EVALUATION OF NUMERICAL METHODS FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

E.N. Houstis, R.E. Lynch, T.S. Papatheodorou and J.R. Rice
Computer Science Department
Purdue University
West Lafayette, Indiana 47907

CSD TR 204
October, 1976
EVALUATION OF NUMERICAL METHODS FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

E.N. Houstis, R.E. Lynch, T.S. Papatheodorou and J.R. Rice
Computer Science Department
Purdue University
CSD TR 204
October, 1976

CONTENTS

I. Statement of the Problem and Procedures, Conclusions.
II. Comparison of Standard Finite Differences and Collocation with Hermite Cubics.
   1. The Numerical Methods and Problem Set
   2. Results of the Comparisons
   3. Conclusions
III. Comparison of Collocation, Galerkin and Least Squares
   1. The Methods
   2. Results of the Comparisons
   3. Conclusions
IV. Three Observations
   1. Unequal Mesh Spacing for Collocation
   2. Additional Accuracy at the Mesh Nodes for Collocation
   3. Accuracy Depends on the Operators as well as the Solution
V. Comparison with Previous Work

References
Appendix 1: 17 Graphs of the Comparison Data for 17 Problems
Appendix 2: Synopsis of the Numerical Methods
Appendix 3: The Interpolation of Boundary Conditions for Collocation
Appendix 4: The solution of Problem 17.

ABSTRACT

We systematically evaluate four methods for solving two-dimensional, linear elliptic partial differential equations on general domains. The four methods are: standard finite differences; collocation, Galerkin and least-squares using Hermite cubic piecewise polynomials. Our test set of 17 problems ranges from simple to moderately complex. The principal conclusion is that collocation is the most efficient method for general use. Standard finite differences is sometimes more efficient for very crude accuracy (where efficiency is not important anyway) but it is also sometimes enormously less efficient even for very modest accuracy. The accuracy of the Galerkin and least-squares methods is sometimes better than collocation, but the extra cost always negates this advantage for our problems.
I. STATEMENT OF THE PROBLEM AND PROCEDURES, CONCLUSIONS.

Our approach to evaluating numerical methods for partial differential equations has already been outlined in Houstis, et al [1975]. This approach is a specific instance of the general framework presented by Rice [1976a]. Briefly this approach is to first choose a sample set of problems from the domain of interest. The domain here is linear, second order elliptic partial differential equations which are somewhat "general". That is, they have various complications (variable coefficients, curved domains, reentrant corners, etc.) that are typical in applications and which prevent the straightforward use of specialized methods or theories. One next selects some solution methods (four in this paper) and criteria of performance (accuracy achieved, execution time and memory used) and finally one applies the methods to the sample set of problems while measuring the performance criteria.

The cost of solving partial differential equations forces a small sample set (17 problems here) and thus the reliability of the evaluations is not as high as we would like. Nevertheless, most of the phenomena observed here are quite consistent over the problem set which suggests that the probability of this being the result of chance is quite low.

One key to validity of an evaluation such as this is the precise definition of the problems, methods and measures of performance. The sample problem set is presented in the next section. The numerical methods are briefly discussed in Sections II and III and a more detailed synopsis of them is given in Appendix 2.
A common weakness of previous efforts of this type is the lack of precision and information about the numerical methods. It is well known that it is insufficient to simply state "Method X was used". Variations in the implementation of Method X affect the performance measures by factors of 2, 10 or 1000. We believe that we have implemented all the numerical methods used in a way that gives close to maximum performance. We have particularly striven to be "fair" to each method and have not used special techniques (e.g. assembly language code) for one in order to enhance its performance relative to the others.

We summarize our procedure and conclusions as follows:

Problem Class: Second order linear elliptic partial differential equations of general nature i.e. some complication present in coefficients, domain or solution.

Solution Requirements: Moderate accuracy (1 to 3 digits correct) achievable "in core" (60,000 words or less of memory needed).

4 Numerical Methods: Standard Finite Differences; Collocation, Galerkin and Least Squares using piecewise cubic polynomials (Hermite cubics).

Criteria of performance (efficiency): Execution time for a given accuracy. Accuracy is the maximum error divided by the size of the solution and is usually measured in decimal digits.

Conclusions:
1. There is normally a "cross-over point" at low accuracy beyond which Collocation is more efficient than Standard Finite Differences. Even when finite differences is more efficient, it is by a small amount while Collocation is sometimes dramatically more efficient than finite differences.
2. There is practically no difference at all between Galerkin and Least Squares in performance. They tend to be slightly more accurate than Collocation but are very much less efficient because of the increased work to compute the coefficients in the matrix problem to be solved.

II. COMPARISON OF STANDARD FINITE DIFFERENCES AND COLLOCATION WITH HERMITE CUBICS.

II.1 The Numerical Methods and Problem Set. The first comparison made in this paper is between the standard finite difference method (5-point star) and collocation with Hermite cubics. See Appendix 2, Fix and Strang [1973] and Collatz [1966] for detailed information on these methods. Simply stated, in collocation the coefficients of the approximate solution are chosen to satisfy exactly the partial differential equation and boundary conditions at selected points.

In simple situations with a uniform mesh length of $h$, the finite difference method is second order, $O(h^2)$ and collocation is fourth order, $O(h^4)$. Thus, asymptotically in these situations, as the accuracy increases, collocation becomes more efficient than standard finite differences. This suggests the existence of a cross-over point in the performance where collocation becomes more efficient. One of our objectives is to ascertain whether simple collocation applies to more general problems and to determine the expected location of the cross-over point. The operators, domains, boundary conditions and true solutions for the 17 problems we used are given in Table 1. The first 8 were previously considered by us in Houstis et al, [1975]. We give additional information about some of them:

Prob. 2/3. Torsion in a bimetal shaft, Ely and Zienkiewicz [1960]. The shear modulus $G$ is a step function with $G_1/G_2 = 3$ (see Figure 1a). We have replaced the step by a short interval (length = 0.001) where a cubic polynomial blends the two values of $G$ smoothly. We measure accuracy
2 geometry and boundary conditions for problems 2, 3, 14 and 17.
Problem 16 uses the geometry of (c) with the boundary condition \( u = g \) everywhere.

(a) \( u = 0 \)
\[ G = G_1 \quad G = G_2 \]
\[ u = 0 \] (Prob 2)
\[ u = 1 \] (Prob 3)
\[ u_N = 0 \]
\[ u_N = 0 \]

(b) \( u = 2 \)
\[ u = y \]
\[ u = y \]
\[ u = 0 \]
\[ u = 0 \]

(c) \( u = 0 \)
\[ u = g \]
\[ u = 100 \]
\[ u = 0 \]
\[ u = 0 \]
\[ u = 0 \]
\[ u = g \]
Table 1. The 17 problem space sample used in this paper. The letters $f$ and $g$ denote functions whose values are determined to make the problem have the specified true solution. The references are to papers where the problem or a closely related one has been considered.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Partial Differential Equation Operator</th>
<th>True Solution</th>
<th>Size of Solution</th>
<th>Domain</th>
<th>Boundary Conditions</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(e^{xy}u_x)_x + (e^{-xy}u_y)_y - \frac{u}{1+x+y} = f$</td>
<td>$u = e^{xy} \sin(\pi x) \sin(\pi y)$</td>
<td>1.3</td>
<td>Unit Square</td>
<td>$u=0$</td>
<td>[9]</td>
</tr>
<tr>
<td>2</td>
<td>$(\frac{1}{6} u_x)_x + (\frac{1}{6} u_y)_y = f$ with $f$ = -26 or 0</td>
<td>$u$ is unknown</td>
<td>0.87 or 0.8</td>
<td>See Fig. 1a</td>
<td>See Fig. 1a</td>
<td>[7] [9]</td>
</tr>
<tr>
<td>3</td>
<td>$u_{xx} + u_{yy} = f$</td>
<td>$u = (e^x+e^y)/(1+xy)$</td>
<td>7.6</td>
<td>Ellipse</td>
<td>$u=g$</td>
<td>[9]</td>
</tr>
<tr>
<td>4</td>
<td>$u_{xx} + u_{yy} = 0$</td>
<td>$u = \tan^{-1}(y/x)$</td>
<td>2.6</td>
<td>Circle</td>
<td>$u_{N}=g$</td>
<td>[9]</td>
</tr>
<tr>
<td>5</td>
<td>$u_{xx} + (1+y^2)u_{yy} - u_x - (1+y^2)u_y = f$</td>
<td>$u = e^{x+y} + (x^2 - x)^2 \log(1+y^2)$</td>
<td>7.4</td>
<td>Unit Square</td>
<td>$u-u_{N}=0$</td>
<td>[9]</td>
</tr>
<tr>
<td>6</td>
<td>$u_{xx} + u_{yy} = -6xye^x e^y (xy + x + y - 3)$</td>
<td>$u = 3e^x e^y (x - x^2)(y - y^2)$</td>
<td>0.58</td>
<td>Unit Square</td>
<td>$u=0$</td>
<td>[9] [14]</td>
</tr>
<tr>
<td>7</td>
<td>$u_{xx} + u_{yy} = f$</td>
<td>$u = x^{5/2}y^{5/2} - xy^{5/2} - x^{5/2}y + xy$</td>
<td>0.1</td>
<td>Unit Square</td>
<td>$u=0$</td>
<td>[9]</td>
</tr>
<tr>
<td>Problem</td>
<td>Partial Differential Equation Operator</td>
<td>True Solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 4u_{xx} + u_{yy} - 64u = f )</td>
<td>( u = 4(x^2 - x)(\cos(2\pi y) - 1) ),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>( u_{xx} + u_{yy} - [100 + \cos(3\pi x)\sin(2\pi y)]u = f )</td>
<td>( u = [5.4 - \cos(4\pi x)]\sin(\pi x)(y^2 - y) [5.4 - \cos(4\pi y)] \times [1/(1 + \phi^4) - 1/2] )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/12</td>
<td>( u_{xx} + u_{yy} - 100u = (u^2 - 100)\cosh y/\cosh u ) with ( u = 10 ) or 20</td>
<td>( u = \cosh 10x/\cosh 10 + \cosh uy/\cosh u ),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>( u_{xx} + u_{yy} = f )</td>
<td>( u = \phi(x) \ast \phi(y) ), see text,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>( u_{xx} + u_{yy} = f )</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>( u_{xx} + u_{yy} = f )</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>( u_{xx} + u_{yy} = 2e^{x+y} )</td>
<td>( u = e^{x+y} ),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>( u_{xx} + u_{yy} = f )</td>
<td>4.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Domain} \quad \text{Boundary Conditions} \quad \text{References} \]

\begin{align*}
\text{Problem} & \quad \text{Size of Solution} \quad \text{Domain} \quad \text{References} \\
9 & \quad 2.0 \quad \text{Unit Square} \quad u=0 \quad [10] \\
10 & \quad 3.2 \quad \text{Unit Square} \quad u=0 \\
1/12 & \quad 2.0 \quad \text{Unit Square} \quad u=g \\
13 & \quad 1.0 \quad \text{Unit Square} \quad u=g \\
14 & \quad 2.0 \quad \text{See Fig. 1b} \quad \text{See Fig. 1b} \quad [6] \\
15 & \quad 0.6 \quad \text{Unit Square} \quad u=0 \quad [17] \\
16 & \quad 4.9 \quad \text{See Fig. 1c} \quad u=g \quad [17] \\
17 & \quad 100.0 \quad \text{See Fig. 1c} \quad \text{See Fig. 1c} \quad [17] \\
\end{align*}
here by comparing with a numerical solution we have computed which we believe is much more accurate than the ones considered in this paper.

