u_{xx} + a_{04}(x, y)u_{xy} + a_{05}(x, y)u_{yy} \\ & \text{on } D_0. \end{aligned} \quad (2.2)$$

Then $L_0[u] = f_0(x, y)$ is a linear partial differential equation on D_0 . We assume that D_0 is an open polygonal domain which is not necessarily simply connected. For $k = 1, \dots, n$ we form the n boundary conditions by defining

$$L_k[u] \equiv a_{k0}(x, y)u + a_{k1}(x, y)u_N \text{ on } D_k \quad (2.3)$$

where u_N is the normal derivative of u with respect to the boundary D_k of D . As will be seen in the next section, the boundary

conditions may also include terms with u_x and u_y . The functions $a_{ki}(x,y)$ for $k = 0,1,\dots,n$ are, like $f_k(x,y)$, analytic functions on D_k .

We wish to obtain a function $v(\alpha,x,y)$ which will approximate $u(x,y)$, the exact solution of (2.1). For the special case $a_{00}(x,y) = 1$ and $a_{0j} = 0$ for $j = 1,\dots,4$ (2.2) reduces to $u = f_0(x,y)$. In this case we interpret the approximation problem as that of finding a best approximation in the minmax norm to $f_0(x,y)$ (just approximating a function). In the rest of this discussion we shall refer to (2.2) as a partial differential equation, with $u = f_0(x,y)$ as a special case.

The approximation

$$v(\alpha,x,y) = \sum_{i=1}^m \alpha_i \phi_i(x,y) \quad (2.4)$$

will be given by a linear combination of m selected functions $\phi_i(x,y)$, $i = 1,\dots,m$. As mentioned in Section 1 the class of functions used in PDE is a tensor product of splines. The details concerning the exact nature of these splines appears in [7] and [12]. We only comment that in the one-dimensional case a basis for an arbitrary spline of specified degree r and uniform knot size is obtained by a slight modification of B-splines [5]. We denote these basis functions as β splines. For the two-dimensional case the β splines are used to form a basis for tensor product splines as explained in [12]. This representation simplifies the computation

by leading to well-conditioned matrices with a special structure. In the system PDE we may select β splines of degree $r \in \{2, 3, 4, 5\}$. We also select the number m_x and m_y of the β splines to be used along the x and y axis respectively, where $m_x > r$, $m_y > r$, and we are restricted to $m_x m_y \leq 60$. The knot sizes are automatically determined using

$$\Delta_x = \frac{b_1}{m_x - r} \quad \text{and} \quad \Delta_y = \frac{b_2}{m_y - r} \quad (2.5)$$

where the numerator is just the width of the domain along the particular axis. These splines will be used to form $m = m_x m_y$ tensor products. In the next part of this section, for simplicity of notation, we shall use

$$v(\alpha, x) = \sum_{i=1}^m \alpha_i \phi_i(x) \quad (2.6)$$

where we understand that $\phi_i(x)$ is just one of these tensor products, and where $\alpha \in E^m$ denotes a column vector with elements α_i , and $x \in E^2$.

Our boundary value problem as given in (2.1) can be expressed

$$L_i[u] = f_k(x) \quad \text{on} \quad D_k, \quad k = 0, 1, \dots, n. \quad (2.7)$$

This problem will be approximated on a discrete grid of p_k points $x_j \in D_k$ for $j \in 1, \dots, p_k$. Note that x_j now designates a distinct

grid point, and we define p_0 discrete grid points on the interior D_0 , and p_k points on the boundary D_k , $k \in 1, \dots, n$. Thus we have a total of $p = \sum_{k=0}^n p_k$ grid points.

We wish to determine a function $v(\alpha, x)$ for $x \in D$ which minimizes a weighted sum of the maximum errors in the differential equation and boundary conditions over the p discrete points. In particular, we want to find α so as to minimize

$$\Psi(\alpha) \equiv \sum_{k=0}^n \omega_k \|L_k[v(\alpha)] - f_k\|_{D_k} \quad (2.8)$$

where the ω_k are non-negative weights and

$$\|L_k[v(\alpha)] - f_k\|_{D_k} \equiv \max_{x_j \in D_k} |L_k[v(\alpha, x_j)] - f_k(x_j)|.$$

If we let

$$\begin{aligned} \sigma_{ik} &= L_k[\phi_i(x)] \text{ and } \sigma_{ikj} = L_k[\phi_i(x_j)], \quad x_j \in D_k \\ \text{for } j &= 1, \dots, p_k \end{aligned} \quad (2.9)$$

then (2.8) can be expressed as minimizing

$$\Psi(\alpha) = \sum_{k=0}^n \omega_k \left\| \sum_{i=1}^m \alpha_i \sigma_{ik} - f_k \right\|_{D_k} \quad (2.10)$$

where

$$\left\| \sum_{i=1}^m \alpha_i \sigma_{ik} - f_k \right\|_{D_k} \equiv \max_{x_j \in D_k} \left| \sum_{i=1}^m \alpha_i \sigma_{ikj} - f_k(x_j) \right|.$$

For the details as to how a problem of this nature is expressed and solved using LP, see [8].

We now briefly mention the type of auxiliary conditions that the system PDE can handle. We may place lower and/or upper bounds on the elements of the coefficient vector α . In addition, lower and/or upper bounds may be placed on any of the five functions $v(\alpha, x, y)$, $v_x(\alpha, x, y)$, $v_y(\alpha, x, y)$, $v_{xx}(\alpha, x, y)$, and $v_{yy}(\alpha, x, y)$ at any point $(\tilde{x}, \tilde{y}) \in D$. As is illustrated by one of the examples in [8], imposing the bounds on an appropriate set of points in D might insure that our resultant approximation be convex or concave. The manner in which these auxiliary conditions may be added to the LP problem is also discussed in [8].

Now suppose that we have successfully solved the LP problem and obtained the coefficient vector $\hat{\alpha}$ and corresponding best approximation $v(\hat{\alpha}, x, y)$. We might then want to see contour plots of the errors in the differential equation (also called "defect") given by

$$ED(x, y) \equiv L_0[v(\hat{\alpha}, x, y)] - f_0(x, y) \quad (2.11)$$

and graphs of the error in the boundary conditions, given by

$$CND_k(x, y) \equiv L_k[v(\hat{\alpha}, x, y)] - f_k(x, y) \quad k = 1, \dots, n. \quad (2.12)$$

These graphs are displayed by PDE upon request, as described in the next section. Also, if we happen to know the exact solution

$u(x, y)$, we may request a plot of the solution error given by

$$SE(x, y) \equiv v(\hat{\alpha}, x, y) - u(x, y). \quad (2.13)$$

As has been discussed in [11] and [4], if we have a monotone property (maximum principle) then the maximum grid errors in the differential equation (2.11) and boundary conditions (2.12) give us an error bound for the approximate solution $v(\hat{\alpha}, x, y)$.

It should be noted that the number and location of the p grid points is an important consideration which is not discussed in any detail in this paper, but is left as a topic for further research. Ideally, they should be chosen so as to trap the maximum error. After seeing a plot of the defect (2.11), the user may wish to define additional grid points where the defect is largest and then resolve the problem. The choice of the weights ω_k is also an important consideration which deserves further study. It has been briefly discussed in [3], and an example which shows the effect of modifying these weights for a particular problem is given in Section 4.

3. Description of PDE

In this section we shall demonstrate how the system PDE would be used in order to find an approximate solution to a partial differential equation. We shall only explain those portions of the system that would be visible to the user. A description of the data structure and various "hidden" routines is given in [8].

The system PDE is implemented on the Univac 1108 and Adage AGT/10 computers. The Adage has a 16,000 30-bit word memory, two magnetic tape drives, a graphical display with light pen, and various function switches and analogue inputs. PDE actually consists of two programs, APDE and UPDE which run on the Adage and Univac 1108 respectively. These two programs interact in a master/slave environment, with the role of the master being assumed by APDE. They communicate via a high speed data channel connecting the two computers. The user interacts only with APDE and in general would not even be aware of the role played by UPDE. Almost all of the routines in APDE are written in ADEPT, the Adage machine language, whereas UPDE is written primarily in FORTRAN.

To initiate the program UPDE, a small deck of about five cards must be submitted at the 9300 remote I/O station which is adjacent to the Adage. This deck instructs the 1108 to fetch from drum the program UPDE and execute it. Usually UPDE is active within a

couple of minutes. Assuming APDE has been initiated at the Adage, the display shown in Figure III.1 would appear. In this frame we are asked if we wish to approximate a two-dimensional function or a partial differential equation.

DO YOU WISH TO APPROXIMATE:

* A TWO-DIMENSIONAL FUNCTION

* A PARTIAL DIFFERENTIAL EQUATION

Figure III.1

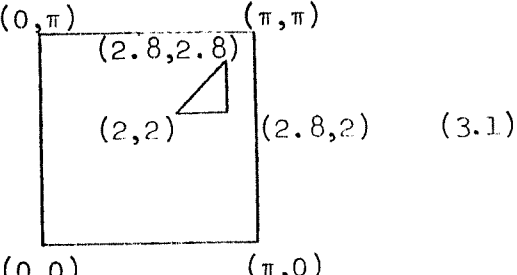
Our selection would be made by pointing at the appropriate line with the light pen. We will call this "tagging".

