Solution of Non-Linear Partial Differential Equations

with the Chebyshev Spectral Method

by

Lloyd B. Eldred

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Aerospace and Ocean Engineering

APPROVED:

Rakesh K. Kapania

Robert W. Walters

Eric R. Johnson

October, 1989

Blacksburg, Virginia

Solution of Non-Linear Partial Differential Equations

with the Chebyshev Spectral Method

by

Lloyd B. Eldred

Rakesh K. Kapania

Aerospace and Ocean Engineering

(ABSTRACT)

The Spectral method is a powerful numerical technique for solving engineering differential equations. The method is a specialization of the method of weighted residuals. Trial functions that are easily and exactly differentiable are used. Often the functions used also satisfy an orthogonality equation, which can improve the efficiency of the approximation. Generally, the entire domain is modeled, but multiple sub-domains may be used.

A Chebyshev-Collocation Spectral method is used to solve a variety of ordinary and partial differential equations. The Chebyshev Polynomial series follows a well established recursion relation for caluculation of the polynomials and their derivatives. Two different schemes are studied for formulation of the problems, a Fast Fourier Transform approach, and a matrix multiplication approach. First, the one-dimensional ordinary differential equation representing the deflection of a tapered bar under its own weight is studied. Next, the two dimensional Poisson's equation is examined. Lastly, a two-dimensional, highly non-linear, two parameter Bratu's equation is solved.

Each problem's results are compared to results from other methods or published data. Accuracy is very good, with a significant improvement in computer time.

Acknowledgements

This thesis is dedicated to my family and friends for their support throughout the years of work leading to this document. I would also like to acknowledge the support, advice, and assistance of Dr. Rakesh Kapania without whom this would not been possible. Further, I would like to acknowledge the financial support of the Federal Highway Administration, and the aid of during a portion of this research. Thanks are also due to Drs. Robert Walters and Eric Johnson for serving on my committee.

Table of Contents

List of Illustrations

1.0 Introduction

With the advent of high speed computers, increasing attention has been given to solving engineering problems on computers. The differential equations that describe such problems are often extremely complex and difficult to solve. Current numerical techniques can require many hours of expensive computational time to solve. Consequently, improvements in computational efficiency are extremely desirable.

1.1 Spectral Methods

The family of techniques known as the method of weighted residuals have been used extensively to perform approximate solutions of a wide variety of problems. The so called spectral method is a specialization of this set of general techniques. By proper selection of trial functions to match the physics of a particular problem, spectacular spatial accuracies are possible. The term "infinite order accuracy" is often used to describe this

phenomena. Perhaps a better term would be "exponential" convergence, as the error of the solution decreases faster than any negative power of n.

Gottlieb and Orszag[1] and Peyret[2] provide tutorials on the basics of the various spectral methods. Both documents start from the method of weighted residuals, and continue through to showing results for several simple, example differential equations.

1.2 Previous Work

The spectral method is a very powerful technique for solving differential equations of interest to the engineering community. The technique can lead to substantially quicker convergence and fewer required degrees of freedom for comparable accuracy in comparison to finite element or finite difference techniques.

For instance, Haidvogel and Zang[3] examined a Chebyshev-Tau solution of the Poisson equation. They used both an alternating direction and a matrix diagonalization approach to solving the resulting Chebyshev system of algebraic equations . They reported results and performance equivalent to finite difference approaches to the solution, with slightly better accuracy in the spectral approach.

A spectral technique for solving the 2-D and 3-D Helmholtz equations was put forward by Haldenwang, et. al.[4] They examined several techniques for solving the spectral system of equations. Higher accuracy and reduced or similar computational costs were reported, when compared to a canned finite difference solver. A significant portion of their nu-

merical costs were the calculation of eigenvectors for the spectral matrix, an expense avoided in the current work.

Street and Hussaini[5] modeled the Taylor-Couette flow problem: flow between two coaxial circular cylinders, with the inner cylinder rotating and the outer one stationary. They used a Chebyshev collocation scheme to model the conservation form of the time dependent, incompressible Navier-Stokes equations. They were able to examine a variety of laminar-turbulent transition phenomina.

Ku, et. al.[6] solved a three dimensional incompessible, time dependent driven cavity problem using a Chebyshev-collocation scheme. They used calculation of eigenvectors and eigenvalues to reduce the storage requirements of the solution, at the expense of solution speed.

Haldenwang and Labrosse[7] examined a Chebyshev-Tau solution to both a two dimensional and three dimensional thermally driven cavity problem. Their iterative time dependent approach compared well with other iterative schemes, requiring generally fewer degrees of freedom for the same level of precision.

However, boundary condition treatment with spectral methods are often extremely difficult to manage. Since the spectral method is global in nature, singularities, or sharp changes in boundary conditions may result in poisoning the solution in a region well away from the singularity. Macaraeg and Streett[8,9] examined this problem for a step change in an edge boundary condition for a Poisson problem. Figure 1 shows the poisoning of the solution in the area around the boundary condition singularity.

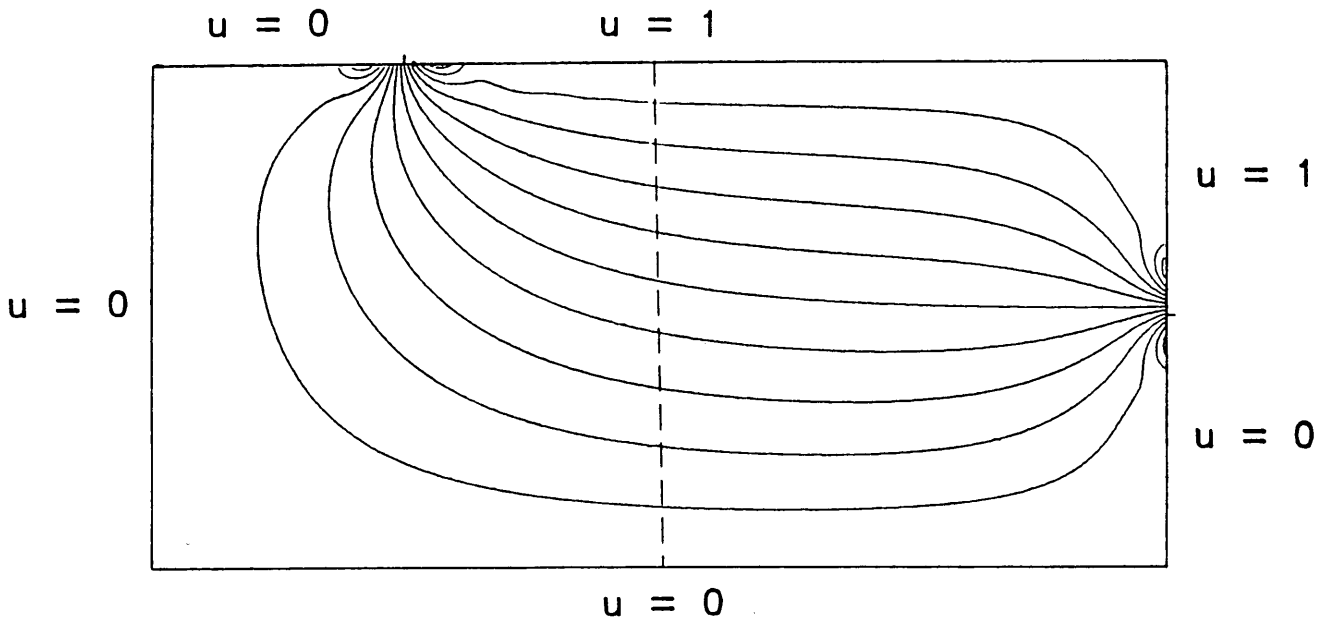

Figure 1. Poisoning of solution near a boundary condition singularity. Source: Macaraeg, Streett[9]

1.3 This Work

The work described in this thesis covers the initial process of learning a method through to some new discoveries. Work started on a reasonably simple one dimensional differential equation and proceeded to a highly non-linear two dimensional second order partial differential equation.

A one dimensional, second order differential equation was used to get familiar with the spectral methods. The equation used described the deflection of a tapered bar hanging under the effect of its own weight. Two spectral approaches were compaired to a generic finite element solution. The spectral approaches showed significantly faster convergence rates for somewhat higher programmer's effort.

The next problem examined was Poisson's equation. The extension from one dimension to two dimensions was non-trivial. Once accomplished, the performance of the solution was very good. Only a very few terms were needed to converge to the exact solution of the problem.

Bratu's equation is a two dimensional, highly non-linear partial differential equation. In addition to being a nonlinear partial differential equation that represents a wide variety of phenomena in both science and engineering (e.g. heat transfer with heat generation), it is considered to be an excellent test case for evaluating the performance of a solution method. Boyd[10] studied the solution of a one parameter version of the equation using a Chebyshev pseudospectral approach. Kapania[11] solved a two parameter version of the problem using a polynomial-collocation scheme. Both authors assumed, but did not prove, the solution to be symmetric. The current work did not make this assumption,

resulting in slightly more effort in it's solution. It was also discovered that the solution starts out as symmetric, but has unsymmetric regions.

2.0 Spectral Methods

The spectral method is a special case of the method of weighted residuals. Global, easily differentiable, approximation functions are used to approximate the independent variables. Special attention to boundary condition treatments are important when using this method.

2.1 Function Choice

A wide variety of functions such as Fourier series, sines, cosines, Chebyshev series, Bessel functions, or Legendre functions may be used with the spectral method. The domain shape and differentiation requirements of the problem at hand dictate which functions to choose.

A Fourier series is a natural choice for problems that are periodic in nature, such as shell buckling problems. The Fourier series is infinitely differentiable and these derivatives can be calculated exactly in a computer program.

A Chebyshev series is a good general choice for non-periodic problems. The Chebyshev polynomial of degree n, $T_n(x)$ is defined as:

$$T_n(x) = \cos(n \cos^{-1}(x)) \qquad [2.1]$$

Thus,

$$T_0(x) = 1, \qquad [2.2]$$

$$T_1(x) = x, \qquad [2.3]$$

$$T_2(x) = 2x^2 - 1, \qquad [2.4]$$

and in general

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \qquad [2.5]$$

Further, each term in the series is orthogonal to all the other terms in the series, making them ideal for global solution approximations. The polynomials satisfy the orthogonality relation:

$$\int_{-1}^{1} T_n(x)T_m(x)(1-x^2)^{-1/2}dx = \frac{\pi}{2}\, c_n \delta_{nm} \qquad [2.6]$$

where

$$c_0 = 2 \quad \text{and} \quad c_n = 1 \text{ for n } > 0. \qquad [2.7]$$

The series can be differentiated any number of times, exactly within a program. For example, if we have a series

$$f(x) = \sum_{n=0}^{N} a_n T_n(x) \qquad [2.8]$$

Then, any order derivative desired may be represented as a series of the same $T_n(x)$ with different expansion coefficients:

$$f^{(q)}(x) = \sum_{n=0}^{N} a_n^{(q)} T_n(x) \qquad [2.9]$$

The expansion coefficients, $a_n^{(q)}$, are easily related to the original expansion coefficients a_n (or $a_n^{(0)}$). For example:

$$a_n^{(1)} = \frac{2}{c_n} \sum_{\substack{p=n+1 \\ p+n=odd}}^{N} p a_p \qquad [2.10]$$

and

$$a_n^{(2)} = \frac{1}{c_n} \sum_{\substack{p=n+2 \\ p+n=even}}^{N} p(p^2 - n^2) a_p \qquad [2.11]$$

Alternatively, we can generate these derivative expansion coefficients from the recursion relation:

$$c_{n-1} a_{n-1}^{(q)} - a_{n+1}^{(q)} = 2n a_n^{(q-1)} \qquad [2.12]$$

A variety of other functions have been used for particular applications. Bessel and Legendre functions, for instance, have been studied.

2.2 Approximation Technique

Once the approximation functions are chosen, there are a variety of methods that can be used to determine the expansion coefficients needed to model the problem solution. These are the same basic techniques used with any variation of the method of weighted residuals, so will be mentioned only briefly here.

In the Galerkin method each chosen expansion function must individually satisfy the problem boundary conditions. Then the approximation error, or residue, is enforced to be orthogonal to each term in the series approximation.

The Tau method requires only that the approximate solution as a whole satisfies the boundary conditions. Again, the residue is set orthogonal to each term of the approximation series. Thus, a wider choice of trial functions is available, but the potential complexity of performing a orthogonality integral remains.

The collocation method (also called the pseudo-spectral approximation) requires the approximate solution to satisfy the governing differential equations at N points in the domain. The functions need not satisfy the boundary conditions individually. If the fucntions do not satisfy the boundary conditions, the boundary conditions may be enforced by adding the appropriate number of equations to the solution system.

2.3 Derivative Calculations

Two different techniques can be used to calculate derivatives of the problem variables. The first option is to calculate the expansion coefficients, derive the derivative expansion coefficients from them, and calculate the sum to find the values of the derivatives. This can be computationally wasteful, unless the values of the derivatives are a part of the desired output (e.g. in strains and stresses for problems in structural mechanics) of the problem. The second option is to use the relationships between the different levels of coefficients to assemble a set of equations representing the governing differential equation, but never actually calculating the coefficients explicitly.

A Fast Fourier Transform (FFT) can be used to transform the state variables into Fourier space. Then, the relationships between the levels of derivatives may be exploited to assemble the differential equation in Fourier space, and solve for the coefficients of the solution. Once obtained, the inverse FFT may be used to return to the state space solution. The addition of the FFT to the explicit derivative calculation significantly improves the speed of these calculations. With an additional modification, the FFT can be used with the Chebyshev polynomials as well. An example of the application of this approach will be discussed in the section detailing the one dimensional test problem.

Another differentiation technique that is used frequently is called the matrix multiplication technique. The differential operators can be written as matrix operations on an unknown state variable vector. Traditional matrix solution techniques may then be used. This technique is particularly powerful for linear problems, where the differential

operators are constant. Thus, the matrices can be factored once for a particular problem, and the factored form can be used repeatedly.

For instance, we can represent the derivative of u as:

$$\{u'\} = [D^1]\{u\} \qquad [2.13]$$

where $[D^1]$ is the first derivative operator. Then the second derivative may be calculated by applying this operator twice in succession. Since the order of multiplication doesn't matter, we may instead call this product of two D^1 matrices a D^2 matrix, our second derivative operator. Similarly, a great variety of other operators could be constructed. Once we have a stable of differential operators, a differential equation may be represented in terms of these operators. For instance, the equation

$$u'' + 3u' + 12u = f \qquad [2.14]$$

can be represented as

$$[D^2]\{u\} + 3[D^1]\{u\} + 12\{u\} = \{f\}. \qquad [2.15]$$

Factoring the common {u} out to the right gives us

$$\big[[D^2] + 3[D^1] + 12[I]\big]\{u\} = \{f\}. \qquad [2.16]$$

If we call the combined operator A, the differential equation can be represented as

$$[A]\{u\} = \{f\}, \qquad [2.17]$$

which, with appropriate attention to boundary conditions, may be solved using normal matrix techniques. In this work, the ESSL routines DGEF and DGES were used to factor and solve the matrices.

Equations for the terms in the D^1 and D^2 matrices may be found in Ref. 2. The formulas for D^1 are also reprinted in Appendix A. These formulas assume the use of a Gauss-Lobatto grid system, which is a natural choice for Chebyshev modeling. The Gauss-Lobatto grid nodes are defined as

$$x_i = \cos\left(\frac{\pi i}{N}\right) \qquad [2.18]$$

Applying this to a Chebyshev expansion gives the following form for the functional values at the nodes:

$$f(x_i) = \sum_{n=0}^{N} a_n \cos\left(\frac{n\pi i}{N}\right) \qquad [2.19]$$

On a single solution basis, the explicit (FFT) technique is slightly more efficient than the matrix multiplication method. This is also the case in problems where the differential

operators change from step to step, such as non-linear differential equations. However, when the differential operators are constant, a matrix multiplication technique is the desired method. The matrix representing the differential equation may be assembled and factored once, and quickly solved each time it is needed. A significant problem with the current application of the matrix multiplication technique is the production of very large matrices for problems in more than one dimension. This problem will be discussed in detail in the section discussion the two dimensional test problems.

Another consideration in the selection of a method to use is to compare the two methods in programming effort. The FFT scheme requires a significant amount of algebraic preparation before programming. This effort is to assemble the Fourier or Chebyshev space representation of the differential equation. On the other hand, the matrix multiplication technique is much easier to program. Once a few basic subroutines are programmed, an arbitrary differential equation may be assembled using repeated calls to this library of functions.

2.4 Technique Studied

For this work a Chebyshev collocation technique was used. This combination is the most general combination for non-periodic problems. Both FFT and matrix multiplying derivative calculations are used.

2.5 Boundary Conditions

Boundary condition treatments can be a significant problem when using spectral methods. First, adding equations into the equations to be solved can be difficult to accomplish, depending on the solution scheme used. Second, discontinuities in boundary conditions can introduce errors into a solution, resulting in instabilities in the numerical solution.

In collocation schemes a natural, commonly used, approach to enforcing a numerical value at a particular point in the domain is to remove the equation enforcing the differential equation at that point and replace it with an equation enforcing the value. This is not always straight forward. Take for instance the matrix multiplication version of the Poisson equation (discussed in detail in Chapter 4):

$$\left[[D_x^2][u]^T\right]^T + [D_y^2][u] = [F]. \qquad [2.20]$$

Direct replacement of a collocation point's equation is not immediately possible. Expansion of the matrix representation, rearrangement, and reassembly gives a more useful form:

$$[A]\{u\} = \{f\}. \qquad [2.21]$$

This form is now suitable for the equation replacement. However, it uses a much larger amount of computer storage and computational time.

Discontinuities in boundary conditions can also cause problems. Macaraeg and Streett[8,9] examined a Poisson problem with a discontinuity in the edge boundary conditions. Attempting to solve the problem in the standard way was found to be impossible. Figure 1 shows the poisoning of the solution near the discontinuity immediately before the solution diverged. They then performed the solution using two domains, with the interface through the discontinuity. Because of the use of the Gauss-Lobatto gridding system, a large number of nodes were concentrated in the vicinity of the discontinuity. This increased resolution isolated the effects of one singularity, and made solution possible, as shown in figure 2.

Thus, one needs to be aware of the problems that the lack of smooth boundary conditions may cause. In such instances, multiple domains can be used to overcome boundary condition related problems, at the expense of a significant addition in complexity.