Prob. 4. The ellipse is centered at (0,0) with major and minor axes of 2 and 1. By symmetry only a quarter of the elliptical region was used in the computation.

Prob. 5. The circle has radius 0.5 and center at (0.5,0.5). The solution is uniquely determined by imposing the additional condition \( u(0,0.5) = 0 \).

Prob. 8. The true solution has a discontinuity in the "2.5" derivative.

Prob. 10. This is a version of a problem from stratospheric physics, see McDonald et al [1974].

Prob. 11/12. These problems are of boundary layer type; the square is centered at the origin and has side 2. Symmetry was not used.

Prob. 13. The product solution \( \phi(x) \phi(y) \) has a steep slope (or wave front) along a right angle at the center of the domain. We have

\[
\phi(x) = \begin{cases} 
1 & x \leq .35 \\
p(x) & .35 < x < .65 \\
0 & .65 \leq x 
\end{cases}
\]

where \( p(x) \) is a quintic polynomial determined so that \( \phi(x) \) has two continuous derivatives.

Prob. 14. This problem is similar to that of steady flow past a sphere, Desai and Abel [1972]. The true solution satisfies the same boundary conditions and has the same shape as the solution of the physical problem.

Prob. 15. The solution has a sharp peak at the center of the square and it is very small for \((x-.5)^2+(y-.5) > .01\).

Prob. 16/17. This problem is derived from that of heat flow in the concrete shield of a nuclear reactor, see Zienkiewicz and Cheung [1965].
Problem 16 only has the geometry and operator of the real problem. The true solution of Problem 17 (see Appendix 4) is a complicated function which exhibits the same shape (including small singularities at the three reentrant corners) and satisfies the same boundary conditions (except along $x=0$ and $y=0$) as the solution of the physical problem.

Problems 1, 7, 8, 9, 13 and 15 are separable and all the operators except for Prob. 6 are formally self-adjoint.

II.2 Results of the Comparisons. The data obtained are presented in two forms. In Appendix 1 we give a set of 17 graphs of the accuracy achieved versus computer time used. For both methods the error is measured only at the nodes of the grid used. For most problems we have also measured the error at many more points in the domain and this sometimes gives a considerably different result. This is discussed in more detail in Section IV. We used a CDC 6500 whose long word length gives ample insulation from round-off errors in these calculations.

In Table 2 we tabulate the cross-over points for all 17 problems. This is expressed both in terms of accuracy measured in digits as $\log(\text{max error}/\text{solution size})$ and the number $N$ of subdivisions in each variable. For the non-rectangular regions we give an approximate "equivalent" value of $N$ which would give about the same number of unknowns, if the region were rectangular.

We see from Table 2 that the cross-over points range from 0 to 4 digits with 2 as a median value. One of the high cross-over points comes from Problem 16 where high accuracy is obtained by very coarse meshes. Let $N_F$ and $N_C$ denote the values of $N$ at the cross over point for finite differences and collocation, respectively. There is a fairly consistent pattern in the relationship of the values of $N_F$ and $N_C$, namely $\sqrt{N_F/N_C}$ is about 1. The value of $N_C$ is small (from 1 to 6 with 3 as median) for all cases.
Table 2. Tabulation of the cross-over points for 17 problems. The accuracy (in digits) and numbers $N_F$ and $N_C$ of grid lines is given for the comparison of Standard Finite Difference and Collocation with Hermite Cubics.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Digits = log(max error/solution size)</th>
<th>$N_F$</th>
<th>$N_C$</th>
<th>$\sqrt{N_F} / N_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.8</td>
<td>5</td>
<td>2</td>
<td>1.12</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>13</td>
<td>4</td>
<td>0.90</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>12</td>
<td>3</td>
<td>1.15</td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
<td>12</td>
<td>4</td>
<td>0.87</td>
</tr>
<tr>
<td>5</td>
<td>1.9</td>
<td>6</td>
<td>2</td>
<td>1.22</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>1.8</td>
<td>5</td>
<td>1</td>
<td>2.23</td>
</tr>
<tr>
<td>8</td>
<td>4.0</td>
<td>5</td>
<td>2</td>
<td>1.12</td>
</tr>
<tr>
<td>9</td>
<td>3.0</td>
<td>9</td>
<td>4</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>8</td>
<td>3</td>
<td>0.94</td>
</tr>
<tr>
<td>11</td>
<td>2.2</td>
<td>13</td>
<td>6</td>
<td>0.60</td>
</tr>
<tr>
<td>12</td>
<td>1.3</td>
<td>9</td>
<td>4</td>
<td>0.75</td>
</tr>
<tr>
<td>13</td>
<td>1.3</td>
<td>15</td>
<td>5</td>
<td>0.77</td>
</tr>
<tr>
<td>14</td>
<td>3.6</td>
<td>17</td>
<td>5</td>
<td>0.82</td>
</tr>
<tr>
<td>15</td>
<td>1.2</td>
<td>15</td>
<td>4</td>
<td>0.97</td>
</tr>
<tr>
<td>16</td>
<td>4.1</td>
<td>16</td>
<td>4</td>
<td>1.00</td>
</tr>
<tr>
<td>17</td>
<td>1.8</td>
<td>20</td>
<td>6</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Our results here differ in some cases from those published earlier, Houstis et al [1975]. The efficiency of both programs has been improved but their relative efficiency has not changed much. In our earlier paper we measured the error at many points over the entire domain (bilinear interpolation was used to extend the finite difference solutions). The few noticeable differences from the earlier data are due to this change in error measurement. We also previously gave data on memory usage as well as execution time. We have omitted memory data here as the cross-over points for memory are somewhat the same as for execution time (this is true also for the new problems introduced in this paper).
We timed separately the formation and the solution of the linear equations. Both finite differences and collocation are very similar in the breakdown of execution time as seen in Table 3.

Table 3. Sample data on the breakdown of execution time between formation and solution of the linear equations.

<table>
<thead>
<tr>
<th>Prob.</th>
<th>Method</th>
<th>Formation</th>
<th>Solution</th>
<th>Ratio of Formation/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collocation, N=4</td>
<td>0.25 sec</td>
<td>0.48 sec</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>Finite Differences, N=10</td>
<td>0.25</td>
<td>0.56</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>Collocation, N=8</td>
<td>1.0</td>
<td>4.5</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>Finite Differences, N=17</td>
<td>0.9</td>
<td>3.6</td>
<td>0.20</td>
</tr>
<tr>
<td>10</td>
<td>Collocation, N=8</td>
<td>1.4</td>
<td>4.4</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>Finite Differences, N=17</td>
<td>1.2</td>
<td>3.4</td>
<td>0.26</td>
</tr>
</tbody>
</table>

The solution of the matrix equation was always by Gauss elimination (frontal or profile version) and it is possible that iterative methods or nested dissection would be significantly more efficient. Indeed, this is known to be true for certain simple problems and finite differences. However, we are concerned with problems with some complexity (even though we included some simple examples in our sample) and there the theoretical relationship between iterative methods and Gauss elimination is unknown.

Iterative methods also normally involve choosing one or more parameters and that could be very delicate for complex problems. Thus we must leave the question of the impact of using iterative methods on these problems as an open question for future research. The few comparisons that we are aware of have various defects that leaves the situation inconclusive in our minds.

II.3 Conclusions. A study of Table 2 and the graphs in Appendix 1 shows that collocation becomes more efficient than standard finite differences at rather low accuracies and/or small values for N. Furthermore, when finite differences are more efficient, it is by a small margin whereas collocation
is often dramatically more efficient than finite differences. These results cover a reasonably broad range of two-dimensional linear elliptic problems and show that there is no reason from the point of view of efficiency to use the standard finite difference methods for this class of problems.

It is also relevant to note that in practical problems one must almost always compute solutions to higher accuracy than actually required. That is to say, the only reliable ways to be certain that one has an error of, say, 5% (or less) involve computing a solution accurate to 1% or better. This is especially the case for low accuracy requirements (e.g. 1-10% error).

III. COMPARISON OF COLLOCATION, GALERKIN AND LEAST SQUARES.

III.1 The Methods. In all three of these methods we use Hermite cubic polynomials as approximations. More specific details are given in Appendix 2 but there are two facts worth noting here. First, both the Galerkin and Least Squares methods involve the evaluation of integrals and these have been estimated by using 9 point quadrature in each grid rectangle based on the tensor product of the 3 point Gauss rule. All the information from the equation must be evaluated at 9 points, this compares with 4 points needed for collocation in each element (grid rectangle).

Second, the Galerkin and Least Squares methods were implemented only for the case where the boundary conditions can be exactly satisfied by choosing the Hermite cubic basis appropriately. This restriction makes them intrinsically less flexible and should give them an advantage over collocation whenever they are applicable. To offset this advantage we used the same Hermite cubic basis for collocation on those problems where all three methods are compared. In complex problems it can be very difficult (and tedious) to modify the original problem into one where the boundary conditions can be satisfied exactly by piecewise cubic polynomials.
There are only six problems (1, 7, 8, 9, 10, and 15) where Galerkin and Least squares could be applied, but the results are so consistent that this number seems sufficient to draw general conclusions.

III.2 Results of the Comparisons. The graphs given in Appendix 1 for these six problems show the data for all three methods. An examination of these graphs shows that there is rarely a significant difference between the Galerkin and Least Squares method. Table 4 gives a sample of some additional typical data for comparing the collocation and Galerkin methods.

One sees from Table 4 that collocation is always faster for equal accuracy. The advantage decreases as N increases and an operations count shows that eventually the Galerkin method is faster. This is because eventually most of the time is spent in solving the linear system and the Galerkin system is symmetric and hence can be solved twice as fast as the nonsymmetric collocation system. The timing data given in Table 4 is compatible with an operations count analysis for these two methods. One also sees for a fixed set of elements (grid) that collocation is sometimes much less accurate than Galerkin and never more accurate. However, the graphs show that the accuracy advantage of Galerkin never compensates for its speed disadvantage in these cases. One may compare accuracy from the graphs by noting that the last point plotted for each method has the same number of elements.

Note that Problem 10 involves fairly complicated functions in the differential operator and that this has a large negative effect for the Galerkin and Least Squares methods.

III.3 Conclusions. We see that collocation is a more general method and that it is also more efficient than Galerkin or Least Squares. Collocation is more delicate to apply because the boundary collocation points must be selected carefully for complicated regions. See Appendix 3. Thus collocation
is the method of choice among these three for the class of problems represented here.

Table 4. Selected data comparing collocation and Galerkin for six problems. Times are given in seconds.

