APPM 4660 Project: Spectral Methods for 2D Boundary Value Problems

Derrick Choi[1], Ryan Stewart[21]
Date: May 3, 2022

Many problems in physics and engineering such as heat conduction through a rod, ideal fluid flow over a body, and deformation of a body due to applied forces all can be described by a particular boundary value problem (BVP). Generally, these BVPs are solved using some numerical method. This project will specifically explore how spectral methods are used to solve BVPs. The project first briefly reviews spectral approximations and the formulation for solving one-dimensional BVPs with the Chebyshev spectral collocation method. Then, the project examines the implementation of the Chebyshev spectral collocation method to solve two-dimensional (2D) BVPs. The project is further extended by exploring time dependent problems, also known as initial boundary value problems (IBVPs), which require discretization in both space and time. Furthermore, the convergence of the spectral collocation methods are explored for both 2D BVPs and IBVPs. Lastly, numerical experiments are conducted for approximating the solution to the 2D Heat Equation and the 2D Helmholtz equation. Results from the numerical experiments demonstrate the spectral convergence (when expected) for the Chebyshev spectral collocation method. The project concludes with other possible extensions of the project to explore in the future.

I. Introduction

There are many numerical methods to solving boundary value problems (BVPs), which are described by a set of partial or ordinary differential equations and a set of boundary conditions (BCs). Of these numerical methods, the three main techniques for solving BVPs are finite differences, finite elements, and spectral methods. Each of these methods have their own advantages and disadvantages: finite differences yield a sparse linear system of equations but may need to include additional points outside of the domain to maintain the same order of approximation when enforcing non-Dirichlet boundary conditions; finite element methods also yield a sparse linear system of equations, require less smoothness and continuity restrictions on the solution to the BVP, can be easily used on complex domains, but requires a rederivation of a weak form of the BVP if the boundary conditions change; and spectral methods converge at a much faster rate (exponentially) than finite difference (FD) and finite element methods (FEM), but spectral methods require solving a dense linear system of equations[1]. Because of this interesting fact that spectral methods can have exponential convergence, this project will explore solving BVPs with the Chebyshev spectral collocation method in two dimensions (2D).

[1]ID:109211641
[2]ID:109225153

II. Spectral Approximations

Since solving BVPs accurately requires being able to approximate a derivative to a high-degree of accuracy, a numerical method is needed for approximating derivatives. Usually, this involves using the values of a function u at set of discrete points, $\{x_j\}_{j=0}^N$, denoted by $\{u(x_j)\}_{j=0}^N$. One could attempt to use a Finite Difference Method. While this is a suitable method for approximations, finite difference methods are limited in that they compute their interpolating functions locally. The conceptual idea behind spectral methods are to work in the limit of higher order finite difference methods, essentially working with a differentiation formula of infinite order and infinite bandwidth [2]. However, working with matrices that are infinite in size is unrealistic and so we work with a finite grid of mesh points or interpolation nodes, $\{x_j\}_{j=0}^N$, which yield matrices of finite size.

A. Approximations with Chebyshev Polynomials

One typical basis of functions to approximate a function u are polynomials. As such, we wish to pick to approximate our solution to a BVP ($u(x)$) as a sum of Lagrange polynomials (Eq. 1, , where c_j is an unknown coefficient $\mathscr{L}_j(x)$ denotes a Lagrange polynomial at a discrete point x_j.

$$u_N(x) = \sum_{j=0}^{N} c_j \mathscr{L}_j(x) \tag{1}$$

Then, if a set of values of $u(x)$ at the set of discrete points $\{x_j\}_{j=0}^N$ is given, a unique interpolating polynomial $p(x)$ of degree $\leq N$ can be formed such that Eq. 2 is satisfied.

$$p(x_j) = u(x_j) = v_j \quad \text{for} \quad 0 \leq j \leq N \tag{2}$$

Furthermore, the derivative of the function u at the discrete points x_j can be simply calculated by first taking the derivative of $p(x)$ and then evaluating at each point. By noting that Eq. 2 yields a set of $N+1$ equations and that evaluating $p'(x)$ still yields $N+1$ equations, we can express the derivative at the discrete points as product of $(N+1) \times (N+1)$ matrix (D_N) and a $(N+1) \times 1$ vector. Namely, setting a vector $\vec{w}$ with entries $w_j = p'(x_j)$, we get Eq. 3, which gives the derivative of the interpolating polynomial at the interpolation nodes.