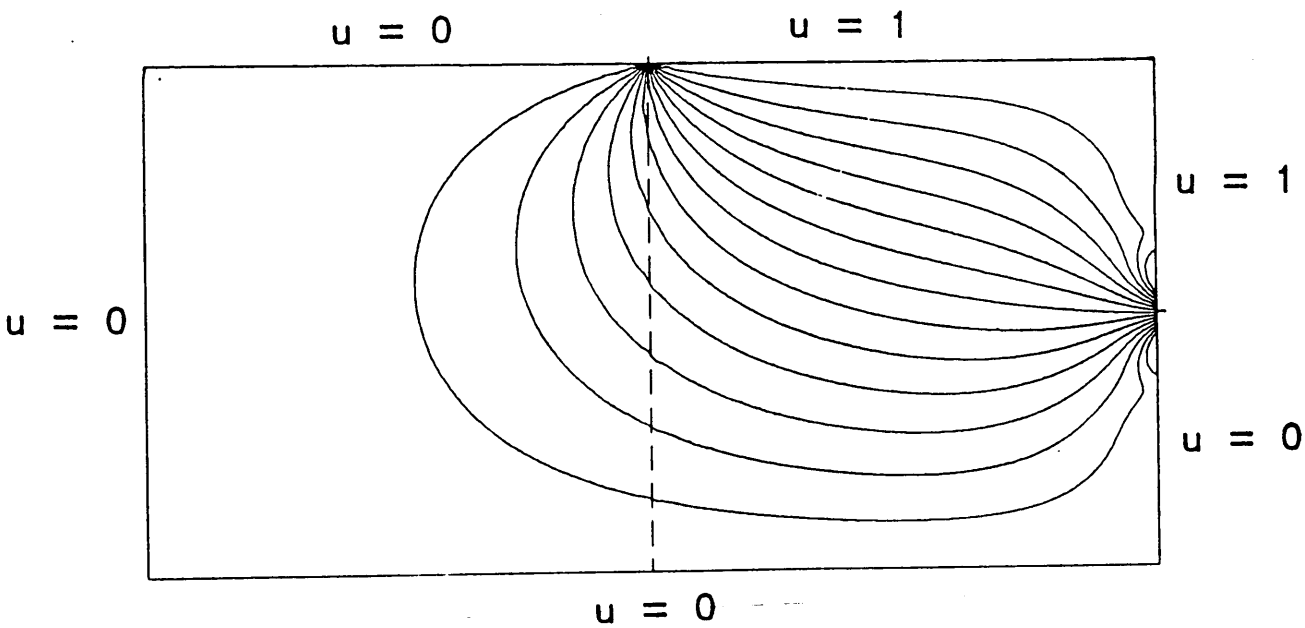

Figure 2. Two Domain Laplace Solution Source:Macaraeg,Streett[9]

3.0 One Dimensional Model Problem

A one dimensional model was used to gain familiarity with the spectral technique. The problem of a tapered bar hanging under its own weight was modeled with both a spectral technique and a finite element approach. The comparison was rather striking.

The governing differential equation for the problem is:

$$u''A + u'A' + cA = 0 \qquad [3.1]$$

where

$u =$ nondimensional deflection

$A =$ nondimensional cross-sectional area

$c =$ forcing constant, a function of the acceleration of

gravity, bar modulus, length, and density, $\frac{\rho g \ell}{4E}$.

For purposes of this study, the bar area was chosen to be circular, with a linear variation in radius from root to tip. A parameter called "Taper Ratio", the ratio of the tip radius to the root radius, was varied from 0.1 to 4.0 to test the techniques.

3.1 Explicit Approach

The first approach used was the explicit (FFT) approach. Each derivative in the non-dimensional differential equation above was replaced by the summation representing it's expansion in Chebyshev space:

$$A \sum_{k=0}^{N} a_k^{(2)} T_k(x) + A' \sum_{k=0}^{N} a_k^{(1)} T_k(x) = -cA \qquad [3.2]$$

Then, the $a^{(2)}$ and $a^{(1)}$ terms are replaced by the summation that gives them in terms of $a^{(0)}$

$$\sum_{k=0}^{n} \frac{1}{c_k} \sum_{\substack{p=k+2 \\ p+k=even}}^{N} p(p^2 - k^2) a_p T_k A + \sum_{k=0}^{N} \frac{2}{c_k} \sum_{\substack{p=k+1 \\ p+k=odd}}^{N} p a_p T_k A' = -cA \qquad [3.3]$$

After expanding and rearrangement, this can be written as

$$\sum_{k=0}^{N} \beta_k(x) a_k = -c \qquad [3.4]$$

where

$$\beta_n = n[2A' \sum_{\substack{k=n-1 \\ \text{step -2}}}^{0} \frac{T_k}{c_k} + A \sum_{\substack{k=n-2 \\ \text{step -2}}}^{0} (n^2 - k^2) \frac{T_k}{c_k}] \qquad [3.5]$$

and

$$\begin{aligned} c_0 &= 2 \\ c_n &= 1, \quad 0 < n < N \\ c_N &= 2 \end{aligned} \qquad [3.6]$$

Boundary conditions are enforced at both ends of the bar. The root condition is a zero deflection condition

$$u(-1) = 0 \qquad [3.7]$$

which after manipulation similar to that above becomes

$$\sum_{k=0}^{N} \gamma_k a_k = 0 \qquad [3.8]$$

with

$$\gamma_k = (-1)^{k.} \qquad [3.9]$$

The tip condition enforces that there is zero force applied at the tip. This is enforced by

$$u'A + A'u = 0 \text{ at } x = 1 \qquad [3.10]$$

which after manipulation becomes

$$\sum_{k=0}^{N} \alpha_k a_k = 0, \qquad [3.11]$$

with

$$\alpha_k = A' + 2nA \sum_{\substack{k=n-1 \\ \text{step } -2}}^{0} \frac{1}{c_{k.}} \qquad [3.12]$$

A matrix is assembled, enforcing the root condition at the root collocation point, the tip condition at the tip collocation point, and the differential equation at the remaining, N-1 points. This is N-1 because the collocation points are numbered from 0 to N. Thus, a value of N = 3 corresponds to 4 collocation points being used.

For the purpose of this study, N was started at 3 for each value of taper ratio and increased by one until the solution for the tip deflection stopped changing from that calculated with the previous value of N.

3.2 Matrix Multiplication Approach

A Chebyshev matrix multiply technique was also used. It started solving each problem with 4 terms (N = 3) and added terms until the solution converged. The main computational effort for each solution was the solution of one (N + 1) x (N + 1) system of equations. The differential equation was represented as:

$$[D]\{u\} = \{f\} \qquad [3.13]$$

where

$$[D] = [D^2] + \begin{bmatrix} \backslash & \\ & A'/A \\ & & \backslash \end{bmatrix} [D^1] \qquad [3.14]$$

and

$$\{f\} = c\left\{ \frac{A'}{A} \right\} \qquad [3.15]$$

After assembly of the differential operator matrix [D], the equations representing the differential equation at the tip and root of the bar were removed and replaced with

equations representing the boundary conditions. The root boundary condition enforced zero deflection at the root:

$$[1\,0\,0 \ldots 0]\{u\} = 0 \qquad [3.16]$$

This equation replaces the first row of the matrix equation 3.13.

The tip boundary condition enforces a zero force value at the tip. The equation

$$\frac{d}{dx}(EAu) = 0 \text{ at the free end} \qquad [3.17]$$

is rewritten for constant E as:

$$u' + u\frac{A'}{A} = 0 \text{ at the free end} \qquad [3.18]$$

which can be respresented in matrix form as:

$$\left[[D^1] + \frac{A'}{A}[I]\right]\{u\} = \{0\} \qquad [3.19]$$

The last row of this operator corresponds to the tip node. It is removed from the above equation and replaces the last row of the differential operator matrix, and zero is placed

in the last position in the right hand side force vector. Solution of the resulting system of equations yields the approximate solution for $\{u\}$. The number of terms, n used in the approximation is increased until the tip deflection converges.

3.3 Results

The two Chebyshev techniques produced identical results, as anticipated. Even in the worst case, only 15 terms were necessary for convergence.

The two Chebyshev solutions were compared with a finite element solution, to compare speed and accuracy. The finite element solution used the very general constant area bar element. Again, elements were added until the solution converged. Each additional element represented approximately the same computational effort as an additional Chebyshev term in the previous tests, as this solution also involves an NxN system. The worst case tested took 132 elements to converge, nearly 9 times number of terms, and 81 times the computational effort. Figure 3 shows the number of terms it took each approach to converge.

The purpose of this test was to examine the performance of the most general version of both approaches. A custom element can provide better performance for a particular area profile. But, if the area is an unknown "black box" no such customization is possible. For an arbitrary variation in cross sectional area, the uniform bar finite element may be the choice often used. As shown, the Chebyshev technique can perform much better.

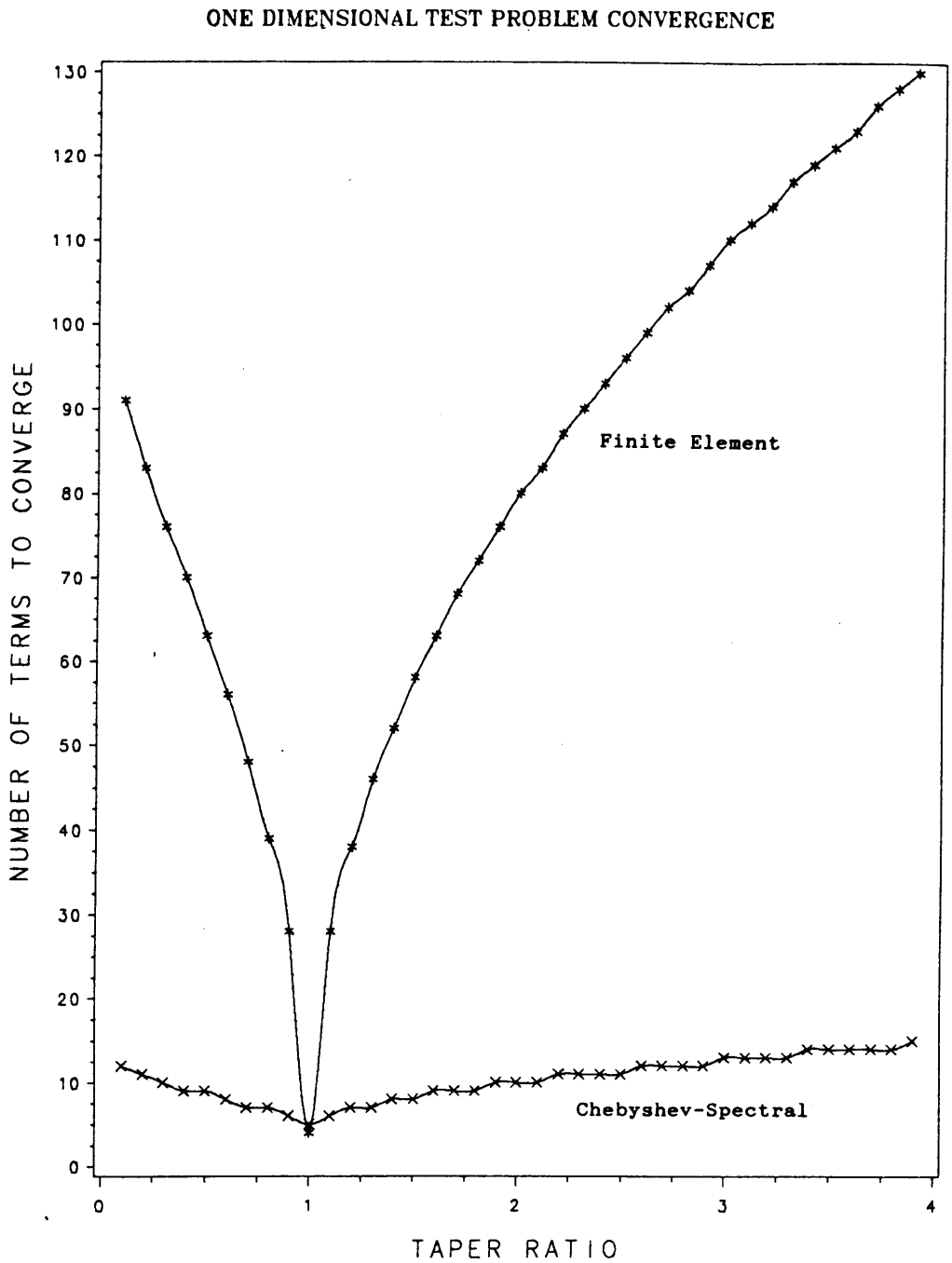

Figure 3. One Dimensional Problem Convergence

4.0 Two Dimensional Test Problem

For the two dimensional test problem, the two dimensional Poisson's Equation on a square domain was chosen. Solution of this equation can be a part of solving more complex two dimensional problems such as the full 2-D Navier Stokes equations. The Poisson's equation is:

$$\nabla^2 u = f \qquad [4.1]$$

4.1 Chebyshev Solution

The two dimensional version of the Chebyshev Poisson solver is not a simple extension from the one dimensional version. The matrix derivative operators were designed to operate on a vector. Thus, a derivative in one direction can be performed just be multi-

plying the operator matrix with the unknown matrix. The other direction requires two transpositions in order to have the operator operate on the proper values. Thus, the matrix multiplying technique gives a system of equations of the form:

$$\left[[D_x^2][u]^T\right]^T + [D_y^2][u] = [F] \qquad [4.2]$$

where

$[u]$ = solution matrix

$[D_x^2]$ = X direction second derivative operator

$[D_y^2]$ = Y direction second derivative operator

$[F]$ = Forcing matrix

The approach used to solve the system was to rearrange the equations into what looks like a one dimensional system:

$$[A]\{u\} = \{F\} \qquad [4.3]$$

This transformation was accomplished by just expanding the first equation and rearranging the equations. If we rearrange [u] to:

$$\{u\} = \{u_{00}\, u_{01}\, u_{02} \ldots u_{10}\, u_{11} \ldots u_{nm}\}^T \qquad [4.4]$$

then the x direction operation

$$\left[[D_x^n][u]^T\right]^T \qquad [4.5]$$

can be rewritten as

$$[C_B^n]\{u\} \qquad [4.6]$$

where $[C_B^n]$ simply contains M + 1 copies of $[D_x^n]$ along it's diagonal. The case when M = 2 is illustrated below:

$$[C_B^n] = \begin{bmatrix} [D_x^n] & [0] & [0] \\ [0] & [D_x^n] & [0] \\ [0] & [0] & [D_x^n] \end{bmatrix} \qquad [4.7]$$

The rearangement of the operation

$$[D_y^n][u] \qquad [4.8]$$

to

$$[C_A^n]\{u\} \qquad [4.9]$$

requires the dispersal of N + 1 copies of each entry in $[D_y^n]$ along diagonal lines in $[C_A^n]$. For illustration, when N = 1 and M = 2, $[D_y^n]$ is a 3 by 3 matrix. Two copies of each element are copied into $[C_A^n]$ to produce:

$$[C_A^n] = \begin{bmatrix} a_{00} & 0 & a_{01} & 0 & a_{02} & 0 \\ 0 & a_{00} & 0 & a_{01} & 0 & a_{02} \\ a_{10} & 0 & a_{11} & 0 & a_{12} & 0 \\ 0 & a_{10} & 0 & a_{11} & 0 & a_{12} \\ a_{20} & 0 & a_{21} & 0 & a_{22} & 0 \\ 0 & a_{20} & 0 & a_{21} & 0 & a_{22} \end{bmatrix} \qquad [4.10]$$

where the a's are the elements of $[D_y^n]$.

Then, $[A]$ from equation 4.3 is just the sum of $[C_A^2]$ and $[C_B^2]$. This approach allows for easy replacement of the appropriate collocation equations with boundary condition equations. Value boundary conditions are enforced by replacing the row corresponding to the node in question with a row of zeros, and a one on the diagonal. The appropriate entry in $\{F\}$ is then changed to the value that is to be given that node. Other types of boundary conditions may also be enforced as illustrated in the one dimensional test problem.

Unfortunately, this technique of rearranging the matrices produces very large matrices which must be factored and solved. The test program was able to overflow available machine storage space (on an IBM 3090 mainframe computer) on a relatively small domain. Suitable reuse of matrix space was able to dodge this problem, but it was always a concern.

4.2 Sample Problem

Good results were obtained for a small domain problem with a known exact solution, a square with zero boundary conditions and a sinusoidal forcing function. First, an exact solution was chosen:

$$u_{exact} = \sin(\pi x)\,\sin(\pi y) \qquad [4.11]$$

Differentiating this solution twice in each direction gives the forcing function:

$$f(x,y) = -2\pi^2 \sin(\pi x)\,\sin(\pi y) \qquad [4.12]$$

The solution converged to the exact solution in a very small number of terms. Figures 4, 5 and 6 show a comparison of the exact solution and approximate solution for three different numbers of terms. Note the quick convergence to a near total lack of error between the approximate and exact curves.