<table>
<thead>
<tr>
<th>Factors of</th>
<th>Time Break Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 to 12</td>
</tr>
<tr>
<td>7</td>
<td>3 to 6</td>
</tr>
<tr>
<td>8</td>
<td>2.5 to 8</td>
</tr>
<tr>
<td>9</td>
<td>3 to 7</td>
</tr>
<tr>
<td>10</td>
<td>6 to 15</td>
</tr>
<tr>
<td>15</td>
<td>5 to 10</td>
</tr>
</tbody>
</table>

IV. THREE OBSERVATIONS.

IV.1 Unequal Mesh Spacing for Collocation. There are two disadvantages to collocation compared to standard finite differences: (1) It is not well known, (2) its implementation is more complicated. The extra complexity (which is not great) of collocation partially stems from its greater flexibility. One manifestation of this is that unequal mesh spacings can be used with no extra difficulty, no loss in accuracy and a negligible increase in computation. By no loss of accuracy we mean that collocation remains a fourth order method as contrasted to standard finite differences where unequal mesh spacing reduces the order from second to first.

In fact, unequal mesh spacing can dramatically increase the accuracy of collocation solutions and often one can see (with little trouble) a reasonable mesh to use. Several examples of this occur among the 17 problems.
considered here, including Prob. 13 (wave front on a right angle) and
Prob. 15 (sharp peak at center). We solved both of these problems with
unequally spaced meshes and the resulting improvements are tabulated
in Table 5. The unequally spaced meshes for these examples were chosen
in what seemed a plausible way, but no systematic attempt was made to
optimize the mesh.

Table 5. Illustration of the possible improvement in accuracy of the collocation
method by using an unequally spaced mesh.

<table>
<thead>
<tr>
<th>Case</th>
<th>Equally Spaced Mesh</th>
<th>Unequally Spaced Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N=6</td>
<td>1.5*10^{-2}</td>
<td>1.8*10^{-3}</td>
</tr>
<tr>
<td>N=8</td>
<td>7*10^{-2}</td>
<td>4.1*10^{-4}</td>
</tr>
<tr>
<td>Prob 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N=3</td>
<td>.57</td>
<td>.29</td>
</tr>
<tr>
<td>N=6</td>
<td>.16</td>
<td>.06</td>
</tr>
<tr>
<td>N=8</td>
<td>.08</td>
<td>.026</td>
</tr>
</tbody>
</table>

IV.2 Additional Accuracy at the Mesh Nodes for Collocation. For general
collocation there is a phenomenon called super convergence, see deBoor and
Swartz [1974] where the order of accuracy at the mesh nodes is higher than
elsewhere. However, in theory this phenomenon does not occur when using
cubic polynomials. Nevertheless, we observed substantially improved accuracy
at the nodes for some problems while there was none for some others. For
two problems there was a constant increase in the accuracy at the nodes:
a factor of 4 for Prob 7 and 15 for Prob 4. In some other problems (e.g. 8,
10, 11, and 13) there was a more erratic factor of increase, but it exceeded
4 in some case of each of these problems. No such phenomenon occurred for
the Least Squares or Galerkin methods.

There is a plausible explanation of this as follows: The nature of the theoretical error term for collocation is different at the mesh nodes than that at other points, but the use of cubic polynomials results in the same order of accuracy for both cases. However, for some problems the coefficient of the principal error term at the nodes might be significantly smaller than that of the general error term. This could account for the phenomenon that we observe.

IV.3 Dependence of Accuracy on the Nature of the Operator as well as the Solution. It is obvious that the difficulty of obtaining a numerical solution of a partial differential equation depends on the nature of the differential operator as well as the nature of its solution. This fact may be overlooked as the theory places heavy emphasis on the nature of the solution. The effect of the operator, however, can be quite significant. For example, compare the widely varying results that are obtained for Problems 6, 7 and 16 whose solutions are nearly the same. On the other hand, Problems 1, 7 and 9 have very similar results as one would guess from the fact that the differential operators and boundary conditions are similar in nature and all three have very well-behaved solutions. We have considered several sets of different problems which all have the same solution and have seen a very wide range of difficulty in obtaining the same function from problems with different operators.
V. COMPARISON WITH PREVIOUS WORK.

There has been little effort on systematic comparisons of different methods for solving partial differential equations; our previous paper [Houstis et al, 1975] was one of the first. There have been a number of abstract comparisons based on asymptotic rates of convergence and asymptotic operation counts for the solution of linear systems of equations. See [Rice, 1976] and [Birkhoff and Fix, 1971] for a large number of examples of this analysis and references to earlier work. Experience has shown that operation counts are reliable for estimating the efficiency of solving linear systems of equations. For iterative methods one must take extreme care to terminate the iteration at a level compatible with the discretization error of the method. This point is commonly overlooked and invalidates some otherwise interesting comparison studies.

The usefulness of asymptotic rates of convergence as guides to the efficiency of numerical methods for elliptic problems is still open to question. Specifically, it is not known how reliable these rates are as guides for the moderate accuracy requirements of typical applications. Discussions of this question is given in the last section of Strang and Fix [1973] (there asymptotic rates are reliable guides for 3 example problems), in Birkhoff and Fix [1974] and in Swartz [1974] where several different order methods are compared.

Roache [1972] has a section entitled "Remarks on Evaluating Methods" (pp. 109-112) and he strongly favors simple, low order methods and describes the performance of higher order methods as "disappointing". He supports the conclusions with citations of 12 papers, half of which have no relevant material on the question of the performance or comparison of methods. Most of those papers which involve shock wave and turbulence computations suggest that low order methods are the best of the methods used. However,
we (and some of the authors) interpret these papers' results on smoother problems differently than Roache. One paper explicitly states that first order methods compare poorly and a third order method gives "striking" improvement in accuracy with no more computation for some shock wave problems [Burstein and Mirin, 1970]. A comparison of methods for weather prediction by [Grammeltvedt, 1969] suggests to us that fourth order methods may be superior, but Roache states the opposite. None of these papers attempts a controlled comparison of methods and thus no definitive conclusions can be reached from them.

Eason [1976] has a bibliography of 241 items relevant to the least squares method for partial differential equations. He tabulates the references in various ways including Table III. Comparisons where least-squares methods are superior in accuracy, convenience or computing speed and Table IV. Comparisons where least-squares methods produce equivalent or comparable results. Eason is a strong advocate of the least squares method which may explain why a table where least squares does worse is not included. For example, Table III has 26 entries for collocation and 14 for Galerkin. We have examined most of these references and they are, in general, one of two types. First, someone attempts to solve a problem, say, with collocation using 12 polynomial terms and with least squares using 8 trigonometric polynomial terms. The problem has an unknown solution so the actual accuracy is unknown. The author reports his subjective evaluation of the quality of the results obtained. Usually there is insufficient data about the calculation to attempt to reproduce the results. Note that the differences observed are primarily due to using polynomials versus trigonometric polynomials rather than using collocation versus least squares. The second type of paper is more systematic, but involves trivial problems in one way or another (i.e. either the problem is trivial or the method used
is trivial). For example, one sees solutions of three fairly simple
problems by five methods which compute a quadratic polynomial approximation.
Then general conclusions are stated. We did not locate any systematic
and realistic evaluation of methods among these 40 references. Most
papers do not even give conclusive evidence in the particular context of
the problem they consider.

If there is any consistent pattern in the results, it would be that
authors find that the collocation of boundary conditions is delicate.
Many find that least squares approximations to the boundary conditions give
better results, primarily because they do not use good boundary collocation
points. This does suggest that collocation of the differential equation
combined with least squares for the boundary conditions would give a more
robust numerical method with little or no penalty in efficiency.

Leissa et al [1969] present a systematic study of the value of 9
methods for two plate bending problems: a simply supported elliptic plate
and a square plate supported at 4 "random" points. In both cases the
"exact" solution is a series expansion truncated at 48 terms, but the authors
do not view this as just another numerical method which might give worse
results than some of the other methods they apply. The nine methods are
compared on the basis of 11 criteria e.g. "suitability for programming",
"applicability to general regions", "ease in learning". Efficiency and
accuracy were not included directly as criteria and apparently were not
systematically measured. It is important to note that all of the 9 methods
considered were of limited flexibility and none could be applied to all
17 problems included in this study.
REFERENCES


APPENDIX ONE

GRAPHS OF THE COMPARISON DATA FOR 17 PROBLEMS

The data for the comparison of methods is plotted on log-log paper with accuracy achieved versus execution time. The accuracy is plotted as the actual error at the location of the maximum error. The execution time is in seconds on a CDC 6500. A consistent scheme of plotting is used for the four methods: solid for collocation, dots for finite differences, dashes for Galerkin and dot-dash for Least Squares. Occasionally, some extra curves are plotted which are identified by a special label.

One may crudely estimate the "time order" \( a \) of these methods by measuring the slopes of the curves of error vs. time when plotted on log-log paper. The order \( a \) estimated is for the relationship

\[
\text{Error} = O(\text{Time }^{-a})
\]

If one assumes that most of the computer time is spent in solving the linear systems, then one would have

\[
\text{Error} = O(N^{-4a})
\]

This assumption is clearly not satisfied here. In Table A1 we present our estimates of \( a \) and \( 4a \). We see that there is some correlation with the simple model which gives \( 4a = 2 \) for finite differences and \( 4a = 4 \) for the Hermite cubic method. There are also some very wide deviations from this.

Table A1. Measured slopes \( a \) to estimate the order of the methods from their actual performance.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Finite Diff.</th>
<th>Collocation</th>
<th>Galerkin</th>
<th>Finite Diff.</th>
<th>Collocation</th>
<th>Galerkin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a )</td>
<td>( 4a )</td>
<td>( a )</td>
<td>( 4a )</td>
<td>( a )</td>
<td>( 4a )</td>
</tr>
<tr>
<td>1</td>
<td>0.65</td>
<td>2.6</td>
<td>1.44</td>
<td>5.8</td>
<td>1.9</td>
<td>7.6</td>
</tr>
<tr>
<td>2</td>
<td>1.13</td>
<td>4.5</td>
<td>2.4</td>
<td>9.6</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>0.94</td>
<td>3.8</td>
<td>1.7</td>
<td>6.8</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>0.59</td>
<td>2.4</td>
<td>1.37</td>
<td>5.5</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>0.47</td>
<td>1.9</td>
<td>4.0</td>
<td>16.0</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>0.55</td>
<td>2.2</td>
<td>1.46</td>
<td>5.8</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>0.61</td>
<td>2.4</td>
<td>1.39</td>
<td>5.6</td>
<td>2.0</td>
<td>6.2</td>
</tr>
<tr>
<td>8</td>
<td>0.58</td>
<td>2.3</td>
<td>0.67</td>
<td>2.7</td>
<td>1.5</td>
<td>6.1</td>
</tr>
</tbody>
</table>

?
Figure A1. The data for Problems 1 to 4. Galerkin and Least Squares data is given for Problem 1. For Problem 4 we also plot the maximum error over the whole region to compare with that at the nodes.

Figure A2. The data for Problems 5 to 8. The solution to Problem 8 has a mild singularity, which seems to affect the collocation solution more than Galerkin or Least Squares.