Suppose we wish to approximate, as described in Section 2, the partial differential equation

$$u_{xx} + u_{yy} = -2 \sin(x) \sin(y) \text{ on}$$

the domain D with the four

boundary conditions



$$(3.1)$$

- 1) $u = 0$ on D_1 (the four sides of the square)
- 2) $u_N = 0$ on D_2 (the hypotenuse of the triangle)
- 3) $u = .416 \sin(x)$ on D_3 (the horizontal side of the triangle)
- 4) $u = .335 \sin(y)$ on D_4 (the vertical side of the triangle)

where u_N is the normal derivative of u . This problem, which has a known exact solution, is chosen for illustrative purposes.

After tagging the appropriate line in Figure III.1 we are requested by Figure III.2 to enter the partial differential equation (PDE). This is done by using the light pen in conjunction with the sixteen function

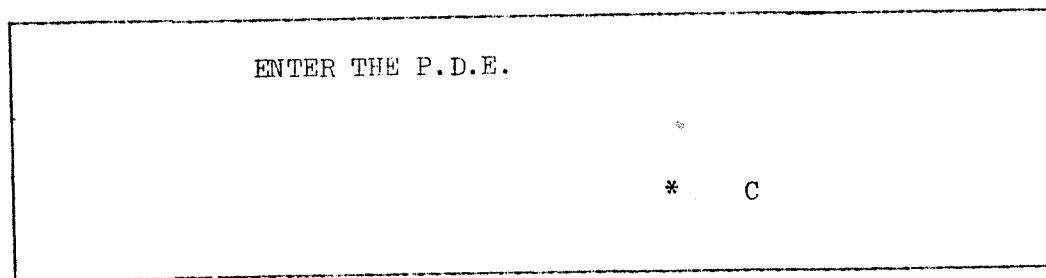


Figure III.2

switches shown in Figure III.3. The function switches programmatically represent the various syntactic units indicated below the switches. When a switch is pressed the appropriate symbol or symbols are added to the display shown in Figure III.2 to form the equation. There are not enough function switches to include one for each of the variables and functions in the set $\{x, y, u, u_x, u_y, u_N, u_{xx}, u_{xy}, u_{yy}\}$, and rather than redefine the switches when one of the members of this set is to be

selected, we decided to use the light pen in conjunction with switch 1 as illustrated below. Whenever switch 1 is pressed, the following list of options is added to the display.

* X	* UX	* UXX
* Y	* UY	* UXY
* U		* UYY
	* =	

and the desired text can be selected with the light pen. Note that

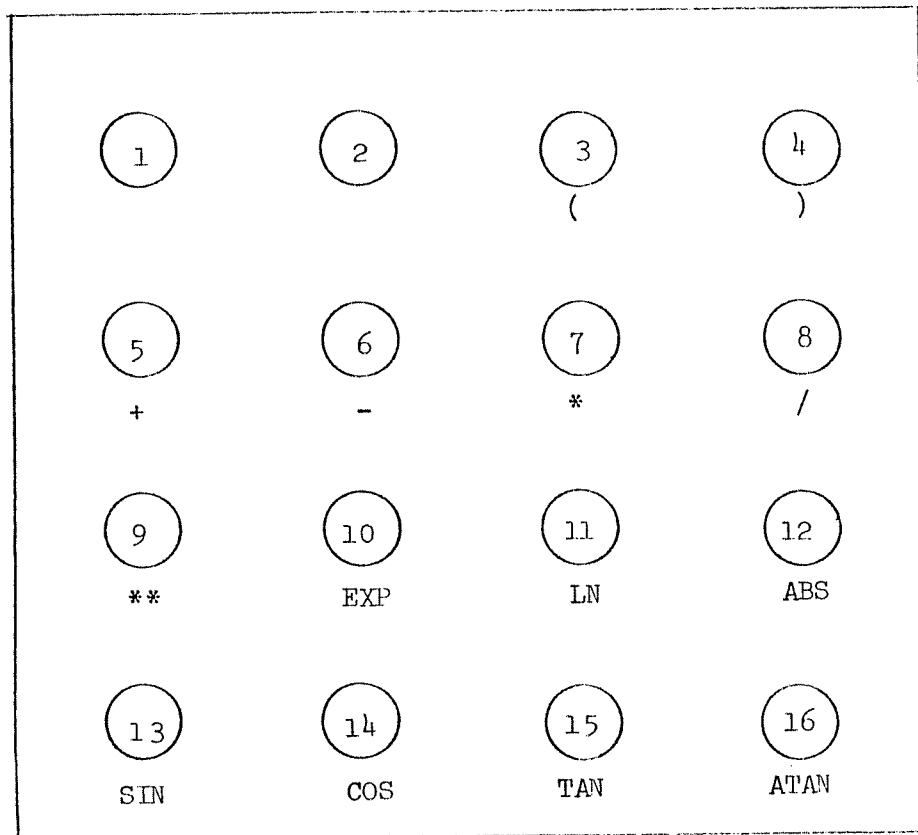


Figure III.3 Panel of Sixteen Function Switches

since we cannot display subscripts, we represent u_x as UX, etc. If the equal sign has been entered into the differential equation and switch 1 is subsequently pressed, then the "*" = " option and the functions of u are not made available since they are not valid syntactic units for the right-hand-side.

Also, if we had indicated in Figure III.1 that we wished to enter a two-dimensional function, then pressing switch 1 would place only the options

* X

* Y

on the display, since the various partial derivatives would not be legitimate entries for this type of problem.

Function switch 2 allows us to enter a constant. When it is pressed an array of numbers like the one in Figure III.10 is added to the display, and the desired constant can be entered with the light pen. By operating the function switches with one hand and the light pen with the other, the equation as shown in Figure III.4 can be generated easily and quickly. If an error has been made, the equation can be erased from right to left by pushing the foot pedal which is essentially another function switch. Each time the foot pedal is operated one syntactic unit in the equation is erased.

```

ENTER THE P.D.E.

UXX+UYY=-2.00000*SIN(X)*SIN(Y)

* C

```

Figure III.4

When the equation has been entered and "* C" is tagged, a simple algebraic compiler (syntax analyzer routine) parses the equation and checks for syntax errors. If there are any syntactical errors we are taken back to Figure III.2, otherwise we move on to Figure III.5 where we are requested to specify the x and y coordinates of the end points of the lines which specify the domain. In Figure III.5 the seven corner points of the domain have been entered. To enter point A the constant "0.0" is first generated using the array of numbers, then when "* X =" and "* Y =" are tagged the constant just formed is placed in the appropriate lines. After tagging "* ENTER" the text "A 0.000 0.000" is added to the display and point A is formally defined as having the coordinates (0,0). The remaining three corners of this outside square are similarly entered. Only closed curves are allowed, and we shall call any closed curve, such as the square, a "cycle". After the four corners of the square or "cycle" are

entered, tagging "* END CYCLE" would place a short horizontal line beneath the character D as shown in Figure III.5, and would indicate that a line is to be drawn from point D to the first point in the cycle, point A. The three points in the second cycle are similarly entered. If an error has been made, tagging "* DELETE" would delete the last entry.

			* X = 2.80000
			* Y = 2.20000
A	0.000	0.000	
B	0.000	3.142	2.20000
C	3.142	3.142	0 1 2
<u>D</u>	3.142	0.000	3 4 5
E	2.200	2.200	6 7 8
F	2.800	2.800	9 - .
<u>G</u>	2.800	2.200	E RESET
			* ENTER
			* DELETE
			* END CYCLE

Figure III.5

The domain just specified by the user is shown in the next display (Figure III.6). If a mistake had been made when specifying the points it would certainly be apparent here, and tagging "* B" would

allow us to go back to Figure III.5 where the necessary corrections could be made. Tagging "* C" allows us to continue to the next display. Most of the displays within APDE have at least the two options "* B" and "* C", so that the user has ample opportunity to make changes or correct mistakes when necessary.