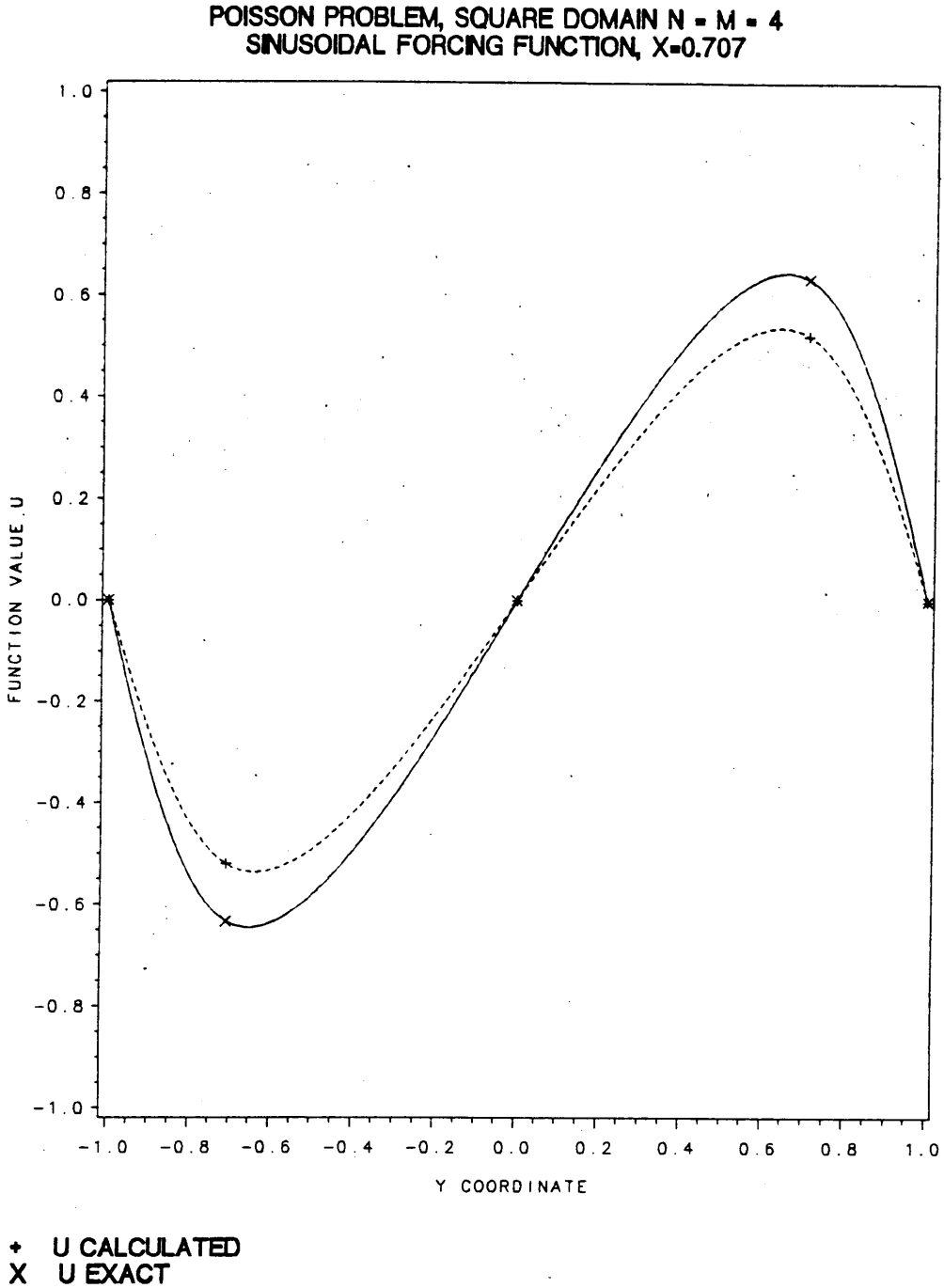

Figure 4. Poisson Solution for N = 4

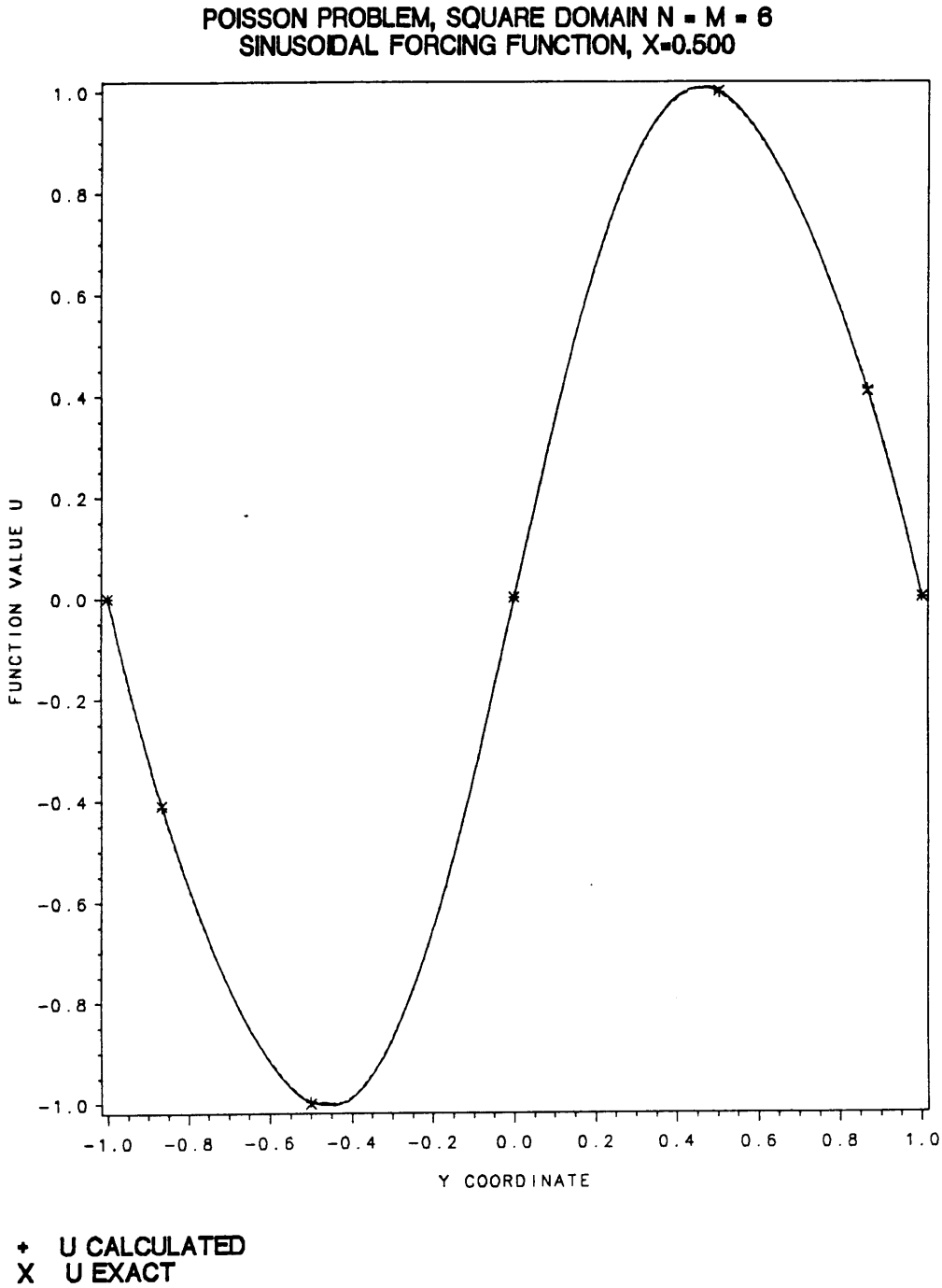

Figure 5. Poisson Solution for N = 6

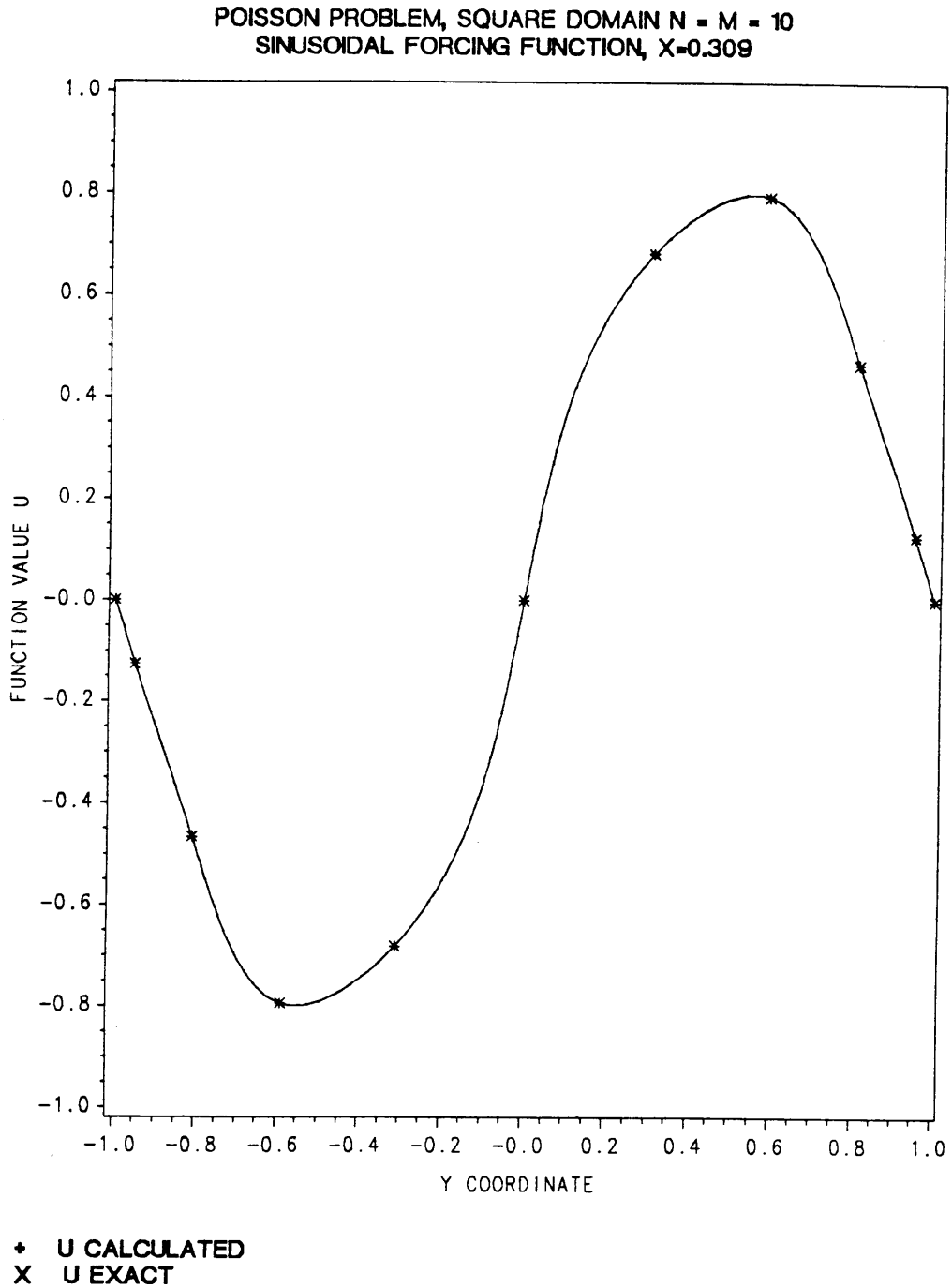

Figure 6. Poisson Solution for N = 10

5.0 Bratu's Equation Problem

Bratu's equation is a non-linear two dimensional differential equation. Further, its solution is generally multi-valued. The two parameter Bratu's equation is:

$$u_{xx} + u_{yy} + \lambda e^{\frac{u}{1+\varepsilon u}} = 0 \qquad [5.1]$$

where

u is the dependent variable

λ is a parameter

ε is a parameter.

Boyd[10] notes that Bratu's equation describes phenonena "ranging from chemical reactor theory and radiative heat transfer to the expansion of the universe." Seydel[12] mentions the equation in relation to the steady state temperature distribution in a reacting material. He describes ε as the product of the gas constant and the surface temperature over the Arrhenious activation energy ($\frac{RT_a}{E}$) and λ as a "non-dimensional lumped parameter."

The equation is solved on a domain between ± 1, i.e.

$$0 \leq |x|, |y| \leq 1$$

with zero boundary conditions on all edges, i.e.

$$u(\pm 1, y) = u(x, \pm 1) = 0 \qquad [5.2]$$

This problem may be viewed as a Poisson's equation with a forcing function that depends on the dependent variable u in a nonlinear fashion. Thus, it was solved using the Poisson solver discussed in the previous section. However, the multivalued solution of the problem required a few significant additions.

Previous solutions of equation have assumed, but not proved, the solution to symmetric in x and y. This assumption does simplify the problem significantly, but was not made in this solution.

5.1 Perturbation Form of Differential Equation

First, rather than using the differential equation itself, a perturbation form of the equation is used. The following substitutions are made:

$$u = u + \Delta u \qquad [5.3]$$

$$\lambda = \lambda + \Delta\lambda \qquad [5.4]$$

Epsilon is not expanded, as the solution is performed for a particular value of this parameter. All second order perturbation terms, such as $\Delta u \Delta\lambda$ are discarded. After some algebraic rarranging the governing equation becomes

$$\Delta u_{xx} + \Delta u_{yy} + a\Delta u + g\Delta\lambda = -u_{xx} - u_{yy} - g\lambda \qquad [5.5]$$

where

$$g = e^{\frac{u}{1+\varepsilon u}} \qquad [5.6]$$

and

$$a = \lambda g\left(1 + \frac{\varepsilon u}{1 + \varepsilon u^2}\right) \qquad [5.7]$$

The right hand side of this equation is the same as the original governing equation, and is zero when at a converged solution point. Previous work with this equation (see for example Boyd[10] and Kapania[11]) has assumed a solution that is symmetric in x and y. No such assumption has been made in this work, with consequences to be discussed later.

5.2 Solution Technique

An initial $u = 0$ everywhere solution for $\lambda = 0$ is assumed. This starting point satisfies the differential equation and boundary conditions. Then, a value of one is assigned to delta lambda. The perturbation form of the differential equation from this point is:

$$\Delta u_{xx} + \Delta u_{yy} = -\Delta\lambda \qquad [5.8]$$

The solution of this Poisson equation gives the change in $u(x,y)$ for a unit $\Delta\lambda$ The entire solution, and the $\Delta\lambda$ are then scaled back to enforce a desired small Δu at the center of the domain. This solution is used to start an iteration at the new $u(0,0)$ value. λ becomes an unknown, replacing $u(0,0)$ which becomes a boundary condition. The perturbation form of the differential equation is then repetitively solved until $\Delta\lambda$ becomes very small.

Once the solution has converged at a particular value of $u(0,0)$, the technique is repeated to advance to a new $u(0,0)$. The general advancement equation is just equation 5.5 with the right hand set equal to zero:

$$\Delta u_{xx} + \Delta u_{yy} + a\Delta u = -g\Delta\lambda \qquad [5.9]$$

The matrix representation of this equation is:

$$[[A] + [a]]\{\Delta u\} = -\Delta\lambda\{g\} \qquad [5.10]$$

Where $[A]$ is the rearranged differential operator from the Poisson solver, and $\Delta\lambda$ starts at a value of 1 and is decreased based on the condition number of the operator matrix. Note, also, that $[a]$ is a diagonal matrix.

Once $\{\Delta u\}$ is found, it is scaled back for a small $\Delta\lambda$ Again, the value of $u(0,0)$ is fixed, and $\Delta\lambda$ becomes a variable. The differential equation is solved for $\Delta\lambda$ and this equation is solved at the center node rather than solving for $u(0,0)$

$$\Delta\lambda = \frac{-1}{g}\left[u_{xx} + u_{yy} + \lambda g + \Delta u_{xx} + \Delta u_{yy} + a\Delta u\right] \qquad [5.11]$$

Then, the modified matrix system of equations is iterated upon until the $\Delta\lambda$ variable becomes small.

This technique of small steps allows the solution to continue moving along a path, rather than leaping to another section of the path.

5.3 Results

Figures 7, 8 and 9 shows the values of λ found for each value of $u(0,0)$ and ε, for $N = 4, 6$ and 10 The only difference between the plots is the point where the solution leaves the smooth, symmetric solution curve. These plots match the one found in Kapania's Bratu solution paper[11] up to the points where the current solution diverges. Examination of the eigenvectors of the matrix at this point shows unsymmetric eigenvectors. This suggests a bifurcation of the solution into multiple, possibly unsymmetric, solution paths.

In fact, after the solution stops following the smooth, symmetric solution curve, it continues to converge to valid solutions that are non-symmetric, and that do not lie along a single simple path such as found in the symmetric region. An eigenvector following technique might be able to continue following a single path in this region.

Figures 10-15 show the complete $u(x,y)$ solution of the equation at various values of ε, λ and $u(0,0)$. Figure 10 shows the solution at a point on the left of figure 7, prior to the peak. Figures 11-13 show the solution at the peak of the λ, $u(0,0)$ curves for three