Figure A3. The data for Problems 9 to 12. Galerkin and Least Squares show erratic behavior for Problem 10. The "boundary layer" of Problem 12 adversely affects both methods of solution.

Figure A4. The data for Problems 13 to 15. The effect of collocation with a non-uniform mesh for the wave front on a right angle (Problem 13) and for an isolated sharp peak (Problem 15) is seen. The erratic behavior of collocation with a uniform mesh for Problem 13 seems to be due to the chance relationship between the mesh and the wave front.

Figure A5. The data for Problems 16 and 17 with the complicated geometry of Figure 1(c). The complex geometry does not adversely affect Problem 16 where surprising accuracy is obtained. The singularities and complex geometry also do not seem to adversely affect Problem 17 (recall that the true solution is of size 100.) where the geometry forced non-uniform meshes for both collocation and finite differences.
PROBLEM 1
SOLUTION SIZE = 1.3

ERROR

10^{-1}

10^{-2}

10^{-3}

10^{-4}

10^{-5}

0.1  1.0  10.  100.

TIME

FINITE DIFFERENCE
LEAST SQUARES
GALERKIN
COLLOCATION
PROBLEM 2
SOLUTION SIZE = 0.87

```
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
```

```
TIME
```

```
10.  100.
```

```
ERROR
```

```
FINITE DIFFERENCE

COLLOCATION
```

24
PROBLEM 3
SOLUTION SIZE = 0.8

ERROR

\[ 10^{-1} \]

\[ 10^{-2} \]

\[ 10^{-3} \]

\[ 10^{-4} \]

TIME

0.1 1.0 10. 100.

- FINITE DIFFERENCE
- COLLOCATION

25
PROBLEM 4
SOLUTION SIZE = 7.6

- FINITE DIFFERENCE, GENERAL POINT
- COLLOCATION, GENERAL POINT
- FINITE DIFFERENCE, NODAL POINTS
- COLLOCATION, NODAL POINTS
PROBLEM 5
SOLUTION SIZE = 2.6
PROBLEM 6
SOLUTION SIZE = 7.4

FINITE DIFFERENCE
PROBLEM 7
SOLUTION SIZE = 0.58

FINITE DIFFERENCE
LEAST SQUARES
GALERKIN
COLLOCATION

ERROR

TIME

10^{-6}

10^{-5}

10^{-4}

10^{-3}

10^{-2}
PROBLEM 8
SOLUTION SIZE = 0.1
PROBLEM 9
SOLUTION SIZE = 2.0

ERROR

- - - - - COLLOCATION
- - - - - FINITE DIFFERENCE
- - - - - GALERKIN
- - - - - LEAST SQUARES

TIME

10^{-5}  10^{-4}  10^{-3}  10^{-2}  10^{-1}
PROBLEM 10
SOLUTION SIZE = 3.2

ERROR

10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}

TIME

0.1
1.0
10.0
100.0

FINITE DIFFERENCE
LEAST SQUARES
COLLOCATION
GALEKIN
PROBLEM 11
SOLUTION SIZE = 2.0

ERROR

10^1

10^-1

10^-2

10^-3

10^-4

TIME

0.1 1.0 10. 100.

FINITE DIFFERENCE

COLLOCATION
PROBLEM 12
SOLUTION SIZE = 2.0

ERROR

FINITE DIFFERENCE

COLLOCATION

TIME

34
PROBLEM 13
SOLUTION SIZE = 1.0
COLLOCATION, UNIFORM MESH

FINITE DIFFERENCE

COLLOCATION
NONUNIFORM MESH

ERROR

10^{-4}

10^{-3}

10^{-2}

10^{-1}

10^{0}

TIME

0.1

1.0

10.

100.

35
PROBLEM 14
SOLUTION SIZE = 2.0

![Graph showing error vs. time for Problem 14 with solution size 2.0, comparing finite difference and collocation methods with error on a logarithmic scale.]
PROBLEM 15
SOLUTION SIZE = 0.6
PROBLEM 16
SOLUTION SIZE = 4.9

FINITE DIFFERENCE

COLLOCATION

ERROR

TIME

10^{-3}

10^{-4}

10^{-5}

10^{-6}

10^{-7}

0.1

1.0

10.

100.
PROBLEM 17
SOLUTION SIZE = 100.0

ERROR

10^{-2}

10^{-1}

10^{0}

10^{1}

10^{2}

TIME

0.1

1.0

10.

100.
APPENDIX TWO
SYNOPSIS OF THE NUMERICAL METHODS

1. Standard Finite Differences. This method has the following components.
   (a) Grid: A rectangular grid is placed over the domain and all points in the domain or on its boundary are used. The grid is uniformly spaced except for Problems 16, 17 where the geometry made that undesirable.
   (b) Approximation to the operator: The derivatives in differential equation are replaced by simple central, 3-point finite difference approximations involving the grid points.
   (c) Approximation to the boundary conditions: Derivatives in Neumann or mixed boundary conditions are approximated as indicated by the diagram
      \[\text{x-derivative at } P \text{ estimated from value at } P \text{ and the 2 } x\text{-points.}\]
      \[\text{y-derivative at } P \text{ estimated from value at } P \text{ and the 2 } y\text{-points.}\]
      The values at the y-points are found by linear interpolation from the 0-points.
   (d) Equation Solution: The linear system is solved by Gauss-elimination taking into account the zeros in the system (profile or frontal method).

2. Collocation. This method has the following components.
   (a) Elements: A rectangular grid is placed over the domain. Rectangular elements whose center is not inside the domain are discarded. The grid is uniform unless noted except for Problems 16, 17.
   (b) Approximation space: the Hermite bicubics defined at the end of this appendix.
   (c) Approximation to the operator: The approximate solution satisfies the differential equation exactly at the four Gauss point of a rectangular element.
   For non-rectangular elements near the boundary the four Gauss points are
projected inside the element as indicated by the diagram.

\[ x = \text{differential equation collocation points} \]

(d) Approximation to the boundary conditions: The boundary conditions are interpolated at a selected set of boundary points for either Dirichlet, Neumann or Mixed boundary conditions. If the domain is a rectangle and the problem has Dirichlet conditions = 0 (Problems 1, 7, 8, 9, 10 and 15) then the Hermite bicubics are selected so as to automatically satisfy the boundary conditions and no boundary approximation equations are used. This is the same procedure as for the Galerkin and Least Squares methods. See Appendix 3 for details on how the boundary collocation points are selected.

(e) Equation Solution: Same as for standard finite differences.

3. Ritz-Galerkin and Least Squares. The components of these methods are:

(a) Elements: same as for collocation.

(b) Approximation space: same as for collocation.

(c) Approximation to the operator: In each element \( E \) of the partition we have the Galerkin equations

\[
\sum_{i=1}^{16} a_i \iint_{E} \left( p \frac{\partial B_i}{\partial x} \frac{\partial B_j}{\partial x} + q \frac{\partial B_i}{\partial y} \frac{\partial B_j}{\partial y} + r B_i B_j \right) \, dx \, dy = \iint_{E} f B_j \, dx \, dy
\]

where the operator \( L \) and the true solution \( U^* \) are defined by

\[
LU^* = (p U^*)_x + (q U^*)_y + r U^* = f
\]

and

41
D_x, D_y = differentiation operators

B_i(x,y), B_j(x,y) = the i and j elements of the Hermite bicubic basis

\( a_i \) = coefficient of \( B_i \) in the approximate solution (the index \( i \) refers to one element only)

The Least Squares equation in each element is

\[
\sum_{i=1}^{16} a_i \int_E L(B_i) \cdot L(B_j) \, dx \, dy = \int_E f(x,y) L(B_i) \, dx \, dy
\]

The integrals in these equations are approximated by the 9-point Gauss quadrature rule for rectangles (only rectangular domains were used with these methods).

(d) Approximation to the boundary conditions: the boundary condition were exactly satisfied by the Hermite cubic basis for all problems (1, 7, 8, 9, 10 and 15) attempted with these methods.

(e) Equation solution: The local equations are assembled (by the direct stiffness method) to form the global matrix. This equation is solved by Gauss elimination for positive definite matrices.

4. The Rectangular Bicubic Hermite Element. The situation is shown in the diagram

\( s = x/a \) and \( 0 < s < 1 \)
\( t = y/b \) and \( 0 < t < 1 \)

The numerical labels on the corners are used later to index the points.
We use 8 one-dimensional functions to construct the 16 basis functions for the rectangle:

\[ B_{x1} = 1 - 3s^2 + 2s^3 \]
\[ B_{x2} = s^2(3 - 2s) \]
\[ B_{x3} = 2s(s - 1)^2 \]
\[ B_{x4} = 2s^2(s - 1) \]
\[ B_{y1} = 1 - 3t^2 + 2t^3 \]
\[ B_{y2} = t^2(3 - 2t) \]
\[ B_{y3} = bt(t - 1)^2 \]
\[ B_{y4} = bt^2(t - 1) \]

Then \( u^*(x, y) \) is approximated in each rectangle by

\[
\begin{align*}
  u(x, y) &= B_{x1} B_{y1} u_1 + B_{x2} B_{y1} u_2 + B_{x2} B_{y2} u_3 + B_{x1} B_{y2} u_4 \\
  &+ B_{x3} B_{y1} \sigma_{x1} + B_{x4} B_{y1} \sigma_{x2} + B_{x4} B_{y2} \sigma_{x3} + B_{x3} B_{y2} \sigma_{x4} \\
  &+ B_{x1} B_{y3} \tau_{xy1} + B_{x2} B_{y3} \tau_{xy2} + B_{x2} B_{y4} \tau_{xy3} + B_{x1} B_{y4} \tau_{xy4} \\
  &+ B_{x3} B_{y3} \tau_{xy1} + B_{x4} B_{y3} \tau_{xy2} + B_{x4} B_{y4} \tau_{xy3} + B_{x3} B_{y4} \tau_{xy4}
\end{align*}
\]

where \( u_i \) = value at the point \( i \)
\( \sigma_{x1}, \sigma_{y1} = x \) and \( y \) derivatives at the point \( i \)
\( \tau_{xy1} = xy \) (cross) derivative at the point \( i \).

The 16 functions in the above equation are the ones denoted by \( B_i(x, y) \) earlier in the Galerkin and Least Squares equations, e.g. \( B_1(x, y) = B_{x1} B_{y1} \).
APPENDIX THREE

THE INTERPOLATION OF BOUNDARY CONDITIONS FOR COLLOCATION

The most sensitive aspect of collocation is the placement of the boundary collocation points for non-rectangular domains. First, one must take care that these points are reasonably separated from the points in the interior where one collocates with the differential operator. This is not difficult to do even in an automatic way, but the penalty for overlooking this point is an ill-conditioned computation with large errors.