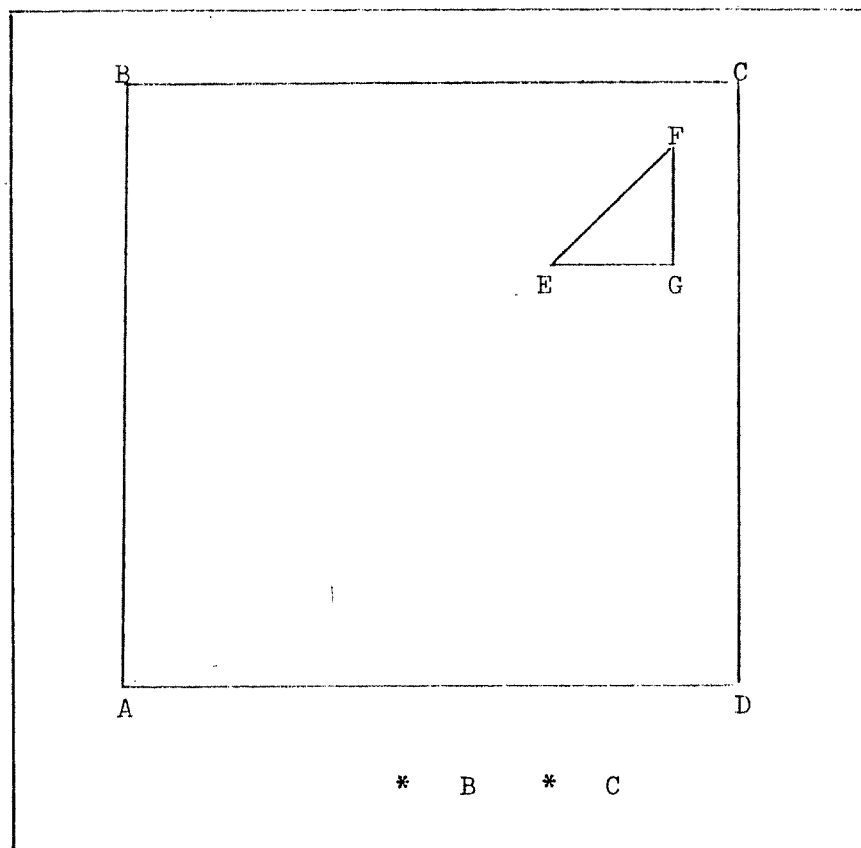


Figure III.6

The differential equation will be approximated, as explained in Section 2, on a grid of points within the domain D. After specifying the number of points to be used along each axis, we would

see Figure III.7 where a grid of 11×11 or 121 points has been placed on the domain. Since we do not wish to approximate the differential equation on the three points lying in the triangular region, we delete these points simply by passing the light pen over them. Each point successively disappears as it is tagged by the pen. If we wished to include a few additional points in our grid, this could be done by using the "floating dot" shown positioned in the triangle of Figure III.7.

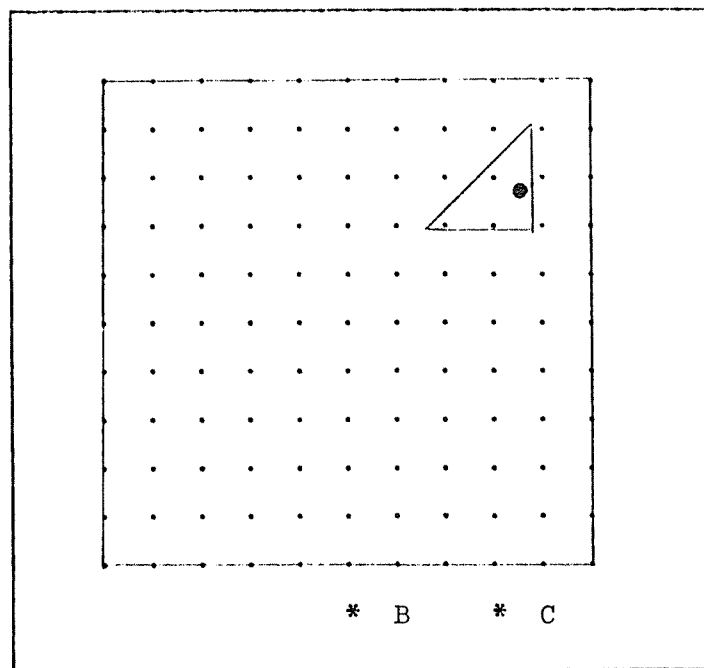


Figure III.7

The x and y coordinates of the floating dot are determined by the positions of two of the six variable control dials. Thus, by manipulating the control dials we can position the dot anywhere. Assume

that we want to add three points to the grid near the hypotenuse of the triangle. After positioning the dot where desired, pressing function switch 1 defines an extra point where the dot is located. Figure III.8 shows us the display after erasing the three points in the triangular region, and adding three points near the hypotenuse. Note that the erase option is also available for any additional points defined in this way. In this manner, using the light pen, variable control dials, and function switch, we may add and delete points as desired.

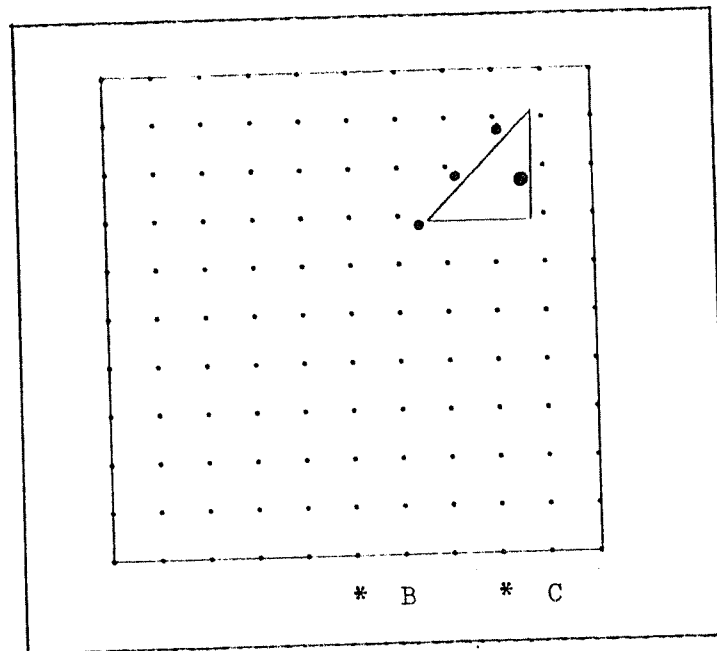


Figure III.8

Now we are requested to enter the boundary conditions. This is done using the light pen and function switches as explained earlier, except that when function switch 1 is pressed only the

following options are made available:

* X	* UX
* Y	* UY
* U	* UN
	* =

Note that the second derivatives, which would not be valid entries, are not made available. In Figure III.9 we see that the first boundary condition $u = 0$ has been entered. After having the syntax validated, we are requested in Figure III.10 to specify the weight ω_1 , as explained in Section 2, for this boundary condition.

1ST BOUNDARY CONDITION

U=0.000000

* C

Figure III.9

Only the weights $\omega_1, \omega_2, \omega_3$, and ω_4 for the boundary conditions may be specified. The weight for the differential equation is automatically set to unity, $\omega_0 = 1$. If we attempted in Figure III.10 to enter a negative weight, we would get no response. After specifying the weight

ENTER THE WEIGHT FOR THIS		
BOUNDARY CONDITION		
* W = 1.000000	0	1.000000
	1	2
	3	4 5
	6	7 8
	9	- .
	E	RESET
* C		

Figure III.10

we are taken back to Figure III.6 where we are requested to tag the line upon which we wish to have this first boundary condition imposed. Tagging line AB takes us to a display where we may specify the number of points to be placed on AB. After requesting a grid of eleven points, we would be shown Figure III.11 where the points are automatically spaced along the proper line. These are points on D_1 at which the first boundary condition will be approximately satisfied. Points may also be added and deleted as explained earlier. When the function switch is pressed, the additional point is automatically "zoomed" from the position of the floating dot which is shown in the figure to the boundary line, so we are again prevented from making inconsistent entries.

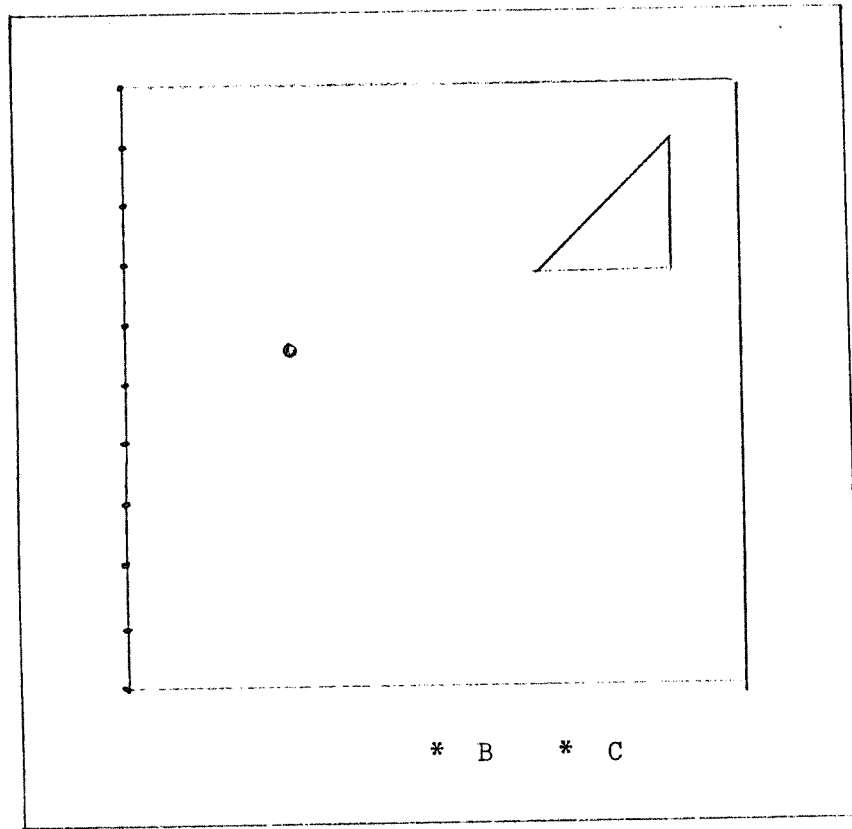


Figure III.11

We are informed in Figure III.12 that the boundary condition will be approximated at eleven points on line AB. Since we wish to add three more lines to this set, tagging "`* ADD`" would take us to Figure III.6 so we may include additional lines. In this manner we would define points on the four lines of the outer boundary and would finally be presented with the display shown in Figure III.13, giving us a summary of what we have done. There are various options available. For instance, if we decide not to include line CD in our

LINE	NO. OF POINTS	
		* ADD
AB	11	
		* DELETE
		* CHANGE
		* REWEIGHT
		* C

Figure III.12

LINE	NO. OF POINTS	
		* ADD
AB	11	
BC	10	* DELETE
CD	10	* CHANGE
DA	9	
		* REWEIGHT
		* C

Figure III.13

set, tagging "`* DELETE`" and "`CD`" would remove it. Similarly, tagging "`* CHANGE`" and "`DA`" for example would allow us to change the number of points on line `DA`. We also may change the weighting of this boundary condition by tagging "`* REWEIGHT`" which would take us to Figure III.10.