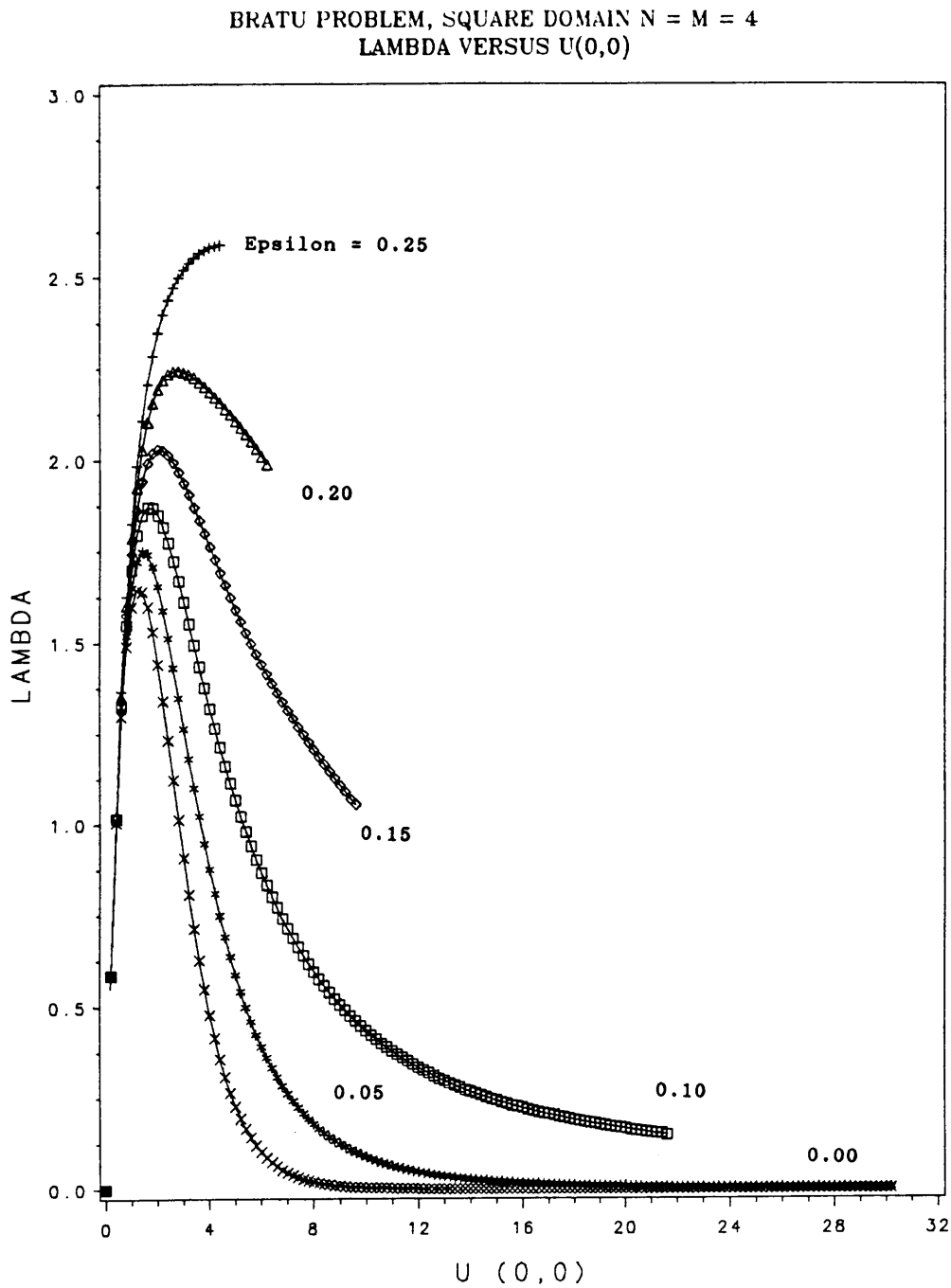

Figure 7. Two Parameter Bratu's Equation Solution N = 4

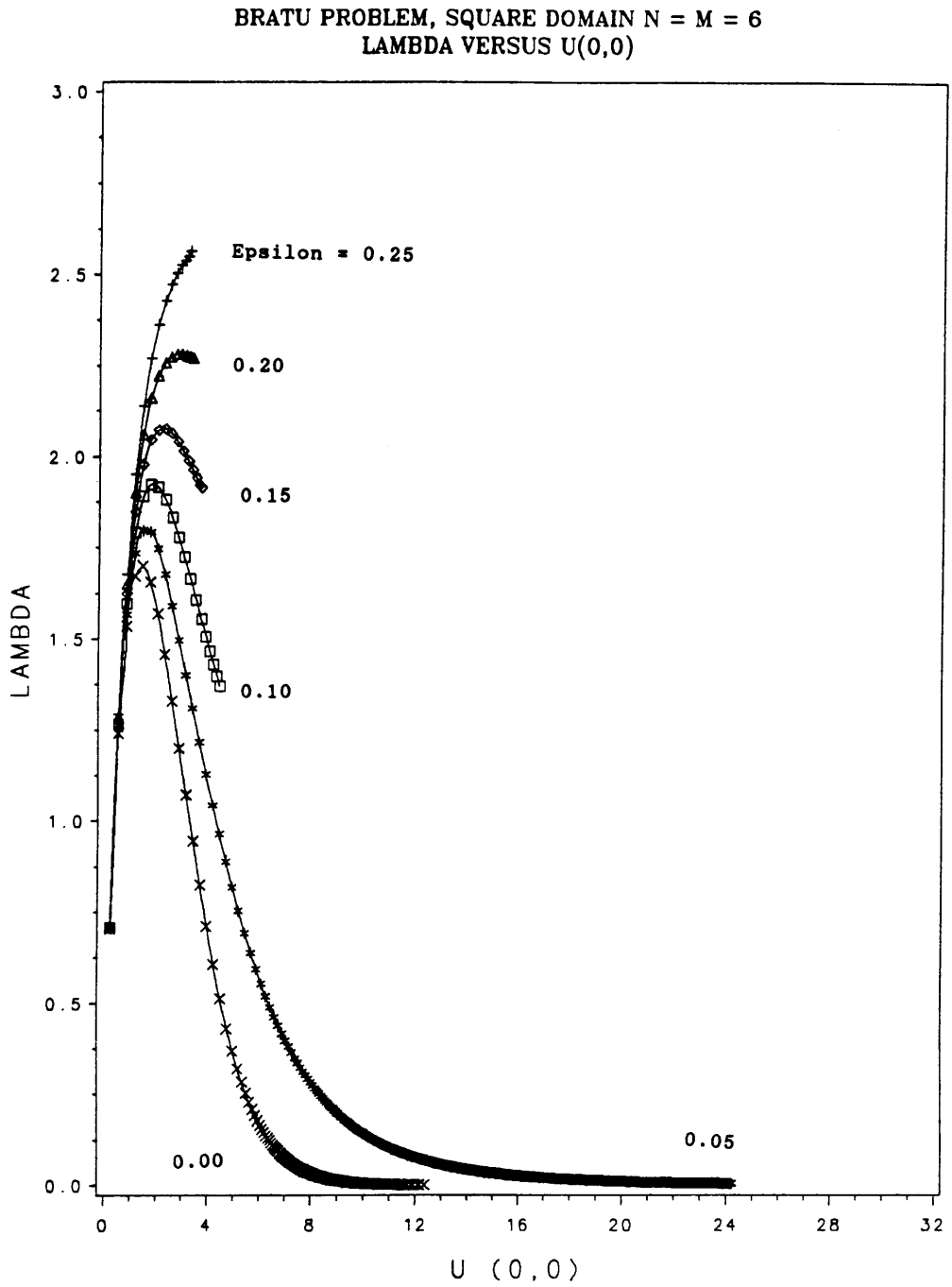

Figure 8. Two Parameter Bratu's Equation Solution N = 6

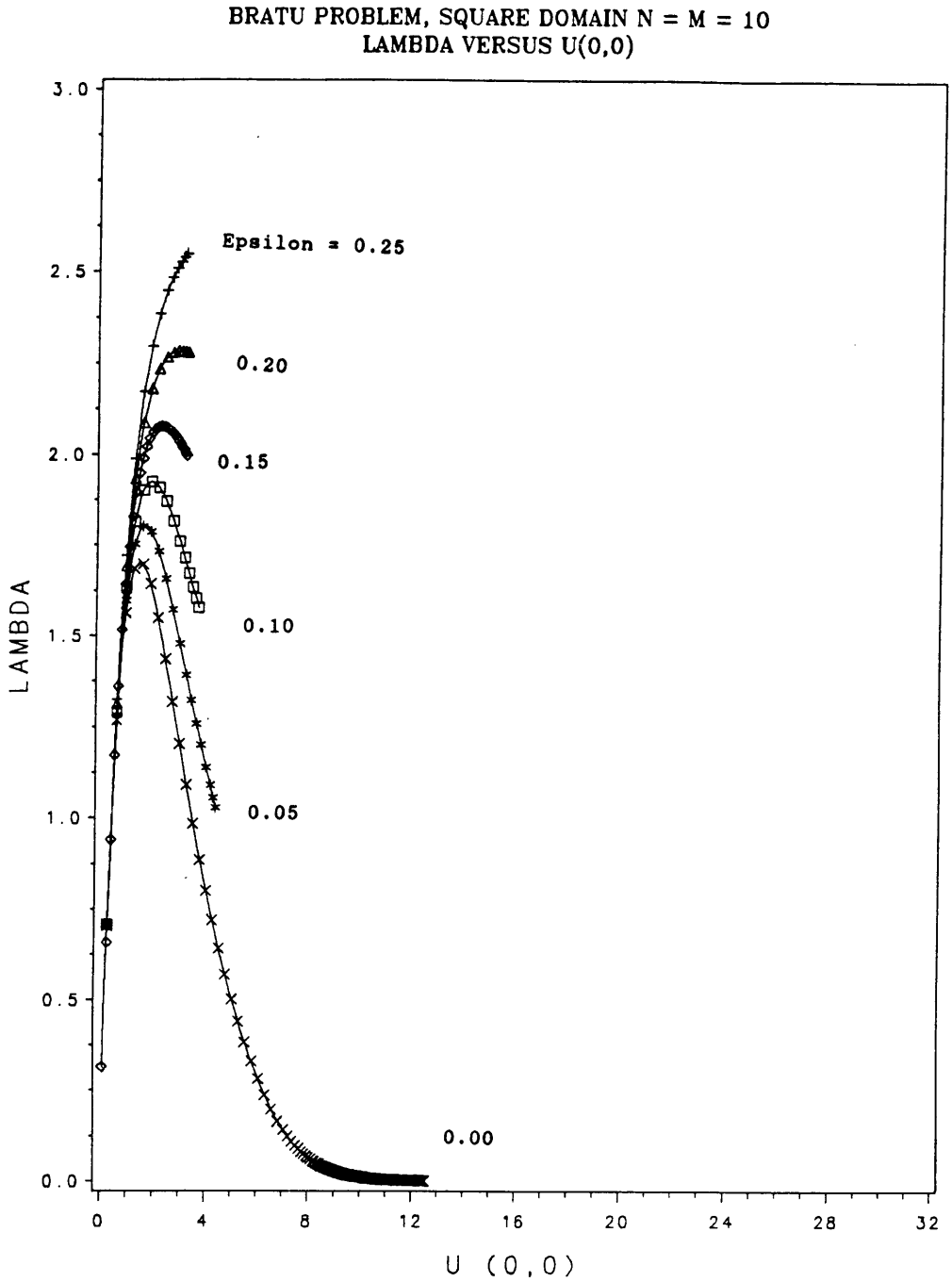

Figure 9. Two Parameter Bratu's Equation Solution N = 10

values of ε. Figure 14 shows the solution at an "end point", the last symmetric solution the program was able to converge to before starting to follow an unsymmetric path. Figure 15 shows a non-symmetric solution. Note, the peak in this figure is not at the center of the domain.

The current algorithm was able to handle the curve and reversal of the solution path. It tracked the multiple valued solution of the problem for a variety of combinations of the parameters λ and ε . It did this extremely effieciently, requiring very few node points to track these curves extremely accurately.