One first overlays the region with a rectangular grid and discards the elements which intersect the domain slightly or not at all. Let $S_b$ be the number of boundary sides of the resulting rectangular partition. Then the number of boundary collocation points required is $2S_b + 4$. We use two basic schemes for distributing the boundary collocation points as illustrated by the diagrams below for a simple rectangle:

![Diagram](image)

**Figure A6.** Two schemes for distributing boundary collocation points. The x's are the systematic collocation points and the O's are the four extra ones.
A theoretical analysis shows that the 2-point scheme is superior for rectangular regions provided the two points are taken to be the Gauss points for each boundary segment. We compared using the Gauss points with equally spaced points and found the equally spaced points give slightly better accuracy and they are slightly easier to use. We made numerous numerical experiments which confirmed that the 2-point scheme is superior for rectangular regions.

The extension of these two schemes to curved domains is illustrated in Figure A7.

![Figure A7](image)

Figure A7. The two schemes for a simple curved domain. The lines show how the collocation points are placed on the edge of the rectangular partition and then mapped onto the portions of the boundary intersecting each rectangular element.

The theoretical advantage of the 2-point scheme no longer holds for curved boundaries and our experiments confirm that it has no advantage over the midpoint scheme in this case. In fact it is, on the average, slightly less accurate. Furthermore, the midpoint scheme automatically gives collocation
of the boundary conditions at any extremities of the domain (for example, for a piecewise rectangular boundary such as in Problems 16 and 17, see Figure 1). It is often essential that collocation of the boundary conditions be made at all exterior corners of the domain.

Our procedure is to use the 2-point scheme for boundaries which are straight (or nearly so) and parallel to a coordinate axis and to use the midpoint scheme otherwise. The two schemes may be used together for a domain such as shown above and we do this as shown in Figure A8.

![Figure A8](image)

Figure A8. The combination of the two schemes for a partially rectangular region. The mapping from the point on the rectangular edges to the curved boundary is indicated.

There seems to be no particularly advantageous method to distribute the 4 extra collocation points beyond putting them in elements with exterior corners and spreading them somewhat evenly around the boundary. We always map the midpoint type collocation points to segments of the curved boundary which are interior to the rectangular partition. The points are placed uniformly on each such segment. At times this may leave rather large segments of a curved boundary "unused", but we have not found a reliable method to place collocation points on the intermediate segments. We do place collocation
outside the rectangular partition for the 2-point scheme. An example is shown in Figure A9 which illustrates these procedures.

Figure A9. Example which illustrates boundary collocation points for the 2-point scheme which are outside the rectangular partition and collocation for the midpoint scheme are inside. Collocation is not done on two large boundary segments.
APPENDIX FOUR

THE SOLUTION OF PROBLEM 17 AND FUNCTIONS INVOLVED IN THE OTHER PROBLEMS

We describe the exact solution \( u \) of Problem 17 for the reactor heat shields \( \nabla^2 u = f \).

We set

\[
u(x,y) = 100 \frac{g(x,y,0,0)}{g(x,y,a,b,c)}
\]

where, by construction, the numerator on the right is zero on the stair-step outer boundary of the domain (see Figure 1). The numerator is the product of \( (x-1) \), \( (y-1) \), and three factors of the form \( r^2/3 \sin(3(\theta + \pi/2)/2) \)

where \( r \) is the distance between \( (x,y) \) and the reentrant corner \( (x_i,y_i) \), \( i = 1,2,3 \). The denominator is a modification of the numerator which is positive in a region containing the boundary of the heat shield and which is equal to the numerator along the circular part of the boundary. Note that this function has the correct singularities at the reentrant corners.

Specifically:

\[
g(x,y,a,b,c) = [(x-1)(y-1) + a C(x,y)] \prod_{i=1}^{3} T(x,y,x_i,y_i,b,c)
\]
\[
C(x,y) = (x^2 + y^2 - .64)^2
\]
\[
T(x,y, x_i, y_i, b, c) = R(x,y, x_i, y_i, b) S(x,y, x_i, y_i, c)
\]
\[
R(x,y, x_i, y_i, b) = [(x-x_i)^2 + (y-y_i)^2 + b C(x,y)]^{1/3}
\]
\[
S(x,y, x_i, y_i, c) = \sin(2[\arctan((y-y_i)/(x-x_i)) + \pi/2]/3) + c C(x,y)
\]

with branch cut along \( y-y_i = x-x_i, x_i < x \)

After some experimentation, we found that \( a = -.5, b = .1, c = 7 \).

gives a solution \( u \) which is similar to that one expects for the temperature in the heat shield.

48
Remark about the evaluation of $u$ and $f = \nabla^2 u$:

In our first attempt at the construction of a suitable $u$, we used a somewhat simpler function [which later proved to be unsuitable because it had zeros in the interior of the region]. A Fortran program was written for the evaluation of $u$ and it was processed by a symbolic differentiator to obtain function subroutines to evaluate $u_{xx}$ and $u_{yy}$. The resulting programs for $u$, $u_{xx}$, $u_{yy}$ were more complicated and much longer than the one we eventually wrote for our more complicated function. We note that $u$, $u_{xx}$, $u_{yy}$ can each be evaluated by successive calls to a number of very simple subroutines. Each of these evaluates $V$, $V_{xx}$, $V_{yy}$ where $V$ is a product $V = WZ$. Schematically the program is:

\[
\begin{align*}
W &= \ldots \\
WX &= \ldots \\
WXX &= \ldots \\
Z &= \ldots \\
ZX &= \ldots \\
ZXX &= \ldots \\
V &= W*Z \\
VX &= WX*Z + W*ZX \\
VXX &= WXX*Z + 2.*WX*ZX + W*ZXX
\end{align*}
\]

and similarly for the $y$-derivatives.

The values of $V$, $VX$, $VXX$, $VY$, $VYY$ are stored in a common block for use by subsequent routines. In most cases, statements like the first six above: $W = \ldots$, $ZXX = \ldots$, do not appear since the values are already computed by previously called subroutines. The program is quickly written and debugged.
**PROBLEM 1 DATA**

FUNCTION CDEF(X,Y,J)
  Z = EXP(X*Y)
  RZ = 1/Z
  GO TO (101,102,103,104,105)
 101 CDEF = Z
  RETURN
 102 CDEF = RZ
  RETURN
 103 CDEF = Y * Z
  RETURN
 104 CDEF = -X * RZ
  RETURN
 105 CDEF = -1/(1+X+Y)
  RETURN
END

FUNCTION F(X,Y,J)
  GO TO (101,102)
 101 PI = 3.14159265358979
  Z = EXP(X*Y)
  RZ = 1/Z
  PIX = PI*X
  PIY = PI*Y
  PI2 = PI*Z
  SINK = SINK(PIX)
  SIMY = SINK(PIY)
  TRUE = Z*SINK*SIMY
  TEMP = PI*PI*TRUE
  XTRUE = X*TRUE
  YTRUE = Y*TRUE
  FX = PI2*COS(PIX)*SINK
  FY = PI2*COS(PIY)*SINK
  DXTR = XTRUE + FX
  DYT = YTRUE + FY
  DXTR = Y*TRUE - TEMP + Z*Y Money
  DYT = X*TRUE - TEMP + Z*X Money
  F = 2*DXTR*X*DYTR + Y*Z*DXTR - X*RZ*DYTR - TRUE/(1+X+Y)
  RETURN
 102 F = 0.
  RETURN
END

FUNCTION TRUE(X,Y)
  GO TO (101,102)
 101 PI = 3.14159265358979
  TRUE = EXP(X*Y)*SINK(PIX)*SINK(PIY)
  RETURN
END

FUNCTION BCDEF(X,Y,J)
  GO TO (101,102,103)
 101 BCDEF = 1.
  RETURN
 102 BCDEF = 0.
  RETURN
 103 BCDEF = 0.
  RETURN
END
FUNCTION F(X,Y,J)
101 F = 0.
RETURN
102 IF(Y.EQ.0..OR.X.EQ.1.) GO TO 1
IF(Y.EQ.0..OR.Y.EQ.5.) GO TO 1
F = 1.
RETURN
IF(X.EQ.0..OR.X.EQ.1.) GO TO 101
F = 0.
RETURN
END
FUNCTION COEF(X,Y,J)
101 COEF = G(X,Y)
RETURN
102 COEF = 0.
RETURN
103 COEF = 0.
RETURN
END
FUNCTION G(X,Y)
E = .00001
X1 = .5-E
X2 = .5 + E
DX = X2 - X1
IF (X < X1) GO TO 1
IF (X > X2) GO TO 2
POL = 3. - 6. *(X-X1)**2 + 4. *(X-X1)**3/(3*X**3)
G = 1./POL
RETURN
G = 1./3.
RETURN
END
FUNCTION BCDEF(X,Y,J)
101 IF(X > 0. AND X < 0.25) GO TO 1
IF(X > 0.75 AND X < 1.) GO TO 1
BCDEF = 0.
RETURN
1 IF(Y = 0.) GO TO 2
BCDEF = 0.
RETURN
2 BCDEF = 0.
RETURN
102 BCDEF = 0.
RETURN
103 IF(X > 0. AND X < .25) GO TO 11
IF(X > .75 AND X < 1.) GO TO 11
BCDEF = 0.
RETURN
11 IF(Y = 0.) GO TO 22
BCDEF = 0.
RETURN
22 BCDEF = 1.
RETURN
END

51
FUNCTION F(X, Y)  
GO TO (101, 102, J)
101 F = -20.
RETURN
102 F = 0.
RETURN
END
FUNCTION CDEF(X, Y, J)
GO TO (110, 110Q, 1103, J)
1101 CDEF = G(X, Y)
RETURN
1102 CDEF = G(X, Y)
RETURN
1103 CDEF = 0.
RETURN
1104 CDEF = 0.
RETURN
1105 CDEF = 0.
RETURN
END
FUNCTION G(X, Y)
E = .00001
XI = .5-E
X£ = .5 + E
DX = X£ - XI
IF X .LE. XI GO TO 1
IF X .GE. X£ GO TO 2
FDL = 3 - 6*(X - XI)**2/DX**2 + 4*(X - XI)**3/DX**3
G = 1/FL
RETURN
1 G = 1./3
RETURN
2 G = 1.
RETURN
END
FUNCTION BDEF(X, Y, J)
GO TO (101, 102, 103, J)
101 IF X .LT. 0 AND X .LT. .25 GO TO 1
IF X .LT. .75 AND X .LT. 1. GO TO 1
BDEF = 1.
RETURN
1 IF Y .EQ. 0 GO TO 2
BDEF = 1.
RETURN
2 BDEF = 0.
RETURN
102 BDEF = 0.
RETURN
103 IF X .LT. 0 AND X .LT. 2 GO TO 11
IF X .LT. .75 AND X .LT. 1. GO TO 11
BDEF = 0.
RETURN
11 IF Y .EQ. 0 GO TO 22
BDEF = 0.
RETURN
22 BDEF = 1.
RETURN
END
FUNCTION TRUE(X, Y)
 TRUE = (EXP(X) + EXP(Y))/(1 + X*Y)
 RETURN
END