Now the second boundary condition $u_N = 0$ and its weight would be entered. In Figure III.14 we see that this boundary condition is to be approximated at five points along the line `EF` (the floating dot is also shown). Since this condition contains a normal derivative,

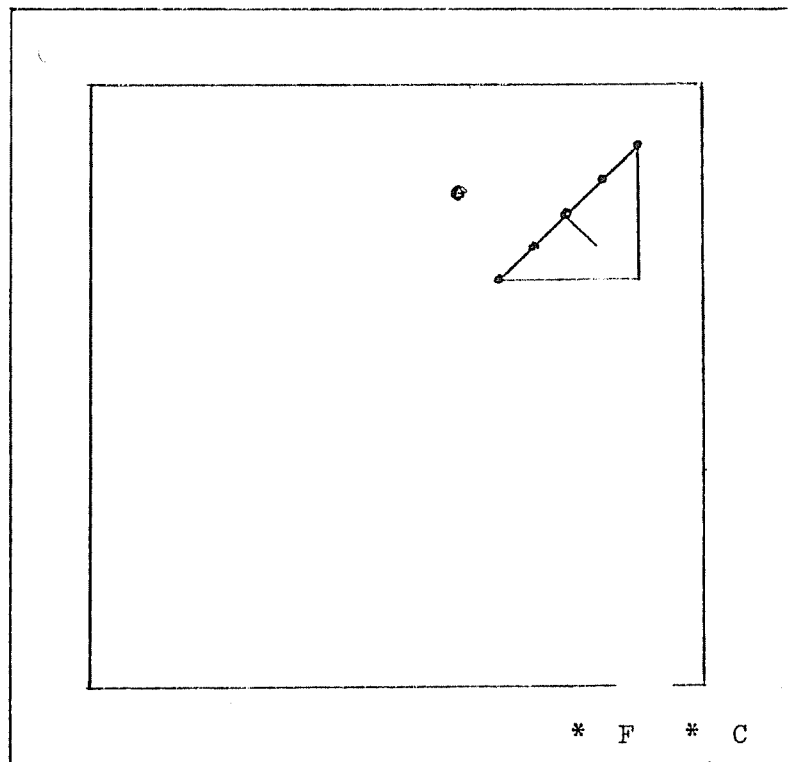


Figure III.14

a short vector perpendicular to EF is calculated (actually by UPDE) and this vector is also shown in the figure. Since this is a homogeneous boundary condition (the right-hand-side is zero), the sign of the normal derivative is not important. In general however, we must specify the interior of the domain with respect to a boundary line. The vector shown indicates the interior direction. If it initially points in the wrong direction it can be "flipped" to point in the opposite direction by tagging " $* F$ ". This is exactly the sort of decision which would be complicated to program but is a trivial matter for the user to resolve. In the same manner we specify that the remaining two boundary conditions are to have a weight of unity and are each to be approximated on a grid of five points on lines GE and FG.

If we have any auxiliary conditions, they may be entered as shown in Figure III.15. In particular, we can place lower and/or upper bounds on the coefficients α_i of the splines, or bounds on the functions $u(x,y)$, $u_x(x,y)$, $u_y(x,y)$, $u_{xx}(x,y)$, or $u_{yy}(x,y)$ at any specified points (as indicated in Section 2, these bounds will actually be placed on the approximation $v(\alpha,x,y)$, $v_x(\alpha,x,y), \dots$). Since we do not wish at this time to impose any auxiliary conditions, we would be taken to Figure III.16 in order to select the degree r of the desired splines. As shown, we have decided to use cubic splines and in Figure III.17 are requesting that sixteen of these splines be used in finding

PLACE BOUNDS ON:

- * THE COEFFS OF THE SPLINES
- * THE FCN U
- * THE FCN UX
- * THE FCN UY
- * THE FCN UXX
- * THE FCN UYY
- * NO MORE BOUNDS

Figure III.15

SELECT THE DEGREE OF THE SPLINES

- | | | |
|-------|-----|-----|
| | * | 2 |
| | * | 3 |
| R = 3 | * | 4 |
| | * | 5 |
| | * B | * C |

Figure III.16

an approximate solution $v(\hat{\alpha}, x, y)$, as explained in Section 2.

```

SELECT  MX  AND  MY, THE NUMBER OF SPLINES TO BE USED
ON THE X-AXIS AND Y-AXIS

THE KNOT SIZES WILL BE:

      3.142/(MX-R)  ON X-AXIS
      3.142/(MY-R)  ON Y-AXIS

*  MX = 4
*  MY = 4

                                4.000000
                                0 1 2
                                3 4 5
                                6 7 8
                                9 - .
                                E  RESET

                                *  C

```

Figure III.17

At this point the problem has been defined and the data, which has been sent up to the 1108 in a piecemeal manner as we entered it, is now used by UPDE to generate the primal LP problem which is then solved by SIMPD_X, a double precision linear programming routine. For this LP problem the 1108 required a total of 16 seconds. The optimal coefficient vector $\hat{\alpha}$ and corresponding approximate solution $v(\hat{\alpha}, x, y)$ are now available for display and error study purposes. The approximate solution and various error curves may now be displayed on the Adage in the form of contour maps. The approximate functions

will be evaluated on a grid of 20×20 points to give a data set of 400 values for the contour maps. A much finer grid would of course give smoother contour maps, but would require that the various functions be evaluated at many more points. As a reasonable compromise we chose a grid of 400 points. Quite often, if a portion of the domain has been cut out as in our example, it is because the solution takes on exceedingly large values in that area. If this were the case, then the contour maps would be distorted unless the cut out region were also eliminated from the maps. In order to allow for this case, a display similar to that shown in Figure III.7, but with a grid of 400 points would appear and we could erase points in the triangular area if we wished. This would be done as UPDE sets up and solves the LP problem.

When the LP problem has been solved, a contour map of the approximate solution $v(\hat{\alpha}, x, y)$ as shown in Figure III.18 would be displayed. It consists of ten level lines which are numbered in the figure. Line ten which is the highest, is also the brightest and has a value of .7249, whereas line 1, the dimmest one, has a value of -.1081. In this plot there is a constant increment of .09256 between level lines. The function attains its minimum value of -.1544 at points on the outside boundary, and its maximum of .7712 at the center. This data is given beneath the contour map.