BRATU PROBLEM, SQUARE DOMAIN N = M = 10
U VERSUS X AND Y, EP = 0.05 LAM = 0.706
U(0,0)= 0.25, PRE PEAK POINT

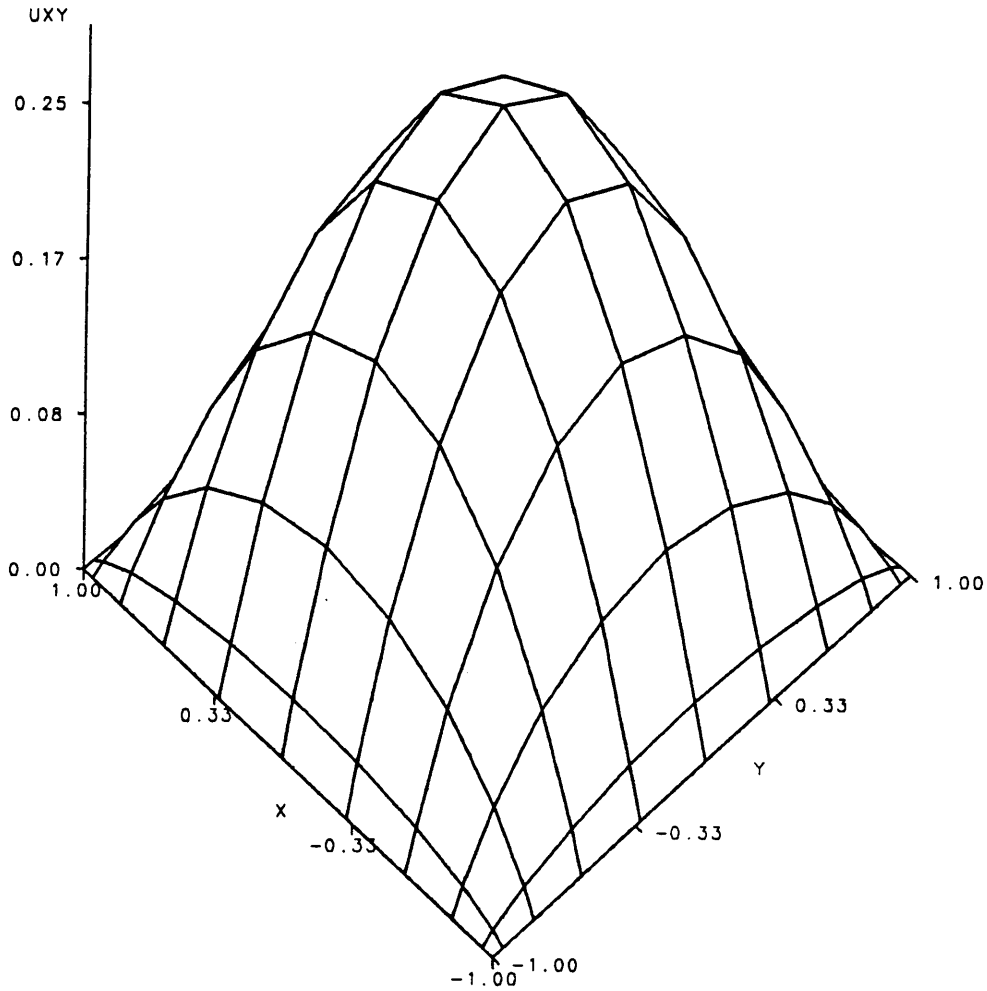

Figure 10. Bratu's Equation Solution, Pre-Peak

BRATU PROBLEM, SQUARE DOMAIN N = M = 10
U VERSUS X AND Y, EP = 0.05 LAM = 1.802
U(0,0) = 1.503, PEAK OF CURVE

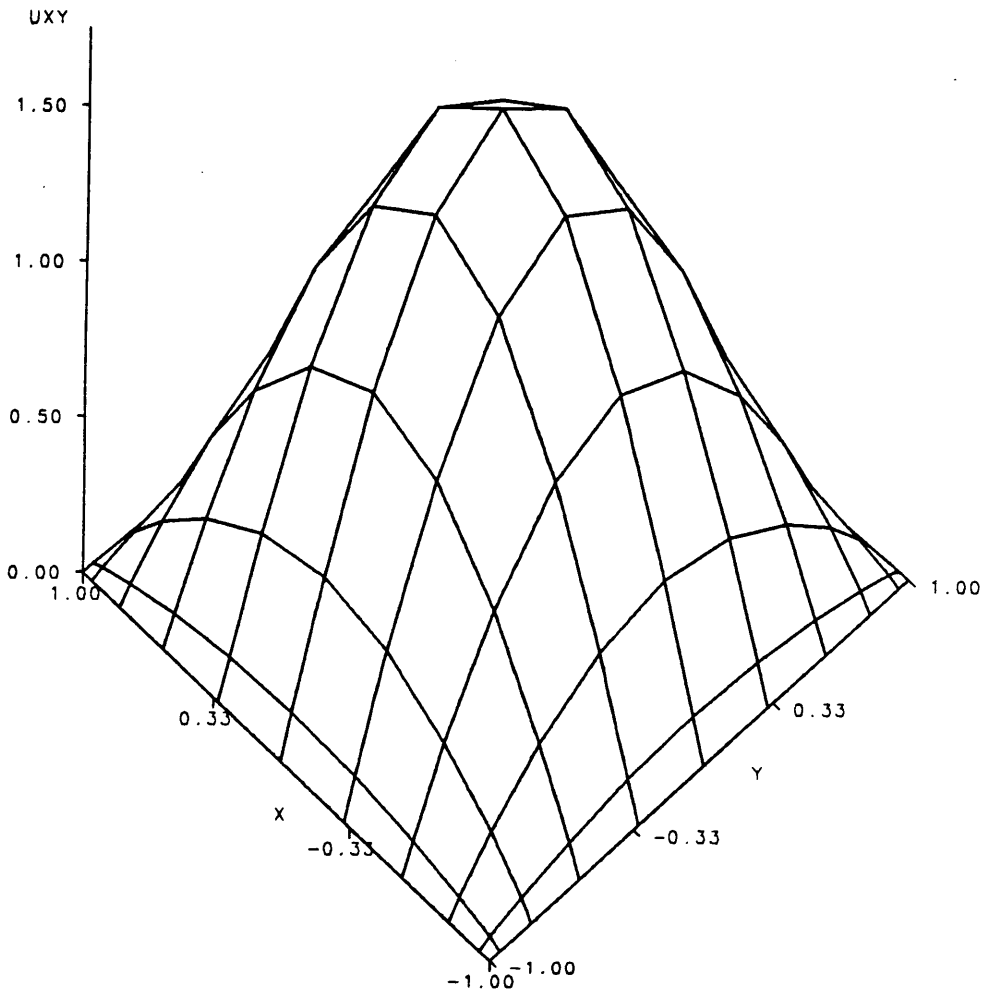

Figure 11. Bratu's Equation Solution, Peak, Epsilon = 0.05

BRATU PROBLEM, SQUARE DOMAIN N = M = 10
U VERSUS X AND Y, EP = 0.15 LAM = 2.078
U(0,0)= 2.179, PEAK POINT

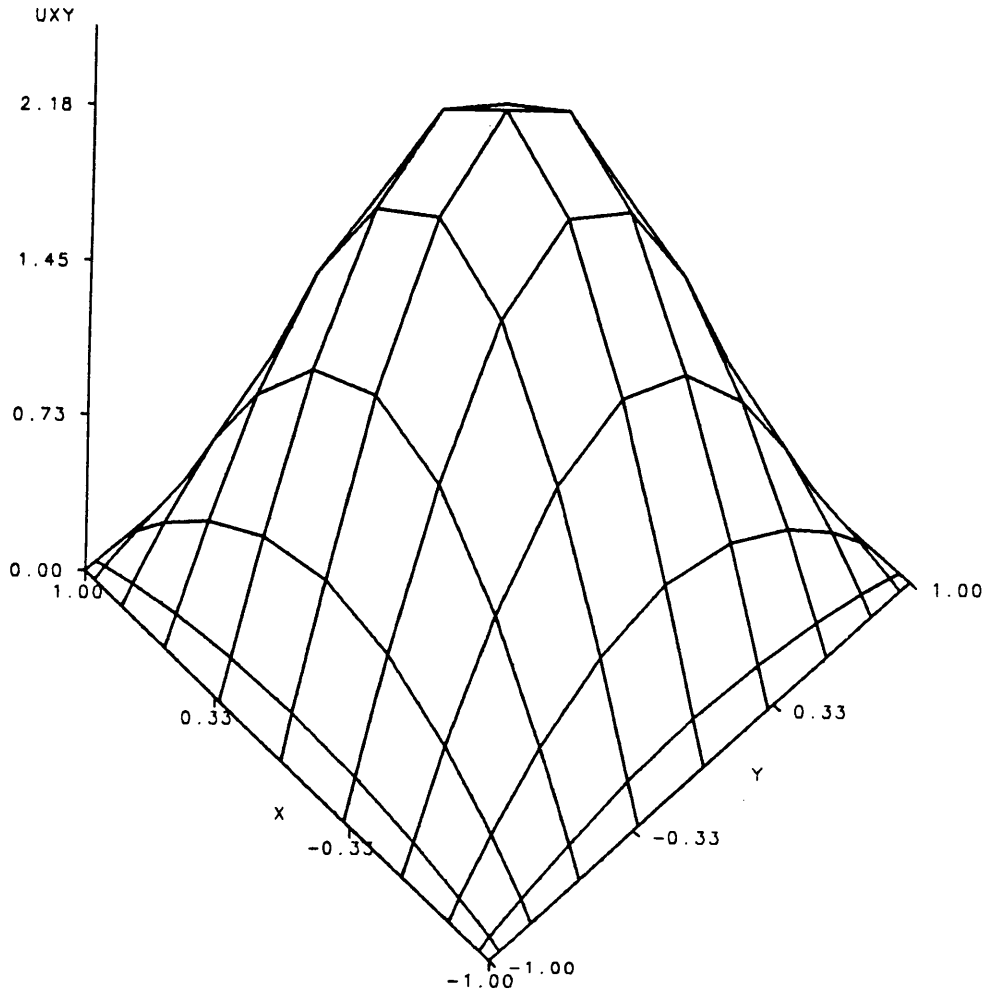

Figure 12. Bratu's Equation Solution, Peak, Epsilon = 0.15

BRATU PROBLEM, SQUARE DOMAIN N = M = 4
U VERSUS X AND Y, EP = 0.25 LAM = 2.582
U(0,0)= 5.071, PEAK POINT

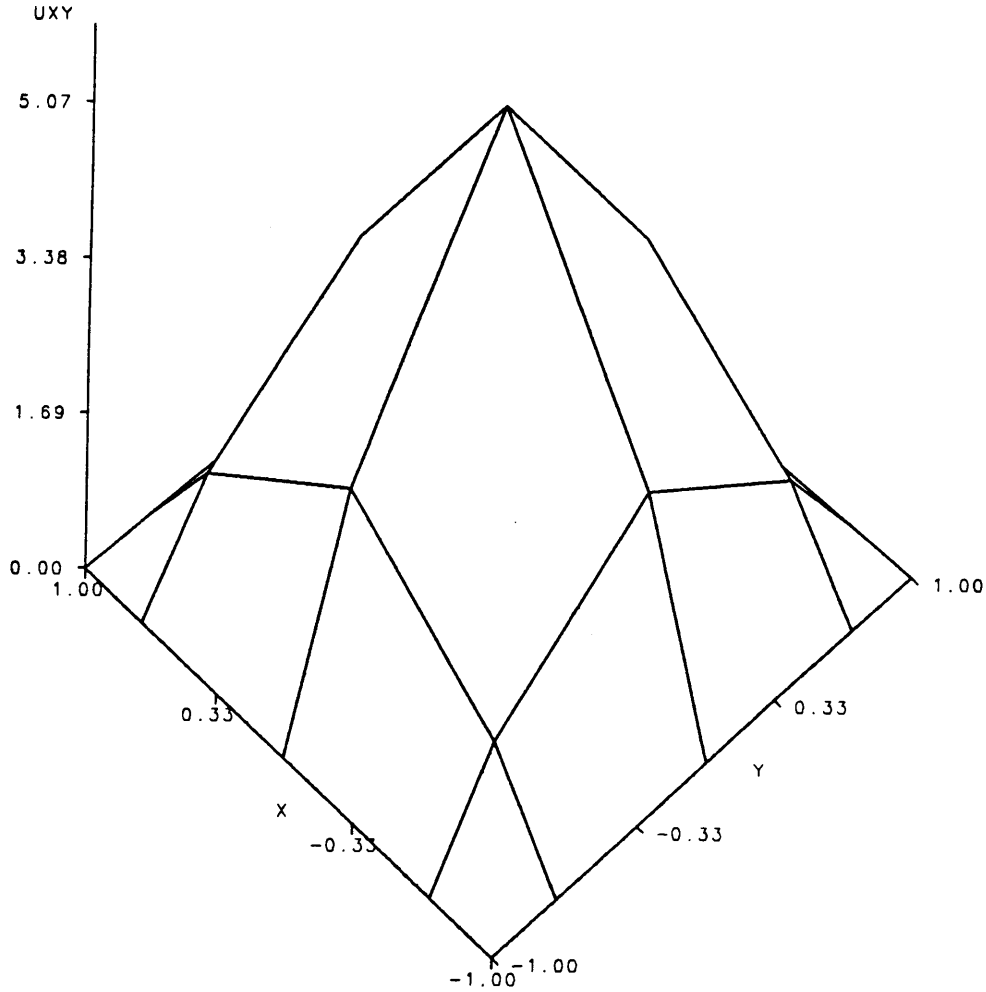

Figure 13. Bratu's Equation Solution, Peak, Epsilon = 0.25

BRATU PROBLEM, SQUARE DOMAIN N = M = 6
U VERSUS X AND Y, EP = 0.15 LAM = 1.922
U(0,0)= 3.620, PLOT 8 END POINT

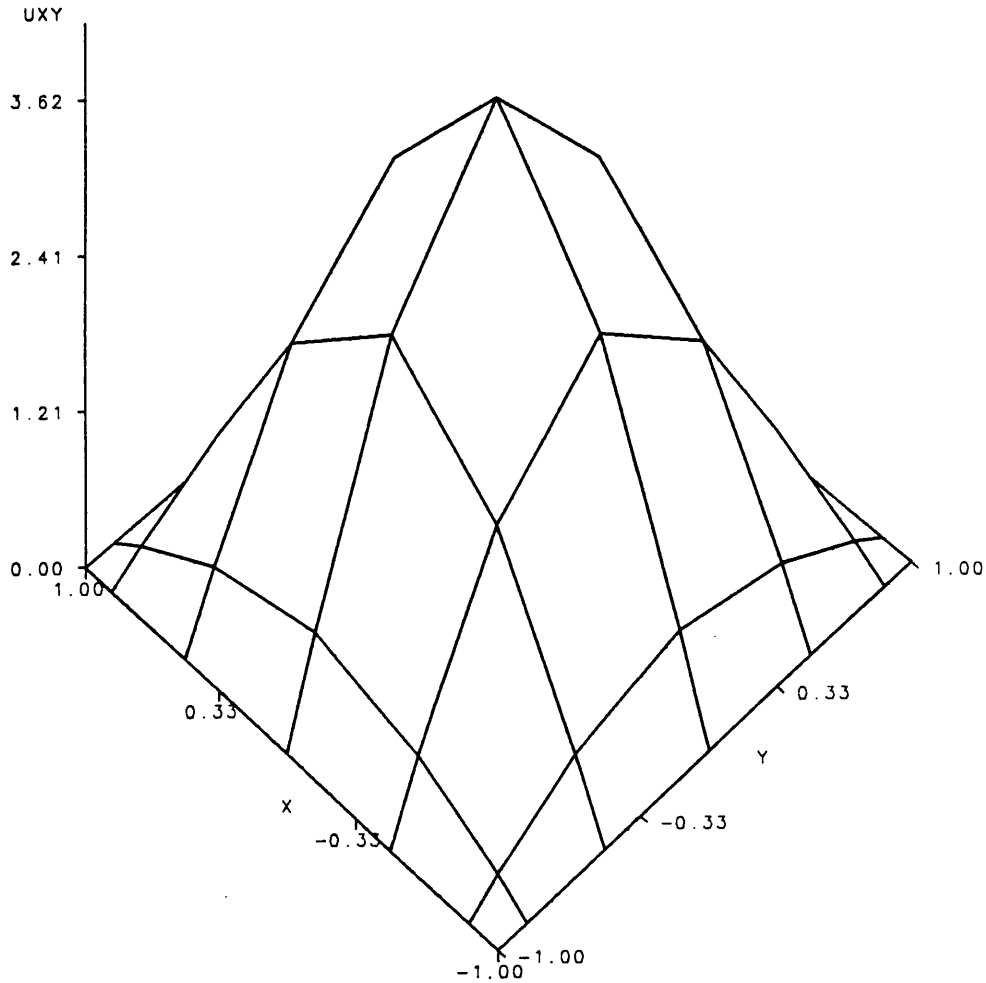

Figure 14. Bratu's Equation Solution, End Point

BRATU PROBLEM, SQUARE DOMAIN N = M = 10
U VERSUS X AND Y, EP = 0.05 LAM = 0.037
U(0,0)= 8.342, UNSYMMETRIC SOLUTION

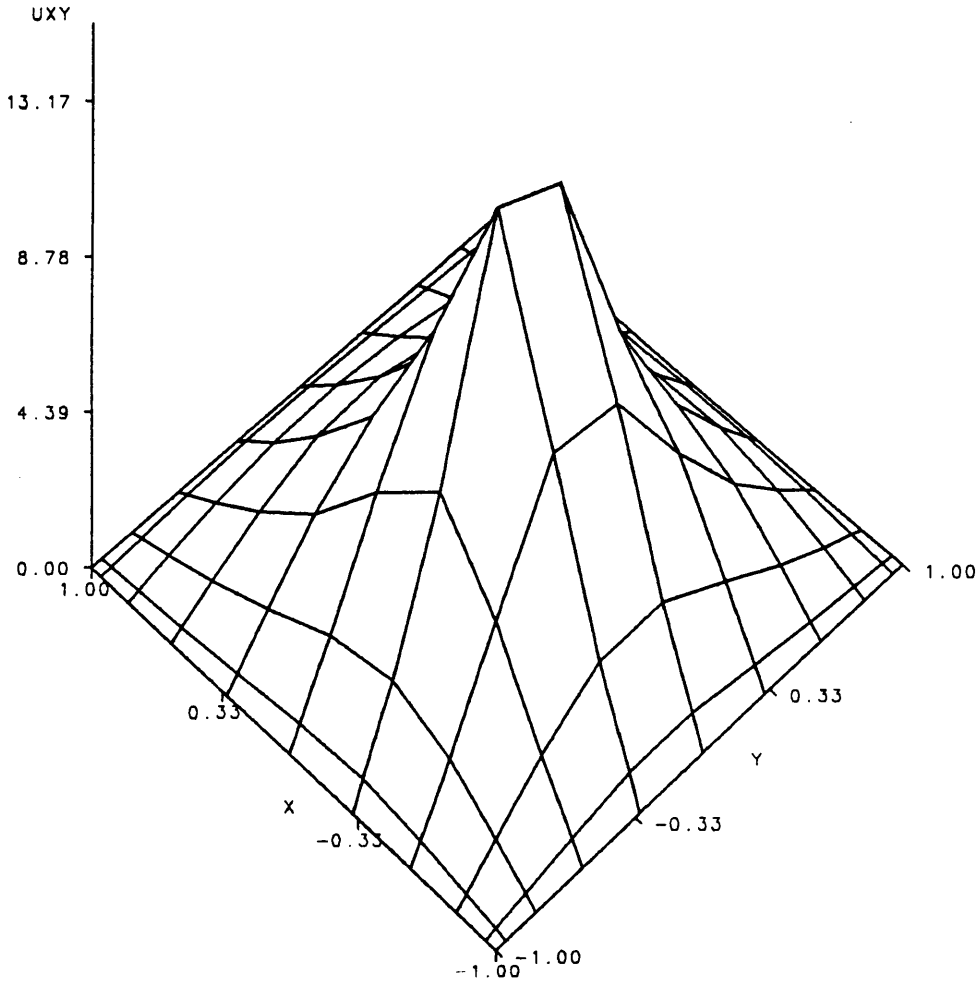

Figure 15. Bratu's Equation Solution, Unsymmetric solution

6.0 Concluding remarks

The spectral methods are extremely effective and efficient techniques for the solution of differential equations. They can give truely phenomenal performance when applied to appropriate problems. However, implementation is far from trivial, with pitfalls for the unwary.

Modeling of a one dimensional differential equation was reasonably quick and easy. The tapered bar problem showed the basic techniques behind the spectral methods. It also revealed the incredible potential of the methods.

The two dimensional Poisson problem was not a simple extension of the one dimesional theory. After some rearrangement of the differential opererators, the equations may be placed in an easy to work with configuration. Boundary conditions may be enforced, then normal matrix solution techniques may be used. This rearragement technique is easy to perform, but squares the dimensions of the matrices involved. The resultant matrix is sparce and could possibly be stored in less space if significant work were done to determine an efficient solution technique. Other researchers have used eigenvector

techniques to solve the matrix equations. This approach is computationally slower, but requires less storage. Neither technique is particularly applicable to large domain problems, but the technique works, requiring only better matrix theory to become useful in much larger cases.

The two parameter Bratu's equation was studied to show the spectral method's applicability to highly non-linear problems. The technique was adapted easily to the problem, with very nice results. An interesting discovery was made during the study. The equations, previously assumed to have symmetric solutions, were found to have regions where the solutions can be non-symmetric in the x and y directions. Away from these regions, the current program was able to track accurately and efficiently the difficult multi-valued solution.

So, the chebyshev-collocation spectral method is an efficient technique for the solution of differential equations. Boundary condition treatment can be a problem and programmer effort can be significant. The family of spectral techniques show a lot of promise, but need some more refinement before they can be used as easily and frequently as the finite element technique.

Appendix A First Derivative Operator

Peyret[2] gives the following functional forms for the entries in the D^1 matrix.

For $0 \leq i,k \leq N$, $i \neq k$

$$d_{i,k} = \frac{c_i}{c_k} \frac{(-1)^{i+k}}{x_i - x_k} \qquad [A.1]$$

Then the diagonal terms are, $1 \leq i \leq N-1$

$$d_{i,i} = -\frac{x_i}{2(1-x_i^2)} \qquad [A.2]$$

and

$$d_{0,0} = -d_{N,N} = \frac{2N^2 + 1}{6} \qquad [A.3]$$

where

$$c_0 = c_N = 2 \qquad [A.4]$$

and

$$c_j = 1 \text{ for } 1 \leq j \leq N - 1 \qquad [A.5]$$

Appendix B One dimensional Explicit test program

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     PROGRAM TO SOLVE THE DIFFERENTIAL EQUATION FOR A
C     BAR HANGING UNDER IT'S OWN WEIGHT BY A CHEBYSHEV
C     COLLOCATION METHOD
C     (C) 1987 LLOYD B. ELDRED
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      REAL*8 C(201),T(201,201),ALPHA(201),BETA(201,201),GAMMA(201),
     $A(201,201),B(201),ABAR(201),ABARPR(201)
      REAL*8 PI,GLDE,TR,TOL,ABR,ABRPR,ATEMP,X,B1,B2,U,ERROR,UOLD
      REAL*8 OLDERR
      INTEGER P,PT,IPVT(201)
      PI=3.141592654
      GLDE=1.D-04
      N=3
```

```
      TR=0.10
      TOL=0.0001
      C(1)=2.0
      GAMMA(1)=1.0
      DO 10 I=2,201
      C(I)=1.0
      ITEMP=I-1
      IEF=IEVEN(ITEMP)
   10 GAMMA(I)=DFLOAT(2*IEF-1)
    1 WRITE(6,1000) TR
 1000 FORMAT(//' VARIABLE AREA BAR CASE -- TR= ',F4.2,' (',
     $'CIRC. CROSS SEC.)'/' TIP DEFLECTIONS'/'  N',T14,'DEFLECTION')
      OLDU=0.0
      OLDERR=1000.D0
      IFBC=0
C     COMMENT OUT NEXT LINE TO ENFORCE FORCE B.C. AT EVERY TR
C      IF(TR.NE.1.0) GOTO 20
      IFBC=1
      CALL AREA(1.0,ABR,ABRPR,TR)
      ALPHA(1)=0.0
      DO 15 I=2,201
      ITEMP=I-1
      IEF=IEVEN(ITEMP)
      ATEMP=0.0
      DO 12 J=IEF,ITEMP,2
   12 ATEMP=ATEMP+1.0/C(J+1)
```

```
 15 ALPHA(I)=2.0*DFLOAT(ITEMP)*ATEMP*ABR
CCCCCCCCCCCCCCCCCCCC
C
C   MAIN LOOP
C
CCCCCCCCCCCCCCCCCCCC
 20 NP1=N+1
    DO 30 I=1,NP1
    T(1,I)=1.0
    X=DCOS(PI*DFLOAT(NP1-I)/DFLOAT(N))
    CALL AREA(X,ABAR(I),ABARPR(I),TR)
    T(2,I)=X
    DO 30 J=3,NP1
 30 T(J,I)=2.0*X*T(J-1,I)-T(J-2,I)
    DO 60 I=2,NP1
    BETA(I,1)=0.0
    BETA(I,2)=ABARPR(I)
    DO 60 P=3,NP1
    PT=P-1
    IPEF=IEVEN(PT)
    IPOF=1-IPEF
    B1=0.0
    DO 40 K=IPEF,PT,2
 40 B1=B1+T(K+1,I)/C(K+1)
    B2=0.0
    PT2=P-2
```

```
      DO 50 K=IPOF,PT2,2
   50 B2=B2+DFLOAT(PT*PT-K*K)*T(K+1,I)/C(K+1)
   60 BETA(I,P)=DFLOAT(PT)*(2.0*ABARPR(I)*B1+B2*ABAR(I))
CCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     ASSEMBLE MATRICES
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCC
      DO 80 J=1,NP1
      B(J)=-GLDE*ABAR(J)/4.0
      A(1,J)=GAMMA(J)
      DO 80 I=2,NP1
   80 A(I,J)=BETA(I,J)
      B(1)=0.0
C  COMMENT OUT NEXT LINE TO ENFORCE FORCE B.C. AT ALL TR
C      IF(TR.NE.1.0) GOTO 70
      DO 85 J=1,NP1
   85 A(NP1,J)=ALPHA(J)
      B(NP1)=0.0
   70 CONTINUE
C      WRITE(6,250)
C 250 FORMAT(/' THE MATRIX A IS:'/)
C      DO 86 I=1,NP1
C  86 WRITE(6,300) (A(I,J),J=1,NP1)
C 300 FORMAT(4E15.6)
C      WRITE(6,350)
```