FUNCTION DTRUE(X, Y)
 Z = 1/(1 + X*Y)
 DTRUE = EXP(X)*Z - TRUE(X, Y)*X*Z
 RETURN
END

FUNCTION DXYTR(X, Y)
 Z = 1/(1 + X*Y)
 DXYTR = - (EXP(X)*X + EXP(Y)*Y) / (Z*Z)
 RETURN
END

FUNCTION FC(X, Y, J)
 EX = EXP(X)
 EY = EXP(Y)
 Z = 1/(1 + X*Y)
 GOTO (1 + X*Y) + J
 1 F = (EX + EY - Z*Z) / (Z + (Y*EX + X*EY - Z*Z) * (Y*EY + X*EX)) / Z
 RETURN
2 F = (EX + EY) / Z
 RETURN
END

FUNCTION COEF(X, Y, J)
 GO TO (1 + 2 + J) + J
 1 COEF = 1.
 RETURN
2 COEF = 1.
 RETURN
3 COEF = 0.
 RETURN
4 COEF = 0.
 RETURN
5 COEF = 0.
 RETURN
END

FUNCTION BCOEF(X, Y, J)
 GO TO (1 + 2 + J) + J
 1 BCOEF = 1.
 RETURN
2 BCOEF = 0.
 RETURN
3 BCOEF = 0.
 RETURN
END
FUNCTION TRUE<Y>  
TRUE = ATAN<Y/X> + 1. 
RETURN  
END

FUNCTION F<Y>X<Y,J>J>
GO TO (101;102), J
101 F = 0.
RETURN
102 IF<X,EQ.,5,.AND., Y,EQ.,0,> GO TO 1
F = -(Y - X)/<X + Y - .25>
RETURN
1 F = -1.
RETURN
END

FUNCTION COEF<Y,X,J,J>
GO TO (101;102;103;104;105), J
101 COEF = 1.
RETURN
102 COEF = 1.
RETURN
103 COEF = 0.
RETURN
104 COEF = 0.
RETURN
105 COEF = 0.
RETURN
END

FUNCTION BCDEF<Y,X,J,J>
GO TO (11;22;33), J
11 BCDEF = 0.
RETURN
11 BCDEF = -1.
RETURN
22 IF<X,EQ.,5,.AND., Y,EQ.,0,> GO TO 22
BCDEF = Y/.5 - 1.
RETURN
33 BCDEF = 0.
RETURN
33 BCDEF = 0.
RETURN
END
**PROBLEM 6 DATA**

FUNCTION COEF(X,Y,J)

GO TO (101,102,103,104,105),J

101 COEF = 1.
RETURN
102 COEF = 1.+Y*Y
RETURN
103 COEF = -1.
RETURN
104 COEF = -(1.+Y*Y)
RETURN
105 COEF = 0.
RETURN
END

FUNCTION F(X,Y,J)

GO TO (101,102),J

101 F = (-4.*X*X*X+18.*X*X-14.*X+2.)*AL0G(1.+Y*Y)-2.*(X*X-X)**2*(Y*Y+Y**3+Y-1.)/(1.+Y*Y)
RETURN
102 IF(X.EQ.0. OR Y.EQ.0.) GO TO 1
F = (AL0G(2.)-1.)*(X*X-X)**2
RETURN
1 F = 2.*EXP(X-Y)
RETURN
END

FUNCTION TRUE(X,Y)
TRUE = EXP(X+Y)+(X*X-X)**2*AL0G(1.+Y*Y)
RETURN
END

FUNCTION BCDEF(X,Y,J)

GO TO (101,102,103),J

101 BCDEF = 1.
RETURN
102 IF(X.EQ.0.) GO TO 1
IF(X.EQ.1.) GO TO 2
BCDEF = 0.
RETURN
1 BCDEF = 1.
RETURN
2 BCDEF = -1.
RETURN
103 IF(Y.EQ.0.) GO TO 11
IF(Y.EQ.1.) GO TO 12
BCDEF = 0.
RETURN
11 BCDEF = 1.
RETURN
12 BCDEF = -1.
RETURN
END
M**M PROBLEM 7 DATA M**M

FUNCTION COEF(X,Y,J)
  GO TO (101,102,103,104,105),J
101 COEF = 1.
  RETURN
102 COEF = 1.
  RETURN
103 COEF = 0.
  RETURN
104 COEF = 0.
  RETURN
105 COEF = 0.
  RETURN
END

FUNCTION F(X,Y,J)
  GO TO (101,102),J
101 F = 6.*X*Y*EXP(X)*EXP(Y)*X*Y+X+Y-3.
  RETURN
102 F = 0.
  RETURN
END

FUNCTION TRUE(X,Y)
 TRUE = 3.*EXP(X)*EXP(Y)*X+Y-1.
  RETURN
END

FUNCTION BCOEF(X,Y,J)
  GO TO (101,102,103),J
101 BCOEF = 1.
  RETURN
102 BCOEF = 0.
  RETURN
103 BCOEF = 0.
  RETURN
END
FUNCTION COEF(X,Y,J)
GO TO (101,102,103,104,105),J
101 COEF = 1.
RETURN
102 COEF = 1.
RETURN
103 COEF = 0.
RETURN
104 COEF = 0.
RETURN
105 COEF = 0.
RETURN
END

FUNCTION F(X,Y,J)
GO TO (101,102), J
101 XR = SQRT(X)
YR = SQRT(Y)
F = 3.75 * (XR * YR * (XR + YR) - XR * Y - X * YR)
RETURN
102 F = 0.
RETURN
END

FUNCTION TRUE(X,Y)
XR = SQRT(X)
YR = SQRT(Y)
TRUE = XR*XR*YR*YR - XR*YR*YR - XR*XR*Y + X*Y
RETURN
END

FUNCTION BCDEF(X,Y,J)
GO TO (101,102,103), J
101 BCDEF = 1.
RETURN
102 BCDEF = 0.
RETURN
103 BCDEF = 0.
RETURN
END
**PROBLEM 9 DATA**

FUNCTION TRUE(X,Y)
  PI = 3.14159265358979
  TRUE = 4.*(X*X-X)*(COS(2.*PI*Y)-1.)
  RETURN
END

FUNCTION F(X,Y,J)
  GO TO (101:102:J)
101 PI = 3.14159265358979
  F = (32. + (256.+16.*PI*PI)*(X-X*X))
  \[ \cos(2.*PI*Y)+256.*X*X-X) -32. \]
  RETURN
102 F = 0.
  RETURN
END

FUNCTION CDEF(X,Y,J)
  GO TO (101:102:103:104:105:J)
101 COEF = 4.
  RETURN
102 COEF = 1.
  RETURN
103 COEF = 0.
  RETURN
104 COEF = 0.
  RETURN
105 COEF = -64.
  RETURN
END

FUNCTION BCOEF(X,Y,J)
  GO TO (101:102:103:J)
101 BCOEF = 1.
  RETURN
102 BCOEF = 0.
  RETURN
103 BCOEF = 0.
  RETURN
END
FUNCTION TRUE(X,Y)
PI=3.141592653589793
FDURP=4.*PI
FPX=FDURP*X
FPY=FDURP*Y
CX=COS(FPX)
CY=SIN(FPY)
F1=-CX+5.4
F2=-CY+5.4
F3= (X-.5)*(X-.5)+(Y-.5)*(Y-.5)
F3£=16.*F3*F3
F33= F3*F3
F34=F3*F3
Z=1.\(1+F34\)
F4=Z-.5
SPX=SINCPI*X
GDFY=Y*Y-Y
TRUE=F1*SPX*GDFY*F2*F4
RETURN
END
FUNCTION FCX,Y)
GO TO (101,102)+J
101 PI=3.141592653589793
FDURP=4.*PI
FPX=FDURP*X
FPY=FDURP*Y
SXTPSQ=FDURP*FDURP
SX=SIN(FPX)
SY=SIN(FPY)
CX=COS(FPX)
CY=COS(FPY)
F1=-CX+5.4
DAX=FDURP*X
DAY=FDURP*Y
DDXF1=FDURP*X
DDYF2=FDURP*Y
F3= (X-.5)*(X-.5)+(Y-.5)*(Y-.5)
F3£=16.*F3*F3
F33= F3*F3
F34=F3*F3
Z=1.\(1+F34\)
F4=Z-.5
DXF3= 5.*<X-.5>
DYF3= 5.*<Y-.5>
DDF3=2.<X-.5>
ZZ=ZZ
UXX=DDXF1*SPX*GDFY*F2*F4
DXF1*PICPX*GDFY*F2*F4
DXF1*SPX*GDFY*DXF1*F2*F4
F1*PI*PI*SPX*GOFY*F2*F4
F1*PICPX*GDFY*F2*F4
DXF1*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
#PI*PCX*GDFY*F2*F4
FUNCTION CDEF(X,Y,J)

CDEF = 100. + CDS<2.*PI*X> + SIN<3.*PI*Y>
RETURN

END

Function BCDEF(X,Y,J)

CDEF = 100. + CDS<2.*PI*X> + SIN<3.*PI*Y>
RETURN

END
**Problem 11: Data**

FUNCTION COEF(X,Y,J)
    GO TO (101,102,103,104,105), J
101 COEF = 1.
    RETURN
102 COEF = 1.
    RETURN
103 COEF = 0.
    RETURN
104 COEF = 0.
    RETURN
105 COEF = -100.
    RETURN
END

FUNCTION F(X,Y,J)
    GO TO (101,102), J
101 F = 0.
    RETURN
102 F = TRUE(X,Y)
    RETURN
END

FUNCTION TRUE(X,Y)
    TRUE = (COSH(10.*X) + COSH(10.*Y)) / COSH(10.)
    RETURN
END

FUNCTION COSH(X)
    COSH = (EXP(X) + EXP(-X)) / 2.
    RETURN
END

FUNCTION BCDEF(X,Y,J)
    GO TO (101,102,103), J
101 BCDEF = 1.
    RETURN
102 BCDEF = 0.
    RETURN
103 BCDEF = 0.
    RETURN
END
FUNCTION COEF(X; Y, J)
  GO TO (101; 102; 103; 104; 105; J)
101  COEF = 1.
    RETURN
102  COEF = 1.
    RETURN
103  COEF = 0.
    RETURN
104  COEF = 0.
    RETURN
105  COEF = 100.
    RETURN
END

FUNCTION F(X; Y; J)
  GO TO (101; 102; 103; 104; 105; J)
101  F = 3.09 * COSH(20. * Y) / COSH(20.)
    RETURN
102  F = TRUE(X; Y)
    RETURN
END

FUNCTION TRUE(X; Y)
  TRUE = COSH(10. * X) * COSH(10.) + COSH(20. * Y) / COSH(20.)
  RETURN
END

FUNCTION COSH(X)
  COSH = (EXP(X) + EXP(-X)) / 2.
  RETURN
END

FUNCTION BCDEF(X; Y; J)
  GO TO (101; 102; 103; 104; 105; J)
101  BCDEF = 1.
    RETURN
102  BCDEF = 0.
    RETURN
103  BCDEF = 0.
    RETURN
END
FUNCTION COEF(X, Y, J)
GO TO (101, 102, 103, 104, 105) J
101 COEF = 1.
RETURN
102 COEF = 1.
RETURN
103 COEF = 0.
RETURN
104 COEF = 0.
RETURN
105 COEF = 0.
RETURN
END