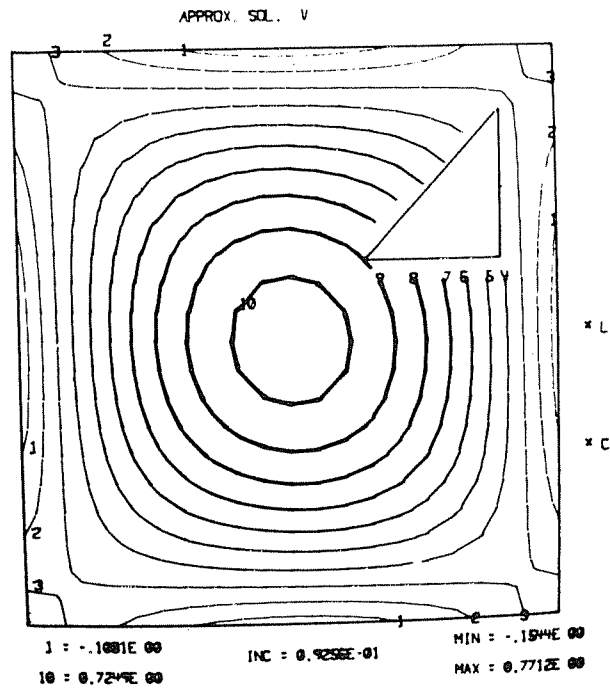


Figure III.18 Contour Map of the Approximate Solution $v(\hat{\alpha}, x, y)$

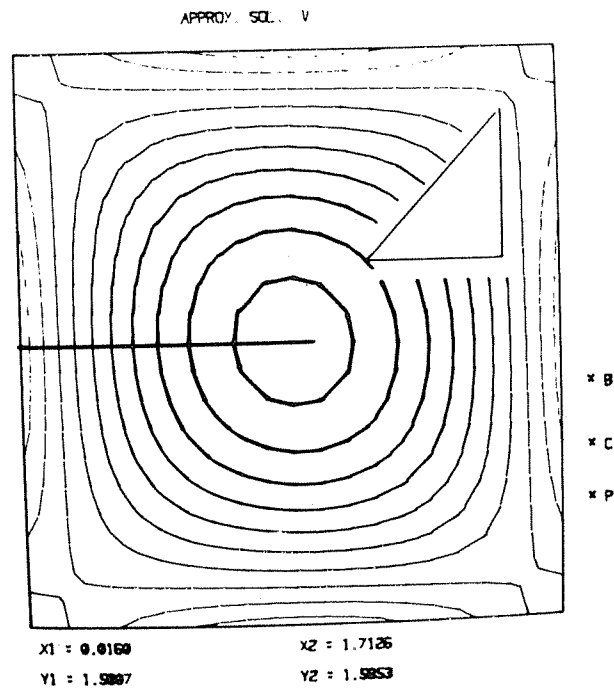


Figure III.19 A Line Superimposed on Approximate Solution

If we wish to examine a cross section of this function in detail, tagging "* L" would place a line on the map as shown in Figure III.19. The x and y coordinates of the two end points of this line $(X1, Y1)$ and $(X2, Y2)$, are determined by four of the variable control dials, and their current positions are shown beneath the contour map. Thus, by manipulating the control dials, a line of the desired length can be positioned anywhere on the scope. Tagging "* C" sends the end coordinates of this line to UPDE where the appropriate function, in this case $v(\hat{\alpha}, x, y)$, is evaluated at 100 points along the line. The results of this calculation are shown in Figure III.20. Again, the end coordinates of the line are shown in the bottom portion of the figure, and we can see that the function goes from a value of $-.1355$ near the outer boundary to $.7711$ near the center. In this manner cross sectional graphs may be obtained from the contour plot of the approximate solution or from any of the contour plots discussed below.

In the next display, Figure III.21, we see a contour map of the error in the partial differential equation, also known as the "defect", given by

$$L_0[v(\hat{\alpha}, x, y)] - f_0 = v_{xx} + v_{yy} + 2 \sin(x) \sin(y).$$

This, of course, is an indication of how well the approximate solution $v(\hat{\alpha}, x, y)$ satisfies the differential equation, and from the figure we

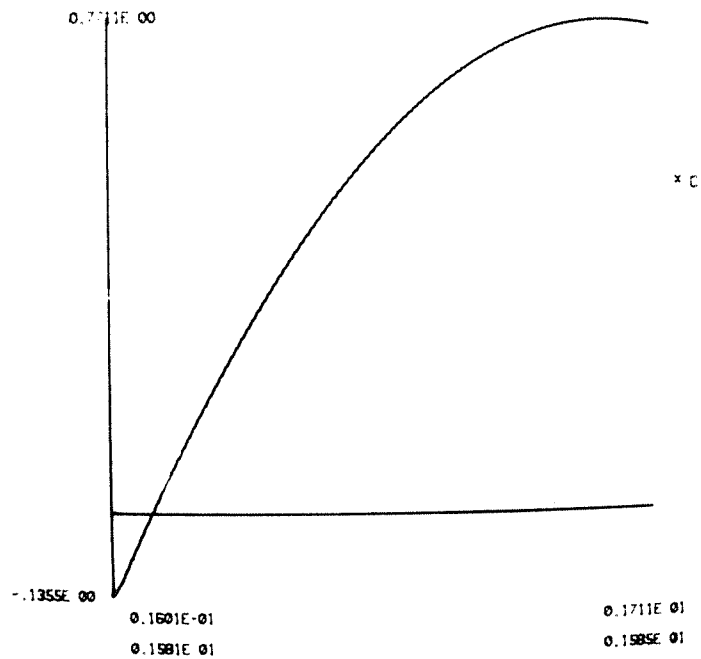


Figure III.20 Cross Sectional Graph of Approximate Solution

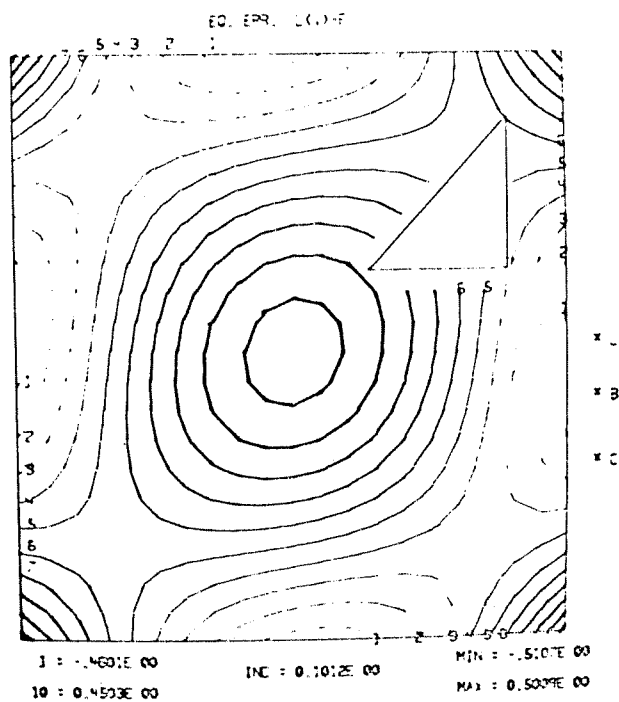


Figure III.21 Error in the Differential Equation

see that the maximum error, which is about .5 in absolute value, is attained at four points along the outside boundary, at the four corners on the outside boundary, and at the center. We also may examine this error along any cross section as illustrated above.

Tagging line AB shows us the display given in Figure III.22 which is the error in the first boundary condition given by $L_1[v(\hat{\alpha}, x, y)] - f_1 = v(\hat{\alpha}, x, y) - 0$ on D_1 . The end coordinates of the line, $(0,0)$ and $(0,\pi)$, are given in the lower portion of the display. The corresponding error on the other three lines could be displayed by tagging the appropriate boundary lines. For instance, tagging line EF in Figure III.21 would automatically give us a plot of the error in the second boundary condition.

The exact solution to (3.1) is $u = \sin(x) \sin(y)$. When the exact solution is known, APDE allows us to enter it and gives a contour plot of the solution error

$$v(\hat{\alpha}, x, y) - u = v - 2 \sin(x) \sin(y)$$

as shown in Figure III.23. The maximum error is .2288 in absolute value and, since $u(\pi/2, \pi/2) = 1$, this represents an error of about 23%. Later in this section we shall see how this error can be significantly reduced by increasing the number of splines.

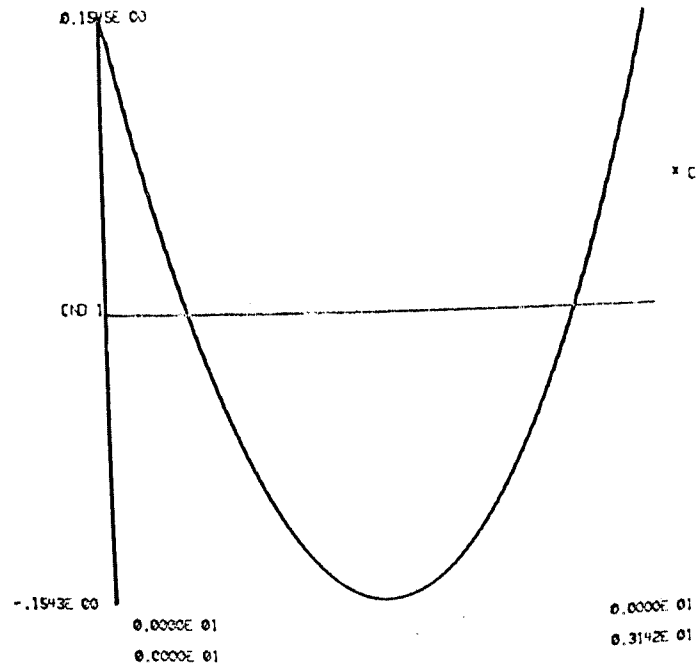


Figure III.22 The Error in the First Boundary Condition

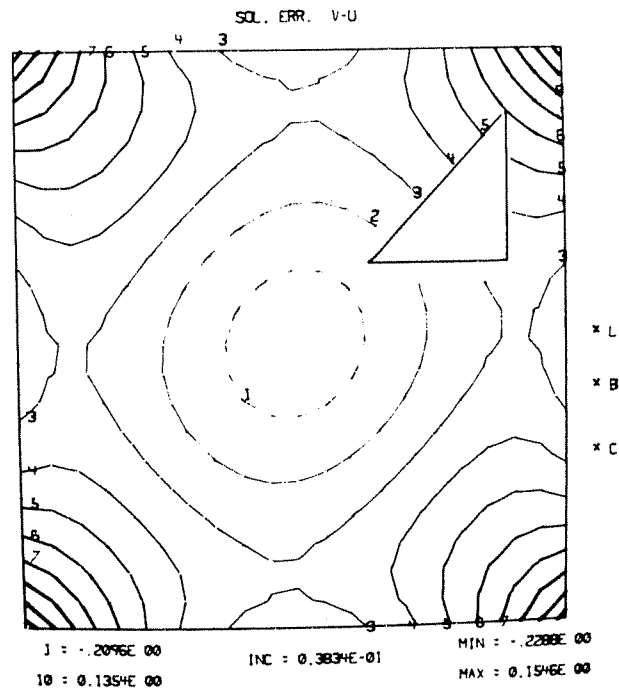


Figure III.23 Solution Error

If we decide to continue working with APDE, we now have a number of options. We may branch to Figure III.1 and enter another problem, or go to Figures III.6 or III.7 where we retain the same differential equation but might define a new domain or grid size, or we may branch to the display shown in Figure III.24. Tagging "* FIRST", "* SECOND", "* THIRD", or "* FOURTH" in this display would allow us to change the weight w_k or the lines on which these boundary conditions are to be imposed. Tagging "* C" takes us to Figures III.16 and III.17 so we may modify the number and degree of the splines, if desired.