```
C 350 FORMAT(/' THE VECTOR B IS:'/)
C     WRITE(6,400) (B(I),I = 1,NP1)
C 400 FORMAT(E15.6)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    SOLVE SYSTEM OF EQUATIONS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL DGEF(A,201,NP1,IPVT)
      CALL DGES(A,201,NP1,IPVT,B)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   CHECK CONVERGENCE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      U = 0.0
      DO 95 J = 1,NP1
   95 U = U + B(J)*T(J,NP1)
      WRITE(6,600) N,U
  600 FORMAT(I4,T10,E15.6)
      ERROR = ABS((U-UOLD)/U)
      IF(ERROR.LT.TOL.AND.OLDERR.LT.TOL) GOTO 97
      N = N + 1
      UOLD = U
      OLDERR = ERROR
```

```
      IF(N.LT.200) GOTO 20
CCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   OUTPUT RESULTS
C
CCCCCCCCCCCCCCCCCCCCCCCCCC
      WRITE(6,700)
  700 FORMAT(' FOLLOWING FAILED TO CONVERGE IN 200 TERMS:')
   97 WRITE(6,100)
  100 FORMAT(/' EXPANSION COEFFICIENTS'/'  N',T18,'A(N)')
      DO 90 I=1,NP1
      I2=I-1
   90 WRITE(6,200) I2,B(I)
  200 FORMAT(T2,I3,T10,E15.6)
      IF(IFBC.EQ.1) WRITE(6,260)
  260 FORMAT(' *** NOTE: FORCE B.C. WAS USED FOR THIS CASE ***')
      WRITE(6,500)
  500 FORMAT(/' DEFLECTIONS:'/'  X',T20,'UBAR(X)',T52,'UBAR/GLDE')
      DO 98 I=1,21
      X=DFLOAT(I-11)/10.0D0
      U=0.0
      DO 99 J=1,NP1
   99 U=U+B(J)*CHEB(J-1,X)
      U2=U/GLDE
   98 WRITE(6,800) X,U,U2
  800 FORMAT(F6.1,T14,E15.6,T47,F14.7)
```

```
      TR=TR+0.10
      N=3
      IF(TR.GT.4.00) STOP
      GOTO 1
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINES
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C  FUNCTION IEVEN(I) --- DETERMINES IF A GIVEN INTEGER
C                       IS EVEN OR NOT
C         INPUT: I, AN INTEGER TO BE TESTED
C         OUTPUT: 1 IF EVEN
C                 0 IF ODD
      FUNCTION IEVEN(I)
      IEVEN=0
      IF ((I/2).EQ.((I+1)/2)) IEVEN=1
      RETURN
      END
C  SUBROUTINE AREA -- CONTAINS INFORMATION ABOUT THE
C                     AREA AND ITS DERIVATIVE
C                     FOR THE BAR UNDER CONSIDERATION
C         INPUT: X, THE SPACIAL COODINATE OF THE POINT CONSIDERED
C                TR, RATIO OF TIP DIA. TO ROOT DIA.
```

```
C         OUTPUT: AR, THE RATIO OF THE AREA'S DERIVATIVE TO THE
C                 AREA (APRIME/AREA)
      SUBROUTINE AREA(X,ABAR,ABARPR,TR)
      REAL*8 X,PI,ABAR,ABARPR,TR,RODL,A0,TEMP
      PI=3.141592654
C CALCULATE THE DIAMETER AS A FUNCTION OF X, KEEPING A UNIT
C AREA ROOT
      RODL=0.25
      A0=PI*RODL*RODL
      TEMP=(TR-1.0)*(1.0+X)/2.0 + 1.0
      ABAR=A0*TEMP*TEMP
      ABARPR=A0*(TR-1.0)*TEMP
      RETURN
      END
C FUNCTION CHEB(N,X) --- CALCULATES THE VALUE OF THE NTH CHEB.
C                 POLYNOMIAL AT THE POINT X
C         INPUT: N, THE ORDER OF THE POLYNOMIAL
C               X, THE POINT TO CALCULATE THE VALUE AT
C         OUTPUT: THE VALUE OF THE FUNCTION AT THAT POINT.
      FUNCTION CHEB(N,X)
      REAL*8 CHEB,X
      CHEB=DCOS(DFLOAT(N)*DARCOS(X))
      RETURN
      END
```

Appendix C One dimensional Matrix text program

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   PROGRAM TO SOLVE PROBLEM OF TAPERED BAR HANGING
C     UNDER IT'S OWN WEIGHT BY A CHEBYSHEV-SPECTRAL
C     METHOD USING A MATRIX MULTIPLY METHOD TO
C     CALCULATE DERIVATIVES
C       BY LLOYD B ELDRED
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (NMX=30)
      PARAMETER (NM2=31)
      REAL*8 D(0:NMX,0:NMX),DA(0:NMX,0:NMX),DB(0:NMX,0:NMX),C(0:NMX)
      REAL*8 X(0:NMX),AR(0:NMX,0:NMX),ARDB(0:NMX,0:NMX),TIP(0:NMX)
      REAL*8 RHS(0:NMX)
      INTEGER IPVT(0:NMX)
```

```
      PI=3.1415935898D0
      TR=0.1D0
      DO 5 I=0,NMX
      C(I)=1.0D0
      DO 5 J=0,NMX
      AR(I,J)=0.0D0
    5 CONTINUE
      C(0)=2.0D0
   10 N=3
      OLDRHSN=0.0D0
   20 C(N)=2.0D0
      DO 30 I=0,N
      RHS(I)=-1.0D-04 / 4.0D0
      X(I)=DCOS(PI*DFLOAT(I)/DFLOAT(N))
      CALL AREA(X(I),TR,A,APRIME)
      AR(I,I)=APRIME/A
   30 CONTINUE
      CALL D1(DB,X,C,N,NMX)
      CALL D2(DA,DB,N,NMX)
      DO 35 I=0,N
      TIP(I)=DB(N,I)
   35 CONTINUE
      TIP(N)=TIP(N)+AR(N,N)
C     WRITE(6,110) ((DB(I,J),J=0,N),I=0,N)
  110 FORMAT(' D1 MATRIX:',255(/4F14.7))
C     WRITE(6,120) ((DA(I,J),J=0,N),I=0,N)
```

```
 120 FORMAT(' D2 MATRIX:',255(/4F14.7))
C     WRITE(6,130) ((AR(I,J),J=0,N),I=0,N)
 130 FORMAT(' AR:',255(/4F14.7))
      NP1=N+1
      CALL MATMUL(AR,NM2,NM2,DB,NM2,NM2,ARDB,NM2,NM2,NP1,NP1
     $,NP1)
C     WRITE(6,140) ((ARDB(I,J),J=0,N),I=0,N)
 140 FORMAT(' ARDB:',255(/4F14.7))
      CALL MATADD(DA,NM2,NM2,ARDB,NM2,NM2,D,NM2,NM2,NP1,NP1)
C   ENFORCE BOUNDARY CONDITIONS IE NO DEFLECTION AT ROOT
C       AND DU/DX=0 AT TIP
      DO 40 I=0,N
      D(0,I)=0.0D0
      D(N,I)=TIP(I)
  40  CONTINUE
      D(0,0)=1.0D0
      RHS(0)=0.0D0
      RHS(N)=0.0D0
C     WRITE(6,150) ((D(I,J),J=0,N),I=0,N)
 150 FORMAT(' FINAL MATRIX:',255(/4F14.7))
      CALL DGEF(D,NM2,NP1,IPVT)
      CALL DGES(D,NM2,NP1,IPVT,RHS)
C     WRITE(6,160) TR,N,(RHS(I)/1.0D-4,I=0,N)
C 160 FORMAT(' TR = ',F5.2,' CASE N = ',I5,
C    $/' DEFLECTIONS:',100(/F14.7))
      DIFF=DABS((RHS(N)-OLDRHSN)/RHS(N))
```

```
      OLDRHSN=RHS(N)
      IF(DIFF.GT.0.01.AND.N.LE.20) THEN
        C(N)=1.0D0
        N=N+1
        GOTO 20
      ENDIF
      WRITE(6,100) TR,N,(RHS(I)/1.D-4,I=0,N)
  100 FORMAT(' TR = ',F5.2,' CASE CONVERGED IN',I5,' STEPS'
     $/' DEFLECTIONS:',100(/G14.7))
      IF(TR.GE.4.0) STOP
      TR=TR+0.1
      C(N)=1.0D0
      N=3
      GOTO 10
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE D1 - PUTS FIRST DERIVATIVE
C                     MATRIX IN 'D' MATRIX
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE D1(D,X,C,N,MAXSIZE)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      REAL*8 D(0:MAXSIZE,0:MAXSIZE),X(0:MAXSIZE),C(0:MAXSIZE)
      DO 10 I=0,N
      DO 10 J=0,N
```

```
      IF(I.EQ.J) GOTO 10
      D(I,J)=C(I)/C(J)*(-1)**(I+J)/(X(I)-X(J))
   10 CONTINUE
      DO 20 I=1,N-1
      D(I,I)=-X(I)/2./(1.-X(I)*X(I))
   20 CONTINUE
      D(0,0)=(2.D0*N*N+1)/6.0D0
      D(N,N)=-(2.D0*N*N+1)/6.0D0
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE D2 - PUTS SECOND DERIVATIVE
C                   MATRIX IN 'D' MATRIX
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE D2(D,D1,N,MAXSIZE)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      REAL*8 D(0:MAXSIZE,0:MAXSIZE),D1(0:MAXSIZE,0:MAXSIZE)
      MP1=MAXSIZE+1
      NP1=N+1
      CALL MATMUL(D1,MP1,MP1,D1,MP1,MP1,D,MP1,MP1,NP1,NP1,NP1)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
C  SUBROUTINE AREA - CONTAINS INFO ABOUT BAR
C                CROSS SECTIONS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE AREA(X,TR,A,APRIME)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      REAL*8 L
      PI=3.1415935898D0
      R0=0.50D0
      L=2.0D0
      R2=R0*R0
      A=PI*R2/L/L*(1.0+(TR-1.0)*(1.0-X)/2.0)**2
      APRIME=-PI*R2/L/L*(TR-1.0)*(1.0+(TR-1.0)*(1.0-X)/2.0)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINE MATMUL, MULTIPLIES UPPER LEFT
C                PORTIONS OF TWO MATRICES
C       A - INPUT MATRIX OF SIZE NA X MA
C       B - INPUT MATRIX OF SIZE NB X MB
C       C - OUTPUT MATRIX OF SIZE NC X MC
C       N,M,NN - AMOUNTS OF A AND B TO MULT:
C               [A]     [B] =   [C]
C              N X NN  NN X M   N X M
```

```
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATMUL(A,NA,MA,B,NB,MB,C,NC,MC,N,M,NN)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I = 1,N
      DO 10 J = 1,M
      C(I,J) = 0.0D0
      DO 10 K = 1,NN
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINE MATADD, ADDS UPPER LEFT
C                PORTIONS OF TWO MATRICES
C      A - INPUT MATRIX OF SIZE NA X MA
C      B - INPUT MATRIX OF SIZE NB X MB
C      C - OUTPUT MATRIX OF SIZE NC X MC
C      N,M - AMOUNTS OF A AND B TO ADD:
C              [A]   [B] =   [C]
C             N X M  N X M   N X M
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATADD(A,NA,MA,B,NB,MB,C,NC,MC,N,M)
```

```
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I = 1,N
      DO 10 J = 1,M
      C(I,J) = A(I,J) + B(I,J)
   10 CONTINUE
      RETURN
      END
```

Appendix D Two dimensional Poisson solver

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   PROGRAM TO SOLVE 2-D POISSON EQUATION WITH
C     ARBITRARY BOUND. COND. BY A CHEBYSHEV-SPECTRAL
C     METHOD USING A MATRIX MULTIPLY METHOD TO
C     CALCULATE DERIVATIVES
C       BY LLOYD B ELDRED
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      PARAMETER(NMX=21)
      PARAMETER(NMX2=440)
C  NMX2=(NMX+1)**2-1
      PARAMETER(NRITE=1)
C NRITE=1, WRITE EVERYTHING; NRITE=0, NORMAL USAGE
      REAL D1X(0:NMX,0:NMX),D1Y(0:NMX,0:NMX)
```

```
      REAL D2X(0:NMX,0:NMX),D2Y(0:NMX,0:NMX)
      REAL U(0:NMX2),X(0:NMX),Y(0:NMX)
      REAL CA(0:NMX2,0:NMX2),CB(0:NMX2,0:NMX2)
      REAL F1(0:NMX,0:NMX),F(0:NMX2),VBDY(NMX2)
      INTEGER IPVT(0:NMX2),IBDY(NMX2),JBDY(NMX2)
      DIMENSION CBAR(0:NMX)
      PI = 3.1415935898
      DO 4 I = 1,NMX
      CBAR(I) = 1.0
    4 CONTINUE
      CBAR(0) = 2.0
C
C  SET UP COMPUTATIONAL GRID
C
      N = 4
      M = 4
      NM = (N + 1)*(M + 1)-1
      CALL CHEBX(X,N,NMX)
      CALL CHEBX(Y,M,NMX)
      IF(NRITE.EQ.1) WRITE(6,90) (X(I),I = 0,N),(Y(I),I = 0,M)
   90 FORMAT(F14.7)
C
C  FORM DERIVATIVE MATRICES
C
      CBAR(N) = 2.0
      CALL D1(D1X,X,CBAR,N,NMX)
```

```
      IF (NRITE.EQ.1) WRITE(6,91)
   91 FORMAT(' D1X:')
      IF (NRITE.EQ.1) WRITE(6,99) ((D1X(I,J),J=0,N),I=0,N)
      CALL D2(D2X,D1X,N,NMX)
      IF (NRITE.EQ.1) WRITE(6,92)
   92 FORMAT(' D2X:')
      IF (NRITE.EQ.1) WRITE(6,99) ((D2X(I,J),J=0,N),I=0,N)
      CALL CBMAKE(D2X,CB,N,M,NMX,NMX2)
      IF (NRITE.EQ.1) WRITE(6,96)
   96 FORMAT(' CB (MODIFIED D2U/DX2)')
      DO 5 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CB(I,J),J=0,NM)
    5 CONTINUE
      CBAR(N)=1.0
      CBAR(M)=2.0
      CALL D1(D1Y,Y,CBAR,M,NMX)
      IF (NRITE.EQ.1) WRITE(6,93)
   93 FORMAT(' D1Y:')
      IF (NRITE.EQ.1) WRITE(6,99) ((D1Y(I,J),J=0,M),I=0,M)
      CALL D2(D2Y,D1Y,M,NMX)
      IF (NRITE.EQ.1) WRITE(6,94)
   94 FORMAT(' D2Y:')
      IF (NRITE.EQ.1) WRITE(6,99) ((D2Y(I,J),J=0,M),I=0,M)
      CALL CAMAKE(D2Y,CA,N,M,NMX,NMX2)
      IF (NRITE.EQ.1) WRITE(6,97)
   97 FORMAT(' CA (MODIFIED D2U/DY2)')
```

```
      DO 6 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CA(I,J),J=0,NM)
    6 CONTINUE
      DO 7 I=0,NM
      DO 7 J=0,NM
      CA(I,J)=CA(I,J)+CB(I,J)
    7 CONTINUE
      IF (NRITE.EQ.1) WRITE(6,98)
   98 FORMAT(' C BEFORE BOUNDARY CONDITIONS')
      DO 8 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CA(I,J),J=0,NM)
    8 CONTINUE
C
C     GENERATE BOUNDARY CONDITIONS
C
      NBDY=0
      DO 10 I=0,N
      NBDY=NBDY+1
      IBDY(NBDY)=I
      JBDY(NBDY)=0
      VBDY(NBDY)=0.0
      NBDY=NBDY+1
      IBDY(NBDY)=I
      JBDY(NBDY)=M
      VBDY(NBDY)=0.0
   10 CONTINUE
```

```
      MM1 = M-1
      DO 20 J = 1,MM1
      NBDY = NBDY + 1
      IBDY(NBDY) = 0
      JBDY(NBDY) = J
      VBDY(NBDY) = 0.0
      NBDY = NBDY + 1
      IBDY(NBDY) = N
      JBDY(NBDY) = J
      VBDY(NBDY) = 0.0
   20 CONTINUE
C
C  ENFORCE BOUNDARY CONDITIONS
C
      CALL BNDVAL(CA,NBDY,IBDY,JBDY,N,M,NMX2)
C     CALL BCDIRX(C,D1X,NBDDX1,IBDDX1,JBDDX1,N,M,NMX2)
C     CALL BCDIRY(C,D1X,NBDDY1,IBDDY1,JBDDY1,N,M,NMX2)
C     CALL BCDIRX(C,D2X,NBDDX2,IBDDX2,JBDDX2,N,M,NMX2)
C     CALL BCDIRY(C,D2X,NBDDY2,IBDDY2,JBDDY2,N,M,NMX2)
      IF (NRITE.EQ.1) WRITE(6,95)
   95 FORMAT(' C AFTER BOUNDARY CONDITIONS')
      DO 25 I = 0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CA(I,J),J = 0,NM)
   99 FORMAT(12(' ',F6.2))
   25 CONTINUE
C
```

```
C    GENERATE RIGHT HAND SIDE, {F}
C
      DO 30 I=0,N
      XP=X(I)
      TEMP=-2.0*PI*PI*SIN(PI*XP)
      DO 30 J=0,M
      YP=Y(J)
C     F1(J,I)=SIN(PI*YP)*TEMP
      F1(J,I)=-1.0
   30 CONTINUE
      DO 35 K=1,NBDY
      I=IBDY(K)
      J=JBDY(K)
      F1(J,I)=VBDY(K)
   35 CONTINUE
      IF (NRITE.EQ.1) WRITE(6,89)
   89 FORMAT(' F - RIGHT HAND SIDE MATRIX')
      DO 26 J=0,M
      IF (NRITE.EQ.1) WRITE(6,88) (F1(J,I),I=0,N)
   88 FORMAT(5(' ',F14.7))
   26 CONTINUE
      CALL FTRANS(F1,F,N,M,NMX,NMX2)
      IF (NRITE.EQ.1) WRITE(6,87) (F(I),I=0,NM)
   87 FORMAT(' TRANSFORMED F'/,100(F14.7/))
C
```