$$\vec{w} = D_N \vec{v}, \quad \text{where} \quad D_N \in \mathbb{R}^{(N+1)\times(N+1)} \tag{3}$$

In relation to solving a BVP, spectral collocation requires that the solution satisfies the BVP at the interpolation nodes. As such, the unknown coefficients c_j in Eq. 1 are the values of u at x_j, which also means that $\vec{v}$ in Eq. 3 is the vector of unknowns to be determined. Additionally, since the Runge phenomenon can occur with using non-optimal interpolation node locations and can hinder the convergence rate of the spectral method [2], we use a popular choice of nodes for non-periodic functions defined on $[-1, 1]$ known as the Chebyshev nodes, given by Eq. 4

$$x_j = \cos\left(\frac{j\pi}{N}\right) \quad \text{for } j = 0, 1, \cdots, N \tag{4}$$

Since the Chebyshev nodes are used and polynomial interpolation is unique, our interpolating polynomials will be the Chebyshev polynomials. Additionally, using the Chebyshev nodes as our

interpolation nodes will result in the following Chebyshev spectral differentiation matrix D_N, which has the following entries as derived in [2]. While Eqs. 5-7 only give the matrix for the first derivative, the spectral convergence allows the pth-derivative to be approximated by raising D_N to the pth power.

$$(D_N)_{1,1} = \frac{2N^2+1}{6} \qquad (D_N)_{(N+1),(N+1)} = \frac{-2N^2+1}{6} \tag{5}$$

$$(D_N)_{j,j} = \frac{-x_j}{2(1-x_j^2)} \quad j = 1, 2, \cdots, N \qquad (D_N)_{i,j} = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(x_i - x_j} \quad i \neq j, \quad i, j = 1, \cdots, N+1 \tag{6}$$

$$c_i = \begin{cases} 2, & i = 1 \text{ or } i = N+1 \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

III. 1-D Boundary Value Problem Formulation

Knowing how to approximate derivatives using the Chebyshev spectral differentiation matrix, we now layout a pseudocode in Algorithm 1 for solving a linear, second-order, variable coefficient BVP in one dimension with Dirichlet or Neumann boundary conditions. In this algorithm, it worth noting that we discretize the derivative operators in the differential equation using the derivative matrix given by Eqs. 5-7, and the boundary conditions are enforced by replacing the first and last row equations with the corresponding rows of the identity matrix (for a Dirichlet boundary condition) or the first derivative matrix (for a Neumann boundary condition). It is worth noting that enforcing the boundary conditions in 1D BVPs amounts to replacing the equations for the first and last endpoints in the interval, and the same idea will extend to 2D BVPs as seen in Section IV.C.

Algorithm 1: Solving Boundary Value Problem with Chebyshev spectral collocation

Solving the BVP of the form: $y'' + p(x)y' + q(x)y = f(x)$ **Begin**

Input :N, Boundary Conditions

Output x, y

:

$x \leftarrow x_j = \cos(\frac{j\pi}{N}), \quad j = 0, 1, \ldots, N$

$D_1 \leftarrow$ Approximation of first derivative operator (Eqs. 5-7) at x_j

$(D_2) \leftarrow (D_N)^2$ Approximation of second derivative at x_j

$A \leftarrow D_2 + \text{diag}(p(x_j))D_1 + \text{diag}(q(x_j))$ ▷ diag denotes a diagonal matrix with entries given by variable coefficent at x_j

$f \leftarrow f(x_j)$, Evaluate forcing term at x_j

```
    if Dirichlet Boundary Condition then
        Row of A corresponding to equation for a boundary node gets replaced by
         corresponding row of (N + 1) × (N + 1) identity matrix
    else
        if Neumann Boundary Condition then
            Row of A corresponding to equation for a boundary node gets replaced by
             corresponding row of first derivative matrix,D_1
        end
    end
    Replace first and last entries of f with Boundary condition values at corresponding
     endpoints
    y ← A^{-1} f, Solve linear system
end
```

IV. 2-D Boundary Value Problems

In this section, we will discuss the formulation behind the Chebyshev spectral collocation method to solve 2D BVPs along with examining the stability and convergence of spectral methods. In addition to this, we will also include an algorithm along with examples and experimental results.