Based on the error information displayed a more accurate approximation was desired. Therefore, the above problem was rerun with

FOR WHICH OF THE FOLLOWING BOUNDARY CONDITIONS DO YOU
WISH TO MAKE CHANGES

* FIRST	
* SECOND	
* THIRD	* C
* FOURTH	

Figure III.24

$m_x = m_y = 6$. In order to do this it was only necessary to redefine the parameters in Figure III.17. A contour plot of the resultant solution error is shown in Figure III.25. The maximum error attained is now $.565 \text{ E-}2$ which represents an error of about $1/2\%$. Note that going from $m = 16$ to $m = 36$ has decreased the error from approximately 23% to approximately $1/2\%$. Examples of a variety of problems which are more interesting from a mathematical point of view are given in the next section.

In addition to graphical output, the 1108 prints out the coefficients α_i of the splines and generates numerical data on all of the displayed graphs. This output, which includes a summary of the problem, is printed off-line and is usually available a minute or two after the current iteration has been completed.

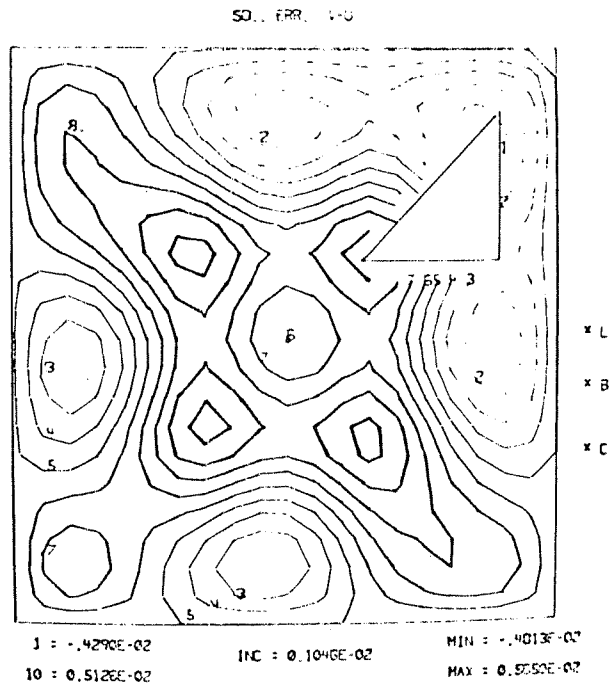


Figure III.25 Solution Error Using 36 Splines.

4. Examples

We shall now give some examples of partial differential equations which have been approximated using PDE. Let us begin by considering the following elliptic boundary value problem:

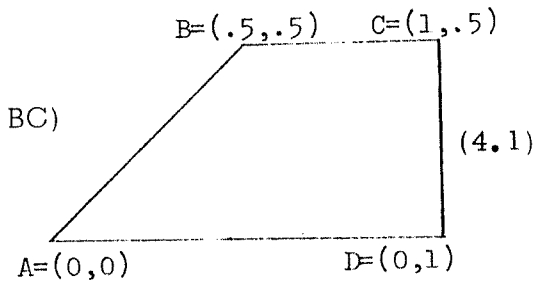
$$L_0[u] \equiv -u_{xx} - u_{yy} = 0 \text{ on } D_0 \text{ (the trapezoid)}$$

$$1) \quad L_1[u] \equiv u = -4x^4 \text{ on } D_1 \text{ (line AB)}$$

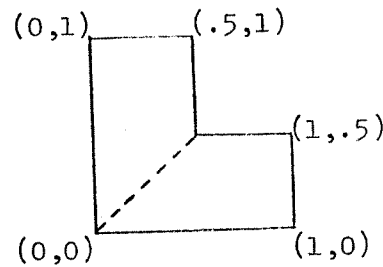
$$2) \quad L_2[u] \equiv u = x^4 - 1.5x^2 + .0625 \text{ on } D_2 \text{ (line BC)}$$

$$3) \quad L_3[u] \equiv u = y^4 - 6y^2 + 1 \text{ on } D_3 \text{ (line CD)}$$

$$4) \quad L_4[u] \equiv u = x^4 \text{ on } D_4 \text{ (line DA).}$$



This problem, which has the exact solution $u = x^4 + y^4 - 6x^2y^2$, is obtained by using symmetry on the L-shaped region shown below.



The differential equation will be approximated at 108 points on D_0 . This set of points is obtained by using a grid of $12 \times 12 = 144$ points, and then erasing the 36 points lying outside the trapezoid. The four boundary conditions will be approximated at 11 points on D_1 , 5 points on D_2 , 10 points on D_3 , and 9 points on D_4 . For this

problem we shall use 25 bi-cubic splines ($m_x = m_y = 5$, $r = 3$) and shall hold these parameters constant as we vary the weights ω_i , $i = 1, \dots, 4$.

Some numerical results are summarized in Table 4.1, where the errors $|L_i[v] - f_i|$ are the maximum errors at a fine grid of points on D_i . They will usually be very close to the actual maximum error on D_i . The first row of Table 4.1 summarizes our results with $\omega_i = 1$, $i = 0, 1, \dots, 4$. The defect in the differential equation $|L_0[v] - f_0|$ is held to a very small value of $.107 \text{ E-}5$. However, the errors in the boundary conditions $|L_i[v] - f_i|$ are much larger, in the range $.01$ to $.03$, and the solution error $|v - u|$ has a maximum of $.033$, or 3.3% . The effects of increasing the boundary weights to $\omega_i = 4$, $i = 1, \dots, 4$, are shown in the second row of Table 4.1, and the third row gives the results after again increasing the weights so that $\omega_i = 16$, $i = 1, \dots, 4$. Notice that in both of these cases the error in the differential equation, which has a small weight in comparison to the boundary conditions, steadily increases whereas the boundary errors go down. Of more importance however, is the fact that the solution error goes down to 1.3% and then to $.5\%$. Figures IV.1 and IV.2 show the defect $L_0[v] - f_0$ and the solution error $v - u$ corresponding to the third row in Table 4.1. We would expect, that as we continue to increase the weights, a point would be reached where

ω_i $i=1, \dots, 4$	$ \bar{u}_0[v] - f_0 $	$ \bar{u}_1[v] - f_1 $	$ \bar{u}_2[v] - f_2 $	$ \bar{u}_3[v] - f_3 $	$ \bar{u}_4[v] - f_4 $	v	$ v - u $
1	.107 E-5	.173 E-1	.106 E-1	.325 E-1	.181 E-1	-.4998 1.033	.3343 E-1
4	.396	.161 E-2	.122 E-1	.110 E-1	.474 E-2	-.508 1.011	.1285 E-1
16	.746	.438 E-3	.143 E-2	.167 E-2	.214 E-2	-.499 1.022	.4932 E-2
32	.762	.181 E-4	.392 E-3	.107 E-2	.417 E-2	-.4993 .9991	.6777 E-2
						-.4992 .9991	.1164 E-2

Table 4.1. Approximation of Elliptic Differential Equation (4.1) and Exact Solution.
See Figures IV.1 through IV.4.

the solution error ceases to improve. This is illustrated by the fourth row of Table 4.1 where, with weights of 32, the solution error has gone up to .68%. From this we can conclude that the best choice of weights for this problem would lie somewhere between 16 and 32. The proper choice of weights for a problem is an important topic which certainly deserves further research.

For a basis of comparison we would like to know the best possible approximation that we can get using the same spline basis. This is obtained by approximating the exact solution directly, as opposed to approximating the differential equation. The approximation obtained, using 127 points in D_0 , is shown in Figure IV.3 and the error is given in Figure IV.4. These results are summarized in the last row of Table 4.1 where we see that the error is .116 E-2, or .12%. Thus, comparing Figures IV.2 and IV.4 shows us that fitting the exact solution directly has given us an approximation that is over four times as good as was obtained using the differential equation with the weights of 16.