FUNCTION F(X, Y, J)
GO TO (101, 102, 103, 104, 105) J
101 F = DBP(X) * P(Y) + P(X) * DBP(Y)
RETURN
102 F = TRUE(X, Y)
RETURN
END

FUNCTION TRUE(X, Y)
TRUE = P(X) * P(Y)
RETURN
END

FUNCTION BCOEF(X, Y, J)
GO TO (101, 102, 103) J
101 BCOEF = 1.
RETURN
102 BCOEF = 0.
RETURN
103 BCOEF = 0.
RETURN
END

FUNCTION P(X)
A = 1.
B = 0.
E = 0.15
X1 = 0.5 - E
X2 = 0.5 + E
IF(X < X1) GO TO 1
IF(X > X2) GO TO 2
DPHI = B - A
DX = X2 - X1
P = A + DPHI * (X - X1) / DX
RETURN
1 P = A
RETURN
2 P = B
RETURN
END

FUNCTION DBP(X)
A = 1.
B = 0.
E = 0.15
X1 = 0.5 - E
X2 = 0.5 + E
IF(X < X1) GO TO 1
IF(X > X2) GO TO 2
SPHI = B - A
DX = X2 - X1
C3 = DPHI / DX
C4 = -3. * DPHI / DX
C5 = 6. * DPHI / DX
S = 2. * C5 * (X - X1) * (X - X2)
RETURN
1 DBP = 0.
RETURN

END
FUNCTION TRUE(T,S)
  E = 0.0625
  X = 4.0
  Y = 4.0
  T1 = 7.*X*(X-2.)*X+Y*(Y-1.)
  T2 = EXP(-4.*X*(Y-2.)*X)*EXP(-4.*X*(Y-1.))
  T3 = (X-2.)*X+Y*(Y-1.)*Y
  TRUE = TIME(T,T)
RETURN
END

FUNCTION F(T,S,J)
  E = 0.0625
  X = 4.0
  Y = 4.0
  T1 = 7.*X*(X-2.)*X+Y*(Y-1.)
  T2 = EXP(-E*X*(Y-2.)*X)*EXP(-E*X*(Y-1.))
  T3 = (X-2.)*X+Y*(Y-1.)*Y
  TRUE = TIME(T,T)
RETURN
END

FUNCTION CDEF(X,P,Y,J)
  1 CDEF = 1.
RETURN
  2 CDEF = 1.
RETURN
  3 CDEF = 0.
RETURN
  4 CDEF = 0.
RETURN
  5 CDEF = 0.
RETURN

FUNCTION CDEF(X,P,Y,J)
  1 CDEF = 1.
RETURN
  2 CDEF = 1.
RETURN
  3 CDEF = 0.
RETURN
  4 CDEF = 0.
RETURN
  5 CDEF = 0.
FUNCTION BCDEF(K+J,J)
GO TO (1,2,3),J
1 BCDEF=1,
RETURN
2 BCDEF=0,
RETURN
3 BCDEF=0,
RETURN
END
FUNCTION COEF(X,Y,J)
    GO TO (101,102,103,104,105,J)
101 COEF = 1.
    RETURN
102 COEF = 1.
    RETURN
103 COEF = 0.
    RETURN
104 COEF = 0.
    RETURN
105 COEF = 0.
    RETURN
END

FUNCTION F(X,Y,J)
    P = .1
    GO TO (101,102,J)
101 TEMP = -((X-.5)**2+(Y-.5)**2)/P**2
    F1 = EXP(TEMP)
    D1UX = -2.*TRUE(X)Y/P**2 + S F1*(X-.5)*(Y-.5)*Y/P**2
    D1UY = -2.*TRUE(X)*Y/P**2 + S F1*(X-.5)*(Y-.5)*X/P**2
    D2UX = -2.*TRUE(X)*Y/P**2 + S F1*(X-.5)*(Y-.5)*X/P**2
    D2UY = -2.*TRUE(X)*Y/P**2 + S F1*(X-.5)*(Y-.5)*X/P**2
    F = (D2UX+D2UY)
    RETURN
102 F = 0.
    RETURN
END

FUNCTION TRUE(X,Y)
    P = .1
    TEMP = -((X-.5)**2+(Y-.5)**2)/P**2
    TRUE = EXP(TEMP)*(X-1.)*X*(Y-1.)*Y/P
    RETURN
END

FUNCTION BCEF(X,Y,J)
    GO TO (101,102,103,J)
101 BCEF = 1.
    RETURN
102 BCEF = 0.
    RETURN
103 BCEF = 0.
    RETURN
END
FUNCTION COEF(X, Y, J)
  GO TO (101, 102, 103, 104, 105), J
  101 COEF = 1.
  RETURN
  102 COEF = 1.
  RETURN
  103 COEF = 0.
  RETURN
  104 COEF = 0.
  RETURN
  105 COEF = 0.
  RETURN
END

FUNCTION TC(X, Y, J)
  GO TO (101, 102), J
  101 F = 2. * TRUE(X, Y)
  RETURN
  102 F = TRUE(X, Y)
  RETURN
END

FUNCTION TRUE(X, Y)
  TRUE = EXP(X + Y)
  RETURN
END

FUNCTION BCDEF(X, Y, J)
  GO TO (101, 102, 103), J
  101 BCDEF = 1.
  RETURN
  102 BCDEF = 0.
  RETURN
  103 BCDEF = 0.
  RETURN
END
FUNCTION F(X,Y,J)
F COMPUTES EITHER THE TRUE SOLUTION OF PROBLEM 17 OR
THE LAPLACIAN UXX + UYY.
LOGICAL NDERV
IF J .EQ. 1 THEN EVALUATE THE LAPLACIAN
ELSE EVALUATE THE TRUE SOLUTION
NDERV = .TRUE.
IF ( J .EQ. 1 ) NDERV = .FALSE.
CALL EVLATE(X,Y,NDERV,
A Q,GQX,QGY,GQXX,GQYY,IFLAG)
F = 17770C-00000000000000000
IFC NDERV  F = 0
IFC .NOT. NDERV .AND. IFLAG .EQ. 0 F = GXX + GYY
RETURN
END
SUBROUTINE EVLATE(X,Y,NDERV,
A Q,GQX,QGY,GQXX,GQYY,IFLAG)
INPUT XX,YY,NDERV,DINJ,DINLY,BUGALL,BUGGAL,
LOCAL VARIABLES X,Y,NDERV FOR XX,YY,NDERV
OUTPUT Q,GQX,QGY,GQXX,GQYY,IFLAG,IFLAG
IFLAG SET TO 0 IF SUCCESSFUL, SET TO 1 IF NOT
WHEN UNSUCCESSFUL, QV SET TO ZERO AND GQX,GQY,GQXX,GQYY
ARE SET TO INDEFINITE, THIS OCCURS AT REENTRANT BOUNDARY
Corners
EVALUATES QV, GQX, GQY, GQXX, GQYY
QV = SIZE*QVAL
SIZE IS CONSTANT SET IN DATA
QVAL = GVAL(6) / SIZE
A,B,C,D ARE CONSTANT SET IN DATA
GVAL = EVAL*FVAL(3)*FVAL(5)*FVAL(7)
FVAL = (X - 1)*(Y - 1) + A*CIR2
CIR = X**2 + Y**2 - RHDSQR
RHDSQR IS CONSTANT SET IN DATA
FVAL(1) = RVAL(1) + FVAL(1)
RVAL(1) = (X - XPT(1)**2 + (Y - YPT(1))**2
+ DINC2)**2 - 1)
XPT(1) IS 1-TH REENTRANT CORNER POINT
THVAL = SNVAL + CCINC
SNVAL(1) = SIN(2*WVAL(1)/S)
WVAL(1) = ARCTAN((Y-YPT(1))*(X-XPT(1)) - PI/2

Y-AXIS

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>---------------</td>
</tr>
</tbody>
</table>

X-AXIS

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>---------------</td>
</tr>
</tbody>
</table>
COMMON /SUBCOM/ A PI1, PI2, PI4, TAUPI1, TAUPI2, TAUPI4, PII1, PII2, PII4, 
PID1, PID2, PID4, TWDPI1, TWDPI2, TWDPI4, THTH1, THTH2, THTH4, 
TUTr, TUTS, NETHR, FRTH, NDDERV, 
X, Y, XDSQR, YDSQR, ANGLE, 
D SIZE = A, B, C, 
E ANVAL = ANVALX, ANVALY, ANVALXX, ANVALYY, 
G BVAL(<2) = BVALX(2), BVALY(2), BVALXX(2), BVALYY(2), 
F CIR = CIRX, CIRY, CIRCXX, CIRCYY, 
J CT1(2) = CT1X(2), CT1Y(2), CT1XX, CT1YY, 
K EVK(2,3) = EVKX(2,3), EVKY(2,3), EVKXX(2,3), EVKYY(2,3), 
L OPQ(2) = OPQX(2), OPQY(2), OPQXX(2), OPQYY(2), 
N SVAL = SVALX, SVALY, SVALXX, SVALYY, 
R THVAL(2) = THVALX(2), THVALY(2), THVALXX(2), THVALYY(2), 
DIMEOTION CTALL(2,5), CREALL(5), EQUIVALENCE (CTALL(1,1)) (CREALL(1)), CIR2) 
DATA CTALL / 1000.0 / 
LOGICAL NDERV, NDER, NDERH, BUGALL, BUGVAL 
REAL INDEX, INDEX 
DATA INDEX / 1777.00,00000000,0000,000B / 
DATA PI1, PI2, PI4, TAUPI1, TAUPI2, TAUPI4, PII1, PII2, PII4, 
A 3.14159265358979, 1.57079632679490, .78539816339745, 
B 5.23598775593975, 4.71238888888888, .66666666666667, 
C .44444444444444, .33333333333333, 1.33333333333333, 
DATA SIZE = A, B, C, RHD, RHOSQR / 100.0, .5, 1.0, 7.0, 8.0, 64.0 / 
DATA XPT / .65, .65, .95 / 
DATA YPT / .7, .9, .9 / 
MAKE W, X, Y = NDER, LOCAL IN /SUBCOM/ 
X = XX, Y = YY 
NDER = NDERH 
XSQR = XW, YSQR = YW 
CALL CIRCLE 
CALL BVALS 
D2 20 I = 1, 3 
XD = X - XPT(I) 
YP = Y - YPT(I) 
LBSOR = VBVMD 
VBVS = VDVBV 
KBVS = KSORS + YSORS 
IFE KBSOR GT. L.E.0 ) THEN TOO CLOSE TO I-TH BOUNDARY CORNER 
JINL = 1 
OV = 0. 
OVX = INDEF 
OVY = INDEF 
OVX = INDEF 
OVY = INDEF 
GO TO 10 
EXIT 
GO TO 30 
CONTINUE 
ELSE CAN EVALUATE 
CALL ANVALS 
CALL SVALS 
CALL BVALS 
GO TO 10 