```
C   SOLVE MATRIX EQUATIONS
C
      NMX2P1 = NMX2 + 1
      NMP1 = NM + 1
      CALL SGEF(CA,NMX2P1,NMP1,IPVT)
      CALL SGES(CA,NMX2P1,NMP1,IPVT,F)
C
C  OUTPUT RESULTS
C
      CALL FUNTRANS(F1,F,N,M,NMX,NMX2)
      WRITE(6,101)
  101 FORMAT('      I    J      APPROX          EXACT',
     $'          X(I)          Y(J)')
      DO 40 I = 0,N
      DO 40 J = 0,M
      UEXACT = SIN(PI*X(I))*SIN(PI*Y(J))
      IF(I.NE.(N/2-1)) GOTO 40
      WRITE(6,100) I,J,F1(I,J),UEXACT,X(I),Y(J)
  100 FORMAT(2X,2I5,2X,F14.7,2X,F14.7,2(2X,F14.7))
   40 CONTINUE
      STOP
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   BNDVAL(C,NBDY,IBDY,JBDY,N,M,MAXN2)
C     SETS UP C MATRIX SO THAT NODE I,J'S
```

```
C     VALUE MAY BE SPECIFIED BY SETTING
C     F(I,J) VALUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE BNDVAL(C,NBDY,IBDY,JBDY,N,M,NMX2)
      DIMENSION C(0:NMX2,0:NMX2),IBDY(NMX2),JBDY(NMX2)
      DO 5 L=1,NBDY
      I=IBDY(L)
      J=JBDY(L)
      IROW=I+(N+1)*J
      NM=(N+1)*(M+1)-1
      DO 10 K=0,NM
      C(IROW,K)=0.0
   10 CONTINUE
      C(IROW,IROW)=1.0
    5 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CAMAKE(A,C,N,M,NM,NMAX)
C     TRANSFORM A MATRIX IN [A][U] TO
C     CA IN [CA]{U}
C  A = INPUT MATRIX, N BY N
C  CA = OUTPUT MATRIX, NM BY NM
```

```
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CAMAKE(A,C,N,M,NMX,NMX2)
      DIMENSION A(0:NMX,0:NMX),C(0:NMX2,0:NMX2)
      DO 5 I=0,NMX2
      DO 5 J=0,NMX2
    5 C(I,J)=0.0
      DO 10 I=0,M
      I2=I*(N+1)
      DO 10 J=0,M
      J2=J*(N+1)
      DO 10 K=0,N
      C(I2+K,J2+K)=A(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CBMAKE(B,C,N,M,NM,NMAX)
C     TRANSFORM B MATRIX IN [U][B] TO
C     CB IN [CB]{U}
C  B = INPUT MATRIX, M BY M
C  CB = OUTPUT MATRIX, NM BY NM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CBMAKE(B,C,N,M,NMX,NMX2)
```

```
      DIMENSION B(0:NMX,0:NMX),C(0:NMX2,0:NMX2)
      DO 5 I=0,NMX2
      DO 5 J=0,NMX2
    5 C(I,J)=0.0
      DO 10 K=0,M
      K2=K*(N+1)
      DO 10 I=0,N
      DO 10 J=0,N
      C(I+K2,J+K2)=B(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CHEBX(X,N,NMX)
C     SETS UP GAUSS-LOBLATTO GRID SYSTEM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CHEBX(X,N,NMX)
      DIMENSION X(0:NMX)
      PI=3.1415935898
      DO 10 I=0,N
      X(I)=COS(PI*FLOAT(I)/FLOAT(N))
   10 CONTINUE
      RETURN
```

```
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE D1 - PUTS FIRST DERIVATIVE
C                 MATRIX IN 'D' MATRIX
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE D1(D,X,C,N,MAXSIZE)
      REAL D(0:MAXSIZE,0:MAXSIZE),X(0:MAXSIZE),C(0:MAXSIZE)
      DO 10 I=0,N
      DO 10 J=0,N
      IF(I.EQ.J) GOTO 10
      D(I,J)=C(I)/C(J)*(-1)**(I+J)/(X(I)-X(J))
   10 CONTINUE
      DO 20 I=1,N-1
      D(I,I)=-X(I)/2./(1.-X(I)*X(I))
   20 CONTINUE
      D(0,0)=(2.*FLOAT(N*N)+1.)/6.0
      D(N,N)=-(2.*FLOAT(N*N)+1.)/6.0
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE D2 - PUTS SECOND DERIVATIVE
C                 MATRIX IN 'D' MATRIX
```

```
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE D2(D,D1,N,MAXSIZE)
      REAL D(0:MAXSIZE,0:MAXSIZE),D1(0:MAXSIZE,0:MAXSIZE)
      MP1=MAXSIZE+1
      NP1=N+1
      CALL MATMUL(D1,MP1,MP1,D1,MP1,MP1,D,MP1,MP1,NP1,NP1,NP1)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  FTRANS(F1,F,N,M,NMX,NMX2)
C     TRANSFORM F1(0:N,0:M) TO F(NM)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FTRANS(F1,F,N,M,NMX,NMX2)
      DIMENSION F1(0:NMX,0:NMX),F(0:NMX2)
      DO 10 I=0,N
      DO 10 J=0,M
      IROW=I+(N+1)*J
      F(IROW)=F1(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
C   FUNTRANS(F1,F,N,M,NMX,NMX2)
C     TRANSFORM F(0:NM) TO F1(0:N,0:M)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FUNTRANS(F1,F,N,M,NMX,NMX2)
      DIMENSION F1(0:NMX,0:NMX),F(0:NMX2)
      DO 10 I=0,N
      DO 10 J=0,M
      IROW=I+(N+1)*J
      F1(I,J)=F(IROW)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   SUBROUTINE MATADD, ADDS UPPER LEFT
C                 PORTIONS OF TWO MATRICES
C       A - INPUT MATRIX OF SIZE NA X MA
C       B - INPUT MATRIX OF SIZE NB X MB
C       C - OUTPUT MATRIX OF SIZE NC X MC
C       N,M - AMOUNTS OF A AND B TO ADD:
C               [A]   [B] =   [C]
C               N X M  N X M   N X M
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATADD(A,NA,MA,B,NB,MB,C,NC,MC,N,M)
```

```
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I = 1,N
      DO 10 J = 1,M
      C(I,J) = A(I,J) + B(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINE MATMUL, MULTIPLIES UPPER LEFT
C                 PORTIONS OF TWO MATRICES
C       A - INPUT MATRIX OF SIZE NA X MA
C       B - INPUT MATRIX OF SIZE NB X MB
C       C - OUTPUT MATRIX OF SIZE NC X MC
C       N,M,NN - AMOUNTS OF A AND B TO MULT:
C                [A]     [B] =   [C]
C               N X NN  NN X M   N X M
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATMUL(A,NA,MA,B,NB,MB,C,NC,MC,N,M,NN)
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I = 1,N
      DO 10 J = 1,M
      C(I,J) = 0.0
      DO 10 K = 1,NN
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
```