We now look at the following variable coefficient wave equation, where the vertical or y axis is to be considered as the time axis:

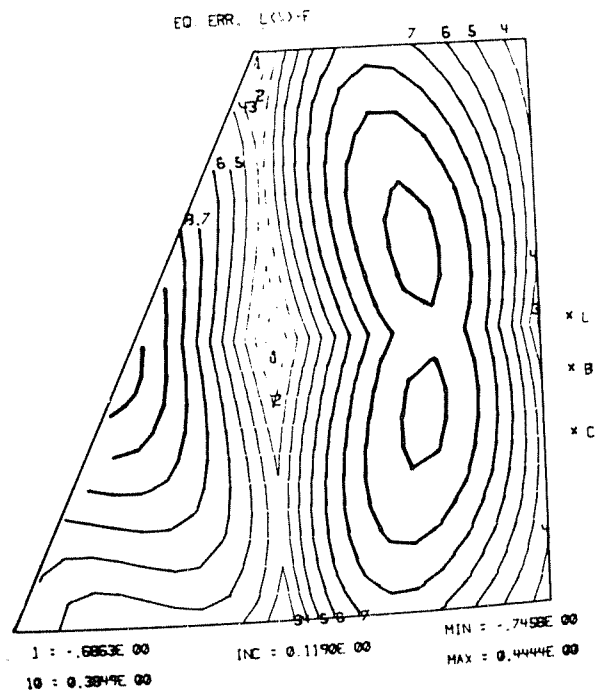


Figure IV.1. Differential Equation Error for Problem (4.1).
See third row of Table 4.1.

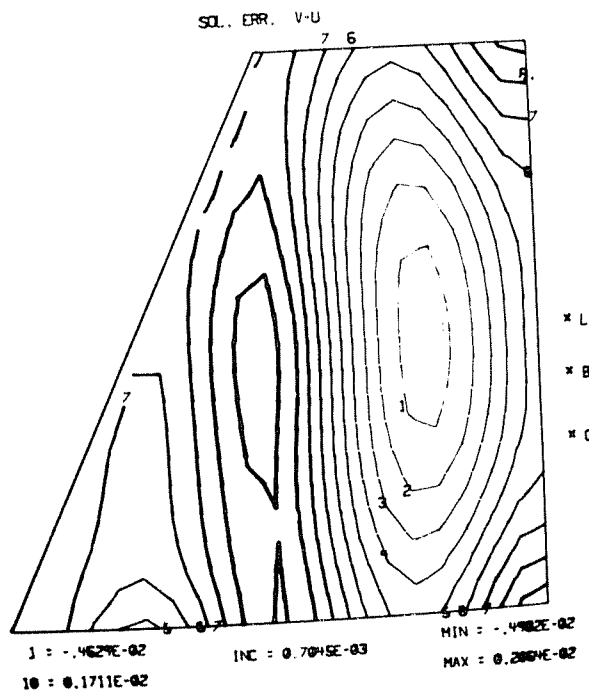


Figure IV.2 Solution Error for Problem (4.1). See third row of Table 4.1.

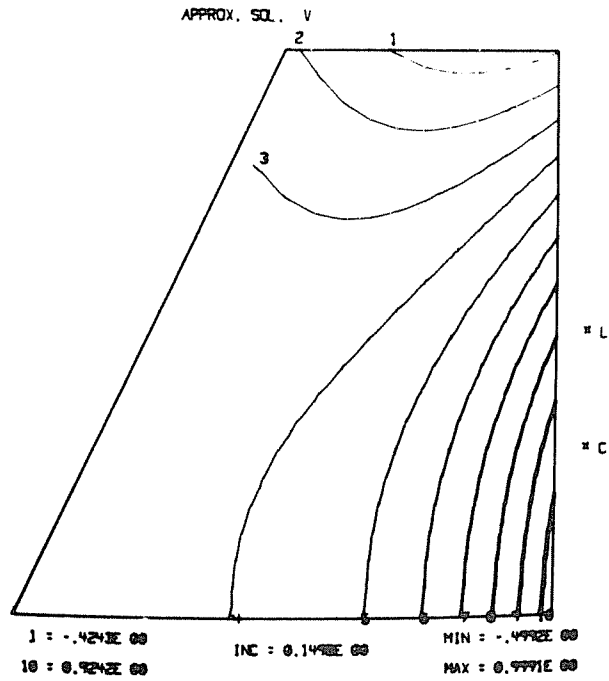


Figure IV.3 Approximate Solution. See last row of Table 4.1.

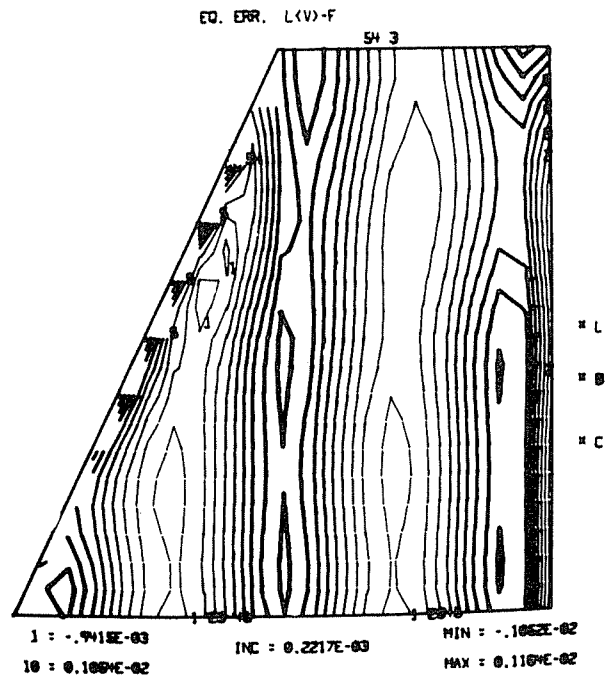


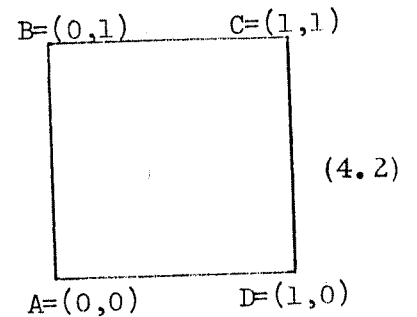
Figure IV.4 Solution Error. See last row of Table 4.1.

$$u_{yy} - (x+1)u_{xx} = 4e^{2xy}(x^2 - (x+1)y^2) \text{ on } D_0$$

$$1) \quad u = 1 \text{ on } D_1 \text{ (lines AB and AD)}$$

$$2) \quad u_y = 2x \text{ on } D_2 \text{ (line AD)}$$

$$3) \quad u = e^{2y} \text{ on } D_3 \text{ (line CD)}$$



The exact solution to this problem is $u = e^{2xy}$. The differential equation will be approximated at 90 points in D_0 not including the three lines AB, AD, and CD. On these lines the boundary conditions will be approximated at a total of 42 points, and the weights will all be held constant at a value of unity.

As shown in the first row of Table 4.2, 25 bi-cubic splines were used in the approximation and resulted in a solution error of .05475, or .74%. The second row shows that after increasing the number of bi-cubic splines to 36 we get a modest decrease in the differential equation and boundary condition errors, and in the solution error. Finally, going to bi-quintic splines but keeping the number of functions constant, gives us a more dramatic decrease in the errors with a resultant solution error of .224 E-2 or .03%. Contour plots of the error in the differential equation and the solution error for this third approximation are given in Figures IV.5 and IV.6. As done in the previous example we now give results obtained after approximating the exact solution directly. The approximate solution and the solution

m_x	m_y	r	$ L_0[v]-x_0^2 $	$ L_1[v]-x_1^2 $	$ L_2[v]-x_2^2 $	$ L_3[v]-x_3^2 $	v	$ v-u $
5	5	3	2.141	.151 E-1	0	.942 E-2	.9846 7.399	.5475 E-1
6	6	3	.836	.314 E-2	0	.377 E-2	.9966 7.386	.3604 E-1
6	6	5	.508 E-1	.115 E-2	0	.113 E-2	.9988 7.391	.2243 E-2
6	6	5					.9999 7.389	.1553 E-3

Table 4.2. Approximation of Hyperbolic Differential Equation (4.2) and Exact Solution.
See Figures IV.5 through IV.8.

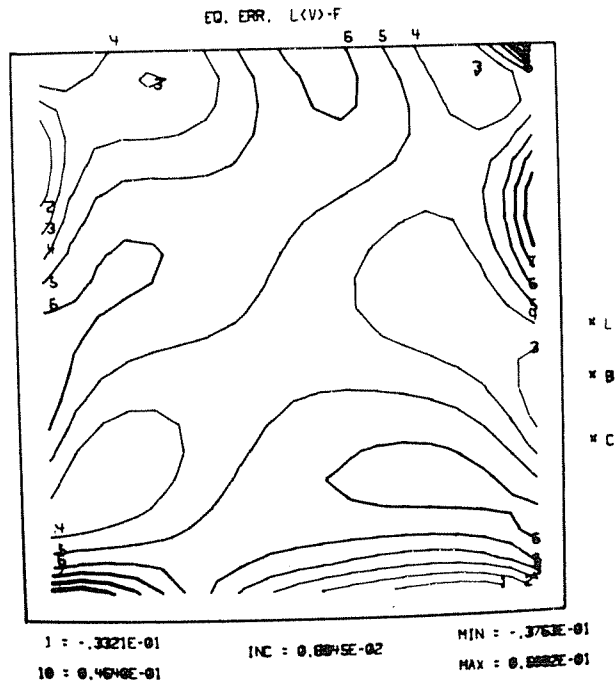


Figure IV.5 Differential Equation Error for Problem (4.2). See third row of Table 4.2.