A. Approximating the Differential Equation

Now, lets begin our extension into two spatial dimensions by taking a look at the Poisson Equation in 2-D. For simplicity, assume both of the functions $u(x, y)$ and $f(x, y)$ are both smooth continuous analytic functions. Let us also define our domain R: $R \equiv \{(x, y)|(x, y) \in [-1, 1] \times [-1, 1]\}$, for the continuation of this report as it will simplify our notation greatly.

$$\begin{cases} \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x, y) \quad \forall (x, y) \in R \\ u(\pm 1, y) = 0 = u(x, \pm 1) \end{cases} \tag{8}$$

Similar to the 1D case, we look for the approximate solutions to the functions $u(x, y)$ of the following form

$$u_N(x, y) = \sum_{j=0}^{N_x} \sum_{k=0}^{N_y} \hat{u}_N(x_j, y_k)\phi_{j,k}(x, y) \tag{9}$$

where $\hat{u}_N(x_j, y_k)$ are coefficients that contain our interpolation nodes, $\{x_j\}_{j=0}^{N_x}$, $\{y_k\}_{k=0}^{N_y}$, which we will choose to be the Chebyshev nodes along their respective directions, $N_x = N = N_y$ and $\phi_{j,k}(x, y)$ are the basis functions. Note that we will also choose to separate our basis functions because we have chosen to construct our interpolation grid by independently discretizing in the x- and y- directions, our Chebyshev nodes [REF Trefethen CH. 7]. This form of an interpolation grid is also know as a tensor product grid. In addition, we will also choose our basis functions to be the Chebyshev polynomials, allowing us to now look for a solution of the familiar form

$$u_N(x,y) = \sum_{j=0}^{N_x}\sum_{k=0}^{N_y} \hat{u}_N(x_j, y_k) T_j(x) T_k(y) \tag{10}$$

Due to this separation, it is trivial to see that if one were to take two partial derivatives both with respect to (w.r.t.) x (or y) then the basis functions associated with y (or x) are unaffected. Taking note of this, if we compress our vector $\vec{u}$ into a column vector then, with the help from the Kronecker Product, see appendix, we are able to represent our partial derivatives as

$$\frac{\partial^2}{\partial x^2} \Rightarrow D_{N_x}^2 \otimes I_{N_y} \tag{11}$$

$$\frac{\partial^2}{\partial y^2} \Rightarrow I_{N_x} \otimes D_{N_y}^2 \tag{12}$$

where D_i is the Chebyshev differentiation matrix that we had shown above (Eqs. 5-7) and I_i are identity matrices, both for $i = N_x, N_y$. This allows us to rewrite our differential equation (Eq. 8) as

$$(D_{N_x}^2 \otimes I_{N_y} + I_{N_x} \otimes D_{N_y}^2)\vec{u} = \vec{f} \tag{13}$$

where $\vec{f}$ is the discretized description of the function $f(x,y)$ from our original differential equation, obtained by evaluating $f(x,y)$ at each point on the tensor product grid. We now must incorporate our boundary conditions into 13. Fortunately, to implement our homogeneous Dirichlet boundary conditions we can simply replace the appropriate rows and/or columns of our differentiation matrix with the rows of an $(N+1)\times(N+1)$ identity matrix corresponding to the nodes on the boundary and set the corresponding right hand sides to zero. A more extensive description on how to apply boundary conditions is given in Section IV.C.

B. Spectral Convergence

This section will cover the convergence of the 2D Chebyshev spectral collocation method. It is important to first note that when checking for convergence of a spectral method in two spatial dimensions that both of the spatial discretizations must converge for the entire method to converge. This is a direct result of the fact that we chose to independently discretize our two spatial dimensions. For this reason, we will only cover the