```
10 CONTINUE
  RETURN
  END
```

Appendix E Bratu's equation solver

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   PROGRAM TO SOLVE 2-D BRATU EQUATION WITH
C     ARBITRARY BOUND. COND. BY A CHEBYSHEV-SPECTRAL
C     METHOD USING A MATRIX MULTIPLY METHOD TO
C     CALCULATE DERIVATIVES
C       BY LLOYD B ELDRED
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER(NMX=10)
      PARAMETER(NMX2=120)
C  NMX2=(NMX+1)**2-1
      PARAMETER(NRITE=0)
C NRITE=1, WRITE EVERYTHING, NRITE=0 NORMAL USAGE
      PARAMETER(NAUX=2*NMX2+1)
```

```
      REAL*8 D1X(0:NMX,0:NMX),D1Y(0:NMX,0:NMX)
      REAL*8 D2X(0:NMX,0:NMX),D2Y(0:NMX,0:NMX)
      REAL*8 X(0:NMX),Y(0:NMX)
      REAL*8 U(0:NMX,0:NMX),UT(0:NMX,0:NMX)
      REAL*8 U0(0:NMX,0:NMX)
      REAL*8 CA(0:NMX2,0:NMX2),CB(0:NMX2,0:NMX2)
      REAL*8 C(0:NMX2,0:NMX2)
      REAL*8 F1(0:NMX,0:NMX),F(0:NMX2),VBDY(NMX2)
      REAL*8 DUX(0:NMX,0:NMX),DUY(0:NMX,0:NMX)
      REAL*8 DUXT(0:NMX,0:NMX)
      REAL*8 FAC(0:NMX2,0:NMX2),JPVT(0:NMX2)
      REAL*8 AUX(0:NAUX)
      COMPLEX*16 EIGVAL(0:NMX2),EIGVECT(0:NMX2,0:NMX2)
      LOGICAL SELECT(0:NMX2)
      INTEGER IPVT(0:NMX2),IBDY(NMX2),JBDY(NMX2)
      REAL*8 CBAR(0:NMX)
      CHARACTER*80 TITLE
      PI = 3.1415935898
      DO 4 I = 1,NMX
      CBAR(I) = 1.0
    4 CONTINUE
      CBAR(0) = 2.0
C
C  SET UP COMPUTATIONAL GRID
C
      N = 10
```

```
      M = 10
      IF(N.GT.NMX.OR.M.GT.NMX) THEN
        WRITE(6,*) ' N AND/OR M EXCEEDS NMX'
        STOP
      ENDIF
      NM = (N + 1)*(M + 1)-1
      N2 = NMX + 1
      NP1 = N + 1
      MP1 = M + 1
      NMP1 = NM + 1
      NMX2P1 = NMX2 + 1
      NMX22 = 2*NMX2P1
      IROW00 = N/2 + (N + 1)*M/2
      IROW01 = IROW00 + 1
      CALL CHEBX(X,N,NMX)
      CALL CHEBX(Y,M,NMX)
      TITLE = ' X COORDINATES:'
      IF(NRITE.EQ.1) WRITE(6,200) TITLE
      IF(NRITE.EQ.1) WRITE(6,90) (X(I),I = 0,N)
      TITLE = ' Y COORDINATES:'
      IF(NRITE.EQ.1) WRITE(6,200) TITLE
      IF(NRITE.EQ.1) WRITE(6,90) (Y(J),J = 0,M)
   90 FORMAT(F14.7)
  200 FORMAT(A80)
C
```

```
C   FORM DERIVATIVE MATRICES
C
      CBAR(N)=2.0
      CALL D1(D1X,X,CBAR,N,NMX)
      TITLE=' D1X:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,99) ((D1X(I,J),J=0,N),I=0,N)
      CALL D2(D2X,D1X,N,NMX)
      TITLE=' D2X:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,99) ((D2X(I,J),J=0,N),I=0,N)
      CALL CBMAKE(D2X,CB,N,M,NMX,NMX2)
      TITLE= ' CB (MODIFIED D2U/DX2):'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 5 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CB(I,J),J=0,NM)
    5 CONTINUE
   99 FORMAT(10(2X,E10.4))
      CBAR(N)=1.0
      CBAR(M)=2.0
      CALL D1(D1Y,Y,CBAR,M,NMX)
      TITLE =' D1Y:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,99) ((D1Y(I,J),J=0,M),I=0,M)
      CALL D2(D2Y,D1Y,M,NMX)
      TITLE =' D2Y:'
```

```
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,99) ((D2Y(I,J),J=0,M),I=0,M)
      CALL CAMAKE(D2Y,CA,N,M,NMX,NMX2)
      TITLE =' CA (MODIFIED D2U/DY2):'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 6 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (CA(I,J),J=0,NM)
    6 CONTINUE
C
C     GENERATE BOUNDARY CONDITIONS
C
      NBDY=0
      DO 10 I=0,N
      NBDY=NBDY+1
      IBDY(NBDY)=I
      JBDY(NBDY)=0
      VBDY(NBDY)=0.0
      NBDY=NBDY+1
      IBDY(NBDY)=I
      JBDY(NBDY)=M
      VBDY(NBDY)=0.0
   10 CONTINUE
      MM1=M-1
      DO 20 J=1,MM1
      NBDY=NBDY+1
      IBDY(NBDY)=0
```

```
      JBDY(NBDY)=J
      VBDY(NBDY)=0.0
      NBDY=NBDY+1
      IBDY(NBDY)=N
      JBDY(NBDY)=J
      VBDY(NBDY)=0.0
   20 CONTINUE
C
C     SET PARAMETERS LAMDA AND EPSILON AND OTHER INITIAL
C       VALUES
C
      RLAM=0.0
      DLAM=1.0
      DELTA=0.25
      D0=DELTA
      ILOOP=0
      EP=0.20
      DEP=0.0
      DO 29 I=0,N
      DO 29 J=0,M
      U(J,I)=0.0
      F1(J,I)=0.0
   29 CONTINUE
      I=N/2
      J=M/2
```

```
      WRITE(6,100) I,J,EP,RLAM,U(J,I),X(I),Y(J),ITER
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  MAIN PROGRAM LOOP STARTS HERE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   ADVANCE TO NEXT POINT
C
   31 ITER=0
      ITER=ITER+1
C
C  ASSEMBLE RIGHT HAND SIDE
C
      DO 30 I=0,N
      DO 30 J=0,M
      G1=U(J,I)/(1.0+EP*U(J,I))
      IF(G1.GT.200.0) THEN
      WRITE(6,85) RLAM,ITER,I,J,U(J,I)
   85 FORMAT(' OVERFLOW AT LAMDA = ',E14.7,', ITER = ',
     $I5,/,' I = ',I5,' J = ',I5,' U(J,I) = ',E14.7)
      STOP
      ENDIF
      IF(G1.LT.-200.0) THEN
      WRITE(6,86) RLAM,ITER,I,J,U(J,I)
   86 FORMAT(' UNDERFLOW AT LAMDA = ',E14.7,', ITER = ',I5,/,
```

```
      $' I = ',I3,' J = ',I3,' U(J,I) = ',E14.7)
      STOP
      ENDIF
      F1(J,I)=-DLAM*EXP(G1)
   30 CONTINUE
C  PLACE BOUNDARY CONDITION VALUES INTO F1
      DO 35 K=1,NBDY
      I=IBDY(K)
      J=JBDY(K)
      F1(J,I)=VBDY(K)
   35 CONTINUE
      TITLE=' F1:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 33 I=0,N
      IF (NRITE.EQ.1) WRITE(6,99) (F1(J,I),J=0,M)
   33 CONTINUE
C TRANSFORM F1 FROM A MATRIX TO F, A VECTOR
      CALL FTRANS(F1,F,N,M,NMX,NMX2)
      TITLE=' F:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,101) (F(I),I=0,NM)
  101 FORMAT(E14.7)
C
C   ASSEMBLE C
C
   19 DO 36 J=0,NM
```

```
      DO 36 I=0,NM
      C(J,I)=CA(J,I)+CB(J,I)
   36 CONTINUE
      DO 45 J=0,M
      DO 45 I=0,N
      EPU=EP*U(J,I)
      G=EXP(U(J,I)/(1.0+EPU))
      A=RLAM*G*(1.0+EPU/(1.0+EPU)**2)
      IROW=I+(N+1)*J
      C(IROW,IROW)=C(IROW,IROW)+A
   45 CONTINUE
      TITLE =' C BEFORE BOUNDARY CONDITIONS'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 8 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (C(I,J),J=0,NM)
    8 CONTINUE
C     ENFORCE BOUNDARY CONDITIONS
      CALL BNDVAL(C,NBDY,IBDY,JBDY,N,M,NMX2)
      TITLE =' C AFTER BOUNDARY CONDITIONS'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 9 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (C(I,J),J=0,NM)
    9 CONTINUE
C
C     FIND CONDITION NUMBER OF MATRIX
C
```

```
      CALL DLFCRG(NMP1,C,NMX2P1,FAC,NMX2P1,JPVT,RCOND)
      WRITE(6,300) RLAM,RP,ITER,RCOND
      IF(ILOOP.EQ.0) THEN
        RCOND0=RCOND
        ILOOP=1
      ENDIF
C
C  SOLVE MATRIX EQUATIONS
C
   18 IF(INEIG.EQ.0) THEN
        CALL DGEF(C,NMX2P1,NMP1,IPVT)
        CALL DGES(C,NMX2P1,NMP1,IPVT,F)
        TITLE =' SOLUTION DELTA U (VECTOR)'
        IF (NRITE.EQ.1) WRITE(6,200) TITLE
        IF (NRITE.EQ.1) WRITE(6,101) (F(I),I=0,NM)
        CALL FUNTRANS(F1,F,N,M,NMX,NMX2)
        TITLE =' SOLUTION DELTA U (MATRIX)'
        IF (NRITE.EQ.1) WRITE(6,200) TITLE
        DO 11 I=0,N
        IF (NRITE.EQ.1) WRITE(6,99) (F1(J,I),J=0,M)
   11   CONTINUE
      ELSE
        CALL DGEEV(1,C,NMX2P1,EIGVAL,EIGVECT,NMX2P1,0,
     $  NMP1,AUX,NAUX)
        DO 82 J=0,NM
   82   F(J)=EIGVECT(J,IE)
```

```
      CALL FUNTRANS(F1,F,N,M,NMX,NMX2)
      ENDIF
C  REENFORCE BOUNDARY CONDITION VALUES INTO F1
      DO 47 K=1,NBDY
      I=IBDY(K)
      J=JBDY(K)
      F1(J,I)=VBDY(K)
   47 CONTINUE
C
C   SCALE AND UPDATE SOLUTION
C
      SCALE=DELTA/F(IROW00)
      DO 37 J=0,M
      DO 37 I=0,N
      F1(J,I)=SCALE*F1(J,I)
      U(J,I)=U(J,I)+F1(J,I)
   37 CONTINUE
      DLAM=SCALE*DLAM
      RLAM=RLAM+DLAM
      DELTA=D0*RCOND/RCOND0
      TITLE =' SCALED SOLUTION DELTA U'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 12 I=0,N
      IF (NRITE.EQ.1) WRITE(6,99) (F1(J,I),J=0,M)
   12 CONTINUE
      I=N/2
```

```
      J = M/2
      IF(INEIG.EQ.1) THEN
        WRITE(6,100) I,J,EP,RLAM,U(J,I),X(I),Y(J),-1
        GOTO 84
      ENDIF
C
C  ITERATE AT CURRENT POINT
C
   32 ITER = ITER + 1
C
C  ASSEMBLE RIGHT HAND SIDE, F1
C    FIRST CALCULATE D2U/DX2 AND D2U/DY2
C
      DO 51 J = 0,M
      DO 51 I = 0,N
      UT(I,J) = U(J,I)
   51 CONTINUE
      CALL MATMUL(D2X,N2,N2,UT,N2,N2,DUXT,N2,N2,NP1,MP1,NP1)
      DO 52 J = 0,M
      DO 52 I = 0,N
      DUX(J,I) = DUXT(I,J)
   52 CONTINUE
      TITLE = ' DU2/DX2 :'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 110 I = 0,N
      IF (NRITE.EQ.1) WRITE(6,99) (DUX(J,I),J = 0,M)
```

```
110 CONTINUE
    CALL MATMUL(D2Y,N2,N2,U,N2,N2,DUY,N2,N2,MP1,NP1,MP1)
    TITLE = ' DU2/DY2 :'
    IF (NRITE.EQ.1) WRITE(6,200) TITLE
    DO 115 I=0,N
    IF (NRITE.EQ.1) WRITE(6,99) (DUY(J,I),J=0,M)
115 CONTINUE
    DO 38 I=0,N
    DO 38 J=0,M
    G1=U(J,I)/(1.0+EP*U(J,I))
    IF(G1.GT.200.0) THEN
      WRITE(6,85) RLAM,ITER,I,J,U(J,I)
      STOP
    ENDIF
    IF(G1.LT.-200.0) THEN
      WRITE(6,86) RLAM,ITER,I,J,U(J,I)
      STOP
    ENDIF
    F1(J,I)=-(DUX(J,I)+DUY(J,I)+RLAM*EXP(G1))
 38 CONTINUE
C PLACE BOUNDARY VALUES IN F1
    DO 39 K=1,NBDY
    I=IBDY(K)
    J=JBDY(K)
    F1(J,I)=VBDY(K)
 39 CONTINUE
```

```
      TITLE = ' F1:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 55 I=0,N
      IF (NRITE.EQ.1) WRITE(6,99) (F1(J,I),J=0,M)
   55 CONTINUE
C TRANSFORM F1 FROM A MATRIX TO F, A VECTOR
      CALL FTRANS(F1,F,N,M,NMX,NMX2)
      TITLE = ' F:'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      IF (NRITE.EQ.1) WRITE(6,101) (F(I),I=0,NM)
C
C  ASSEMBLE C
C
      DO 40 J=0,NM
      DO 40 I=0,NM
      C(J,I)=CA(J,I)+CB(J,I)
   40 CONTINUE
      DO 46 J=0,M
      DO 46 I=0,N
      EPU=EP*U(J,I)
      G=EXP(U(J,I)/(1.0+EPU))
      A=RLAM*G*(1.0+EPU/(1.0+EPU)**2)
      IROW=I+(N+1)*J
      C(IROW,IROW)=C(IROW,IROW)+A
   46 CONTINUE
      TITLE =' C BEFORE BOUNDARY CONDITIONS'
```

```
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 108 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (C(I,J),J=0,NM)
  108 CONTINUE
C   REARRANGE C
C    REMOVE U(0,0) COLUMN
      DO 41 J=IROW00,NM
      DO 41 I=0,NM
      C(I,J)=C(I,J+1)
   41 CONTINUE
C   PLACE G COLUMN IN LAST COLUMN OF C
      DO 42 J=0,M
      DO 42 I=0,N
      G1=U(J,I)/(1.0+EP*U(J,I))
      G=EXP(G1)
      IROW=I+(N+1)*J
      C(IROW,NM)=G
   42 CONTINUE
      TITLE =' C AFTER REMOVING U(0,0) COL AND ADDING G COL'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 120 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (C(I,J),J=0,NM)
  120 CONTINUE
C   ENFORCE BOUNDARY CONDITIONS
      CALL BNDVAL2(C,NBDY,IBDY,JBDY,N,M,NMX2)
      TITLE =' C AFTER BOUNDARY CONDITIONS'
```

```
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 109 I=0,NM
      IF (NRITE.EQ.1) WRITE(6,99) (C(I,J),J=0,NM)
  109 CONTINUE
C
C  FIND CONDITION NUMBER OF MATRIX
C
      CALL DLFCRG(NMP1,C,NMX2P1,FAC,NMX2P1,JPVT,RCOND)
      WRITE(6,300) RLAM,RP,ITER,RCOND
  300 FORMAT(' RLAM = ',E14.7,' EP = ',E14.7,' ITER = ',I3/
     $,' RCOND = ',E14.7)
C
C  IF MATRIX IS ILL CONDITIONED USE AN EIGENVECTOR APPROACH
C   TO ADVANCE ONE STEP ALONG EACH EIGENVECTOR WITH AN
C   EIG. VAL. OF 1.0
C
      DELTA=D0*RCOND/RCOND0
      IF(DELTA.LT.1.D-5) THEN
        TITLE=' NOW ADVANCING VIA EIGENVECTORS'
        WRITE(6,200) TITLE
        IE=0
        INEIG=1
   81   CONTINUE
        RLAM=RLAMOLD
        DELTA=0.1
        IF(EIGVAL(IE).EQ.(1.0D0,0.0D0)) THEN
```

```
      DO 83 J=0,M
      DO 83 I=0,N
83       U(J,I)=U0(J,I)
      GOTO 19
      ENDIF
84   IE=IE+1
     IF(IE.GT.NM) STOP
     GOTO 81
     ENDIF
C
C  SOLVE MATRIX EQUATIONS
C
28 CALL DGEF(C,NMX2P1,NMP1,IPVT)
   CALL DGES(C,NMX2P1,NMP1,IPVT,F)
   TITLE =' DGES SOLUTION (VECTOR)'
   IF (NRITE.EQ.1) WRITE(6,200) TITLE
   IF (NRITE.EQ.1) WRITE(6,101) (F(I),I=0,NM)
   DLAM=F(NM)
   DO 44 I=NM,IROW01,-1
   F(I)=F(I-1)
44 CONTINUE
   F(IROW00)=0.0
   TITLE =' DELTA U SOLUTION (VECTOR), REARRANGED'
   IF (NRITE.EQ.1) WRITE(6,200) TITLE
   IF (NRITE.EQ.1) WRITE(6,101) (F(I),I=0,NM)
C UNTRANSFORM F, BACK TO MATRIX
```

```
      CALL FUNTRANS(F1,F,N,M,NMX,NMX2)
      TITLE =' SOLUTION DELTA U (MATRIX)'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 111 I=0,N
      IF (NRITE.EQ.1) WRITE(6,99) (F1(J,I),J=0,M)
  111 CONTINUE
C REENFORCE BOUNDARY CONDITION VALUES INTO F1
      DO 49 K=1,NBDY
      I=IBDY(K)
      J=JBDY(K)
      F1(J,I)=VBDY(K)
   49 CONTINUE
C
C   UPDATE SOLUTION
C
      DO 50 J=0,M
      DO 50 I=0,N
      U(J,I)=U(J,I)+F1(J,I)
   50 CONTINUE
      TITLE =' SOLUTION U (MATRIX)'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
      DO 112 I=0,N
      IF (NRITE.EQ.1) WRITE(6,99) (U(J,I),J=0,M)
  112 CONTINUE
      TITLE = ' DELTA LAMBDA    NEW LAMBDA'
      IF (NRITE.EQ.1) WRITE(6,200) TITLE
```

```
      RLAM = RLAM + DLAM
      DELTA = D0*RCOND/RCOND0
      IF (NRITE.EQ.1) WRITE(6,125) DLAM,RLAM
  125 FORMAT(2X,E14.7,2X,E14.7)
C
C  CHECK FOR CONVERGENCE
C
C     ERROR = 0.0
C     DO 60 I = 0,N
C     DO 60 J = 0,M
C     IF(ABS(U(J,I)).GT.1.0E-20) THEN
C      ERROR = ERROR + (F1(J,I)/U(J,I))**2
C     ELSE
C      ERROR = ERROR + F1(J,I)**2
C     ENDIF
C  60 CONTINUE
      ERROR = DLAM/RLAM
      IF(ERROR.GT.0.0005.AND.ITER.LT.50) GOTO 32
C
C  OUTPUT CONVERGED RESULTS
C
C     WRITE(6,106)
      DO 70 I = 0,N
      DO 70 J = 0,M
C     I = N/2
C     J = M/2
```

```
      WRITE(6,100) I,J,EP,RLAM,U(J,I),X(I),Y(J),ITER
  100 FORMAT(2I5,5(2X,F8.3),I5)
  106 FORMAT('  I    J     EP    RLAM     U(J,I)    X(I)   Y(J)',
     $'    ITER')
C 106 FORMAT('    I    J    U(J,I)       X(I)       Y(J)        EP',
C    $'        LAM')
   70 CONTINUE
   71 CONTINUE
C     STOP
      IF(U(J,I).GT.10.0) STOP
      ITER=0
      DLAM=1.0
      IF(INEIG.EQ.1) THEN
        IE=IE+1
        IF(IE.GT.NM) STOP
        GOTO 81
        ENDIF
C  SAVE {U0}
      DO 80 J=0,M
      DO 80 I=0,N
   80   U0(J,I)=U(J,I)
      RLAMOLD=RLAM
      GOTO 31
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
C   BNDVAL(C,NBDY,IBDY,JBDY,N,M,MAXN2)
C    SETS UP C MATRIX SO THAT NODE I,J'S
C     VALUE MAY BE SPECIFIED BY SETTING
C     F(I,J) VALUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE BNDVAL(C,NBDY,IBDY,JBDY,N,M,NMX2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION C(0:NMX2,0:NMX2),IBDY(NMX2),JBDY(NMX2)
      NM=(N+1)*(M+1)-1
      DO 5 L=1,NBDY
      I=IBDY(L)
      J=JBDY(L)
      IROW=I+(N+1)*J
      DO 10 K=0,NM
      C(IROW,K)=0.0
   10 CONTINUE
      C(IROW,IROW)=1.0
    5 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   BNDVAL2(C,NBDY,IBDY,JBDY,N,M,MAXN2)
C    SETS UP C MATRIX SO THAT NODE I,J'S
C     VALUE MAY BE SPECIFIED BY SETTING
```

```
C     F(I,J) VALUE
C   *** THIS ROUTINE IS FOR THE REARRANGED C***
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE BNDVAL2(C,NBDY,IBDY,JBDY,N,M,NMX2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION C(0:NMX2,0:NMX2),IBDY(NMX2),JBDY(NMX2)
      NM=(N+1)*(M+1)-1
      IROW00=N/2+(N+1)*M/2
      DO 5 L=1,NBDY
      I=IBDY(L)
      J=JBDY(L)
      IROW=I+(N+1)*J
      DO 10 K=0,NM
      C(IROW,K)=0.0
   10 CONTINUE
      IF(IROW.LE.IROW00) C(IROW,IROW)=1.0
      IF(IROW.GT.IROW00) C(IROW,IROW-1)=1.0
    5 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   CAMAKE(A,C,N,M,NM,NMAX)
C     TRANSFORM A MATRIX IN [A][U] TO
C     CA IN [CA]{U}
```

```
C   A = INPUT MATRIX, N BY N
C   CA = OUTPUT MATRIX, NM BY NM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CAMAKE(A,C,N,M,NMX,NMX2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(0:NMX,0:NMX),C(0:NMX2,0:NMX2)
      DO 5 I=0,NMX2
      DO 5 J=0,NMX2
    5 C(I,J)=0.0
      DO 10 I=0,M
      I2=I*(N+1)
      DO 10 J=0,M
      J2=J*(N+1)
      DO 10 K=0,N
      C(I2+K,J2+K)=A(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   CBMAKE(B,C,N,M,NM,NMAX)
C     TRANSFORM B MATRIX IN [U][B] TO
C     CB IN [CB]{U}
C   B = INPUT MATRIX, M BY M
C   CB = OUTPUT MATRIX, NM BY NM
```

```
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CBMAKE(B,C,N,M,NMX,NMX2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION B(0:NMX,0:NMX),C(0:NMX2,0:NMX2)
      DO 5 I=0,NMX2
      DO 5 J=0,NMX2
    5 C(I,J)=0.0
      DO 10 K=0,M
      K2=K*(N+1)
      DO 10 I=0,N
      DO 10 J=0,N
      C(I+K2,J+K2)=B(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CHEBX(X,N,NMX)
C     SETS UP GAUSS-LOBLATTO GRID SYSTEM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE CHEBX(X,N,NMX)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(0:NMX)
      PI=3.1415935898
```

```
      DO 10 I=0,N
      X(I)=COS(PI*FLOAT(I)/FLOAT(N))
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SUBROUTINE D1 - PUTS FIRST DERIVATIVE
C                     MATRIX IN 'D' MATRIX
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE D1(D,X,C,N,MAXSIZE)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      REAL*8 D(0:MAXSIZE,0:MAXSIZE),X(0:MAXSIZE),C(0:MAXSIZE)
      DO 10 I=0,N
      DO 10 J=0,N
      IF(I.EQ.J) GOTO 10
      D(I,J)=C(I)/C(J)*(-1)**(I+J)/(X(I)-X(J))
   10 CONTINUE
      DO 20 I=1,N-1
      D(I,I)=-X(I)/2./(1.-X(I)*X(I))
   20 CONTINUE
      D(0,0)=(2.*FLOAT(N*N)+1.)/6.0
      D(N,N)=-(2.*FLOAT(N*N)+1.)/6.0
      RETURN
```

```
    END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   SUBROUTINE D2 - PUTS SECOND DERIVATIVE
C                 MATRIX IN 'D' MATRIX
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    SUBROUTINE D2(D,D1,N,MAXSIZE)
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    REAL*8 D(0:MAXSIZE,0:MAXSIZE),D1(0:MAXSIZE,0:MAXSIZE)
    MP1=MAXSIZE+1
    NP1=N+1
    CALL MATMUL(D1,MP1,MP1,D1,MP1,MP1,D,MP1,MP1,NP1,NP1,NP1)
    RETURN
    END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  FTRANS(F1,F,N,M,NMX,NMX2)
C    TRANSFORM F1(0:N,0:M) TO F(NM)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    SUBROUTINE FTRANS(F1,F,N,M,NMX,NMX2)
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    DIMENSION F1(0:NMX,0:NMX),F(0:NMX2)
    DO 10 I=0,N
    DO 10 J=0,M
```

```
      IROW=I+(N+1)*J
      F(IROW)=F1(J,I)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  FUNTRANS(F1,F,N,M,NMX,NMX2)
C     TRANSFORM F(0:NM) TO F1(0:N,0:M)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FUNTRANS(F1,F,N,M,NMX,NMX2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION F1(0:NMX,0:NMX),F(0:NMX2)
      DO 10 I=0,N
      DO 10 J=0,M
      IROW=I+(N+1)*J
      F1(J,I)=F(IROW)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINE MATADD, ADDS UPPER LEFT
C                PORTIONS OF TWO MATRICES
C      A - INPUT MATRIX OF SIZE NA X MA
```

```
C       B - INPUT MATRIX OF SIZE NB X MB
C       C - OUTPUT MATRIX OF SIZE NC X MC
C       N,M - AMOUNTS OF A AND B TO ADD:
C               [A]   [B] =   [C]
C              N X M  N X M   N X M
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATADD(A,NA,MA,B,NB,MB,C,NC,MC,N,M)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I = 1,N
      DO 10 J = 1,M
      C(I,J) = A(I,J) + B(I,J)
   10 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUBROUTINE MATMUL, MULTIPLIES UPPER LEFT
C                PORTIONS OF TWO MATRICES
C       A - INPUT MATRIX OF SIZE NA X MA
C       B - INPUT MATRIX OF SIZE NB X MB
C       C - OUTPUT MATRIX OF SIZE NC X MC
C       N,M,NN - AMOUNTS OF A AND B TO MULT:
C               [A]     [B] =   [C]
C              N X NN  NN X M   N X M
```

```
C
C234567CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MATMUL(A,NA,MA,B,NB,MB,C,NC,MC,N,M,NN)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NA,MA),B(NB,MB),C(NC,MC)
      DO 10 I=1,N
      DO 10 J=1,M
      C(I,J)=0.0
      DO 10 K=1,NN
      C(I,J)=C(I,J)+A(I,K)*B(K,J)
   10 CONTINUE
      RETURN
      END
```

References

(1) Gottlieb, D., Orszag, S. A., Numerical Analysis of Spectral Methods: Theory and Applications, SIAM-CBMS, Philadelphia, 1977

(2) Peyret, R., "Introduction to Spectral Methods with Application to Fluid Mechanics", Computational Fluid Dynamics, March 3-7, 1986, von Karman Institute for Fluid Dynamics, Lecture Series 1986-04

(3) Haidvogel, D.B., Zang, T., "The Accurate Solution of Poisson's Equation by Expansion in Chebyshev Polynomials", Journal of Computational Physics, 30, 167-180, 1979

(4) Haldenwang, P. Labrosse, G., Abboudi, S., Deville, M., "Chebyshev 3-D Spectral and 2-D Pseudospectral Solvers for the Helmholtz Equation", Journal of Computational Physics, 55, 115-290, 1984

(5) Streett, C.L., Hussaini, M.Y., "A Numerical Simulation of Finite-Length Taylor-Couette Flow", AIAA Paper No. 87-1444, AIAA 19th Fluid Dynamics, Plasma, Dynamics and Lasers Conference, June 8-10, 1987.

(6) Ku, H.C., Hirsh, R.S., Taylor, T.D., "A Pseudospectral Method for Solution of the Three-Dimensional Incompressible Navier-Stokes Equations"

(7) Haldenwang, P., Labrosse, G., "2-D and 3-D Spectral Chebyshev Solutions for Free Convection at High Rayleigh Numbers", Sixth International Symposium on Finite Elements in Flow Problems, June 16-20, 1986.

(8) Macaraeg, M.G., Streett, C.L., "Improvements in Spectral Collocation Discretization through a Multiple Domain Technique", Applied Numerical Matematice, Vol 2, No 2, 95-108, April 1986.

(9) Macaraeg, M. G., Streett C.L.,"A Spectral Multi-Domain Technique with Application to Generalized Curvalinear Coordinates", March 1986, NASA TM-87701

(10) Boyd, J.P., "An Analytical and Numerical Study of the Two- Dimensional Bratu Equation", Journal of Scientific Computing, Vol 1, No 2 , 1986.

(11) Kapania, R.K., "A Pseudo-Spectral Study of a 2-Parameter Bratu's Equation", presented at ICES 488, Atlanta, Apr. 1988, To appear in Computational Mechanics; an International Journal

(12) Seydel, R., "From Equilibrium to Chaos, Practical Bifurcation and Stability Analysis",Elsevier, New York, 1988

The vita has been removed from the scanned document