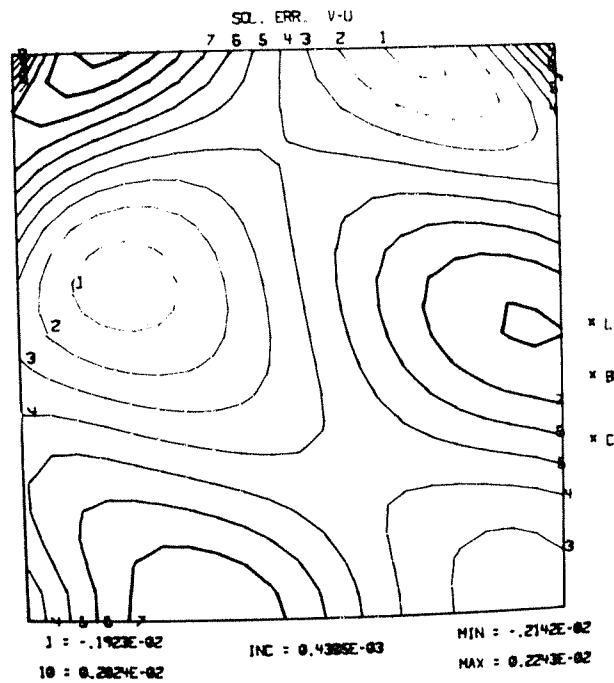


Figure IV.6 Solution Error for Problem (4.2). See third row of Table 4.2.

error are shown in Figures IV.7 and IV.8. This data is summarized in the last row of Table 4.2, where we see that we now have a solution error of $.155 \text{ E-3}$ or $.002\%$. This represents an improvement in the solution error by a factor of over 14, when compared with that shown in Figure IV.6. Also, comparing Figures IV.6 and IV.8 shows that as the amplitude of the error decreases (by a factor of 14), the number of oscillations in each direction increases (by a factor of 2).

We would now like to indicate the amount of human and computer effort required in defining and approximating problems of this type. A rough estimate of this is the amount of time required on the two computers. For the example discussed in Section III, a total of 11 minutes were required on the Adage in order to generate and approximate the problem (this is from log-in to log-out), and during this time the 1108 used only 24 seconds of CPU time. Most of the time on the Adage is spent in defining the problem, and since the problem definition does not have to be re-entered, successive iterations are much faster. Consider the example specified in the first four rows of Table 4.1. These four problems required in total of 25 minutes on the Adage, and this includes the time spent initially defining the problem. On the 1108 about 5 minutes of CPU time were used, but 75% of the CPU time was devoted to just solving the LP problem (an indication that work on a more efficient LP code for this type of problem might be well justified).

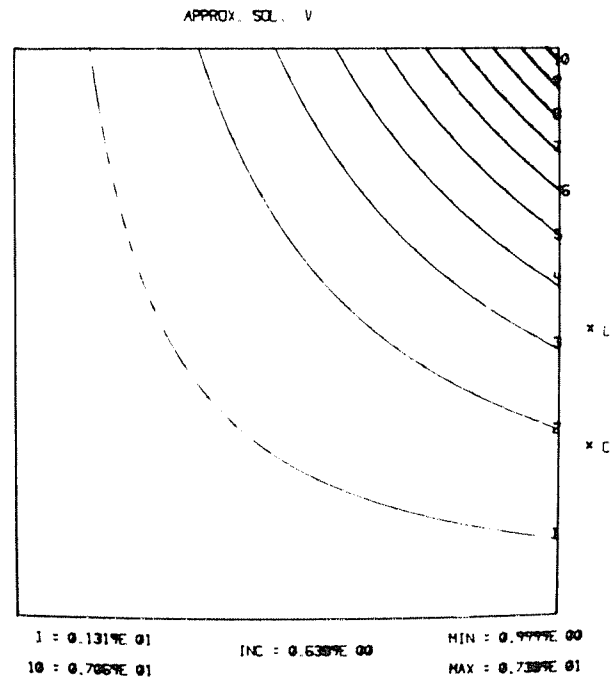


Figure IV.7 Approximate Solution. See last row of Table 4.2.

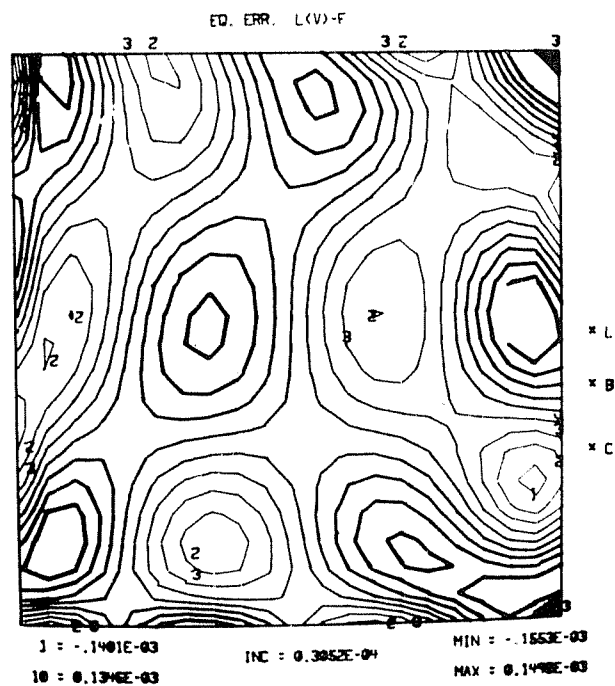


Figure IV.8 Solution Error. See last row of Table 4.2.

ACKNOWLEDGMENT

The author wishes to thank Dr. J. B. Rosen for his sustained guidance during the course of this research.

REFERENCES

1. Ahlberg, J. H. Spline approximation and computer-aided design. In Advances in Computers Vol. 10, F.R. Alt and M. Rubinoff (Eds.), Academic Press, New York, 1970, pp. 275-289.
2. Barrodale, I., and A. Young. Computational experience in solving linear operator equations using the Chebyshev norm. In Numerical Approximation to Functions and Data, J. G. Hayes (Ed.), Athlone Press, London, 1970, pp. 115-142.
3. Bramble, J. H., and A. H. Schatz. Rayleigh-Ritz-Galerkin methods for Dirichlet's problem using subspaces without boundary conditions. Comm. Pure Appl. Math. 23 (1970), pp. 653-675.
4. Collatz, L. Functional Analysis and Numerical Mathematics. Academic Press, New York, 1966.
5. Greville, T. N. E. Introduction to spline functions. In Theory and Applications of Spline Functions, T. N. E. Greville (Ed.), Academic Press, New York, 1969, pp. 1-35.
6. Klerer M. and J. Reingelds (Eds.). Interactive Systems for Experimental Applied Mathematics. Academic Press, New York, 1968.
7. LaFata, P., and J. B. Rosen. An interactive display for approximation by linear programming. Comm. ACM 13, 11 (Nov. 1970), pp. 651-659.
8. LaFata, P. An interactive system for generalized approximation. Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, 1971.
9. Prince, M. D. Interactive Graphics for Computer-aided Design. Addison-Wesley, 1971.
10. Rabinowitz, P. Applications of linear programming to numerical analysis. SIAM Rev. 10 (1968), pp. 121-159.
11. Rosen, J. B. Approximate solution and error bounds for quasi-linear boundary value problems. SIAM J. Numer. Anal. 7, 1 (March 1970), pp. 80-103.

12. Rosen, J. B. Minimum error bounds for multidimensional spline approximation. Tech. Rept. No. 100, Computer Sciences Dept., Univ. of Wis., Madison, October 1970.
13. Rosen, J. B., and P. LaFata. Interactive graphical spline approximation to boundary value problems. In Proc. ACM '71, Association for Computing Machinery, New York, 1971, pp. 466-481.

BIBLIOGRAPHIC DATA SHEET	1. Report No. WIS-CS-71-138	2.	3. Recipient's Accession No.
4. Title and Subtitle An Interactive Graphical System for the Approximation of Partial Differential Equations		5. Report Date October 1971	
7. Author(s) Paul S. LaFata		6.	
9. Performing Organization Name and Address Computer Science Department University of Wisconsin 1210 West Dayton Street Madison, Wisconsin 53706		8. Performing Organization Rept. No.	
		10. Project/Task/Work Unit No.	
		11. Contract/Grant No. GJ-0362	
12. Sponsoring Organization Name and Address National Science Foundation Washington, D. C. 20550		13. Type of Report & Period Covered	
		14.	
15. Supplementary Notes			
16. Abstracts An interactive graphical system which can be used to approximate bivariate functions or to obtain approximate solutions to partial differential equations is described. The function or differential equation is entered symbolically, so the system may be used by someone with no programming experience. The basic mathematical technique is to assume the approximate solution is a linear combination of user selected functions, and to use linear programming to determine the coefficients of these functions. The approximate solution and error information can be graphically displayed in several different ways, and based on these results various parameters may be modified on line so as to obtain a better approximation.			
17. Key Words and Document Analysis. 17a. Descriptors interactive graphics, approximation, partial differential equations			
17b. Identifiers/Open-Ended Terms			
17c. COSATI Field/Group			
18. Availability Statement Available to public.		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 50
		20. Security Class (This Page) UNCLASSIFIED	22. Price