69
CALL THVALS
J = 1
CALL FVVALS(FVVAL(1,J),FVVALX(1,J),FVVALY(1,J),FVVAL(1,J),FVVALX(1,J),FVVALY(1,J));
IF( IFLAG > 0 ) CALL DEBUGS1>
CONTINUE
CALL GVALS
CALL GVALS GV, GVX, G Y, GVXX, GVYY
GV = SIZE GV
IFLAG = 0
CONTINUE
RETURN
END
SUBROUTINE CIRCLE
FORM CIR = X**2 + Y**2 - RHO*SQ
CIRX = CIR**1
AND DERIVATIVES
COMMON /SUBCD/ *** REPEAT VARIABLES HERE ***
DIMENSION CTALL(5,J), CRALL(5>
EQUIVALENCE (CTALL(1), CRALL(1), CIR, CIRX)
LOGICAL NODERV
CIR = X*SQ + Y*SQ - RHO*SQ
CIRX = 2.*CIR*X
CYRY = 2.*CIR*Y
CIRXX = 4.*CIR*X*SQ + Y*SQ - RHO*SQ
CYRYY = X*SQ + 4.*CIR*Y*SQ - RHO*SQ
CONTINUE
RETURN
END
SUBROUTINE BVALS
FORM BVAL = CX-I + A*CIRX*(Y-1 + ACIR)
AND DERIVATIVES
COMMON /SUBCD/ *** REPEAT VARIABLES HERE ***
DIMENSION CTALL(5,J), CRALL(5>
EQUIVALENCE (CTALL(1), CRALL(1), CIR, CIRX)
LOGICAL NODERV
WM = XM - 1.
YN = YM - 1.
ACIR = A*CIR
NFACT = XM + ACIR
YFACT = YM + ACIR
BVAL(1) = WM*YN
BVAL(2) = NFACT*YFACT
CONTINUE
RETURN
END
SUBROUTINE ANVALS

FORM ANVAL = ARCTAN(YD/XD) - Pi/2 AND DERIVATIVES

X:CRCTAN(YD/XD)/DUD = Y*(YD*YD + XD*XD) 

ANGLE MEASURED COUNTER-CLOCKWISE FROM XD-AXIS
ANVAL MEASURED COUNTER-CLOCKWISE FROM YD-AXIS
BRANCH POINT AT XD = YD = 0.
BRANCH CUT ALONG ANGLE = Pi/4. ANVAL = -Pi/4

1. Y-Axis
   ANVAL = 0
   YD-Axis
   ANGLE = Pi/2

2. X-Axis
   ANGLE = 0
   XD-Axis
   ANVAL = Pi/2

3. (X,Y)
   (XD,YD)

COMMON /SUBCOM/ *** REPEAT VARIABLES HERE ***
DIMENSION CALL(2,5), CR2ALL(5)
EQUIVALENCE (CALL(1,1)), (CR2ALL(1), CIR2)

LOGICAL NDERV
IF( NDERV ) GO TO 10

COMPUTE DERIVATIVES
ANVALX = -YD*ROSSR
ANVALY = XD*ROSSR
ANAVLX = -2.*YD*ANVALX/ROSSR
ANAVLY = -2.*YD*ANVALY/ROSSR
10 CONTINUE

IF( ABS YD > GT. ABS XD ) GO TO 20
THEN ANGLE BETWEEN U AND Pi/4 OR 3Pi/4 AND 5Pi/4 OR Pi/4 AND 2Pi
ANGLE = ATAN(YD/XD)
IF( XD .LT. 0. ) ANGLE = Pi + ANGLE

71
IF (ANGLE .LT. 0.) ANGLE = TWOPI + ANGLE

20 CONTINUE
ELSE ANGLE BETWEEN PI/4 AND 3PI/4 OR SPI/4 AND 7PI/4
ANGLE = PI - ATAN (XD/YD)
END

30 CONTINUE

SUBTRACT PI/2 TO MAKE ANVAL BETWEEN 0 AND 3PI/2
ANVAL = ANGLE - PI/2
ADJUST FOR BRANCH CUT

IF (ANVAL .LT. -PI/2) ANVAL = TWOPI + ANVAL

RETURN

END

SUBROUTINE SNVALS

FORM SNVAL = SIN (CNANVAL); DSNVAL, DDSNVAL
COMMON /SUBCOM/ *** REPEAT VARIABLES HERE ***
DIMENSION CTALL(2,5), CR£ALL(5)
EQUIVALENCE (CTALL<1>1%), CT(1), (CR£ALL<1)F, CIR2)
LOGICAL NDERV

ARG = TUTH*ANVAL
SNVAL = SIN (ARG)

IF (NDERV) GO TO 10

SUBROUTINE THVALS

FORM THVAL = SNVAL + C*CIR£ AND DERIVATIVES
C = 0., FOR NUMERATOR
COMMON /SUBCOM/ *** REPEAT VARIABLES HERE ***
DIMENSION CTALL(2,5), CR£ALL(5)
EQUIVALENCE (CTALL<1>1%), CT(1)), (CR£ALL<1)> CIR2)
LOGICAL NDERV

THVAL(1) = SNVAL
THVAL(2) = THVAL(1) + C*CIR£
IF (NDERV) GO TO 10

THVALX (1) = DSNVAL*ANVAL
THVALY (1) = DSNVAL*ANVAL

THVALX(1) = DSNVAL*ANVALX
THVALY(1) = DSNVAL*ANVALY

THVALX(2) = THVALX (1) + C*CIR£X
THVALY(2) = THVALY (1) + C*CIR£Y
THVALX(2) = THVALX(1) + C*CIR£X
THVALY(2) = THVALY(1) + C*CIR£Y
10 CONTINUE

SUBROUTINE RDVALS

FORM RDVAL = (COSR + BCEIR)**(1/3) AND DERIVATIVES
E = D, FOR NUMERATOR
COMMON /SUBCOM/ *** REPEAT VARIABLES HERE ***
DIMENSION CTALL(2,5), CR£ALL(5)
EQUIVALENCE (CTALL(1,1), CT(1)), (CR£ALL(1), CIR2)

LOGICAL NDDERV

SET CIRCLE-TEMP FOR DENOMINATOR (NUMERATOR SET TO ZERO IN DATA)

DO 10 IDERV = 1, 5
   CTALL(2,IDERV) = B*CR£ALL(IDERV)
10   CONTINUE

EVALUATE FOR NUMERATOR AND DENOMINATOR (NUM = 1, DEN = 2)

DO 30 NMDN = 1
   RVALUE = RDSRR + CT(NMDN)
   RDVAL(NMDN) = RVALUE**ONETHR
   IF (NDDERV) GO TO 20

   COMPUTE DERIVATIVES
   TV = -3TH*RDVAL(NMDN)/RVALUE
   TVR = -3TH*RDVAL(NMDN)/RVALUE
   TV = -TH*RDVAL(NMDN)/RVALUE
   TVR = -TH*RDVAL(NMDN)/RVALUE
   TV = -TH*RDVAL(NMDN)/RVALUE
   TVR = -TH*RDVAL(NMDN)/RVALUE
   TV = -TH*RDVAL(NMDN)/RVALUE
   TVR = -TH*RDVAL(NMDN)/RVALUE
   TV = -TH*RDVAL(NMDN)/RVALUE
   TVR = -TH*RDVAL(NMDN)/RVALUE
   CONTINUE
30   CONTINUE

RETURN

END

SUBROUTINE FVALUES(FV, FVX, FVY, FVXX, FVYY)
FORMS: FV = ( RSQ**(-1/2) )*SIN( 2*ANVAL/3 )

COMMON /SUBDIV/ *** REPEAT VARIABLES HERE ***

DIMENSION CTALL(2,5), CR£ALL(5)
EQUIVALENCE (CTALL(1,1), CT(1)), (CR£ALL(1), CIR2)

LOGICAL NDDERV

DIMENSION FV(2), FVX(2), FVY(2), FVXX(2), FVYY(2)

FOR NUMERATOR (1) AND DENOMINATOR (2)

DO 20 N = 1, 2
   FV(N) = RDVAL(N)*THVAL(N)
   GO TO 10

   COMPUTE DERIVATIVES
   FVX(N) = RDVAL(N)*THVAL(N) + RDVAL(N)*THVAL(N)
   FVY(N) = RDVAL(N)*THVAL(N) + RDVAL(N)*THVAL(N)
   FVXX(N) = RDVAL(N)*THVAL(N) + 2.*RDVAL(N)*THVAL(N)
   FVYY(N) = RDVAL(N)*THVAL(N) + 2.*RDVAL(N)*THVAL(N)
   CONTINUE
20   CONTINUE

RETURN

END

SUBROUTINE GVALUES
FORM: GVAL = EVAL*F3*M5*M7 AND DERIVATIVES

COMMON /SUBDIV/ *** REPEAT VARIABLES HERE ***

DIMENSION CTALL(2,5), CR£ALL(5)
EQUIVALENCE (CTALL(1,1), CT(1)), (CR£ALL(1), CIR2)

LOGICAL NDDERV

DIMENSION F357(2), F35K(2), F35KK(2), F35KK(2), F35KK(2), F35KK(2)

RETURN

END
COMPUTE NUMERATORS (1) AND DENOMINATORS (2)

DO 20 M = 1, 8
   FS(N) = FS(N)*FS(N)
   F5(N) = FS(N)*FS(N)
   GVAL(N) = GVAL(N)*GVAL(N)
   IF (NDERV),
   COMPUTE DERIVATIVES
   F57X(N) = F5X(N)*F7(N)
   F57Y(N) = F5Y(N)*F7(N)
   GVALX(N) = GVALX(N)*F57(N)
   GVALY(N) = GVALY(N)*F57(N)
   F57XX(N) = F5X(N)*F7(N) + 2.*F5X(N)*F7(N)
   F57YY(N) = F5Y(N)*F7(N) + 2.*F5Y(N)*F7(N)
   GVALXX(N) = GVALXX(N)*F57(N) + 2.*GVALX(N)*F57(N)
   GVALYY(N) = GVALYY(N)*F57(N) + 2.*GVALY(N)*F57(N)
   10 CONTINUE
20 CONTINUE
RETURN
END

SUBROUTINE GVALS(GV, GVX, GVY, GVXX, GVYY)

FORM GV = |V(GV) = GVAL(1)/GVAL(2)

AND DERIVATIVES

COMMON *** REPEAT VARIABLES HERE ***

DIMENSION CTALL(5), CR2ALL(5)
EQUIVALENCE (CTALL(1), CT(1)), (CR2ALL(1), CIR2)

LOGICAL NDDERV

GV = GVAL(1)/GVAL(2)
IF (NDDERV),
   COMPUTE DERIVATIVES
   FACT = 1./GVAL(2)
   FACTSQ = FACT*FACT
   FACTX = -GVALX(2)*FACTSQ
   FACTY = -GVALY(2)*FACTSQ
   FACTXX = (2.*GVALXX(2)*FACT - GVALXX(2)*FACTSQ)
   FACTYY = (2.*GVALYY(2)*FACT - GVALYY(2)*FACTSQ)
   GVX = GVALX(1)*FACT + GVALX(1)*FACTX
   GVY = GVALY(1)*FACT + GVALY(1)*FACTY
   10 CONTINUE
RETURN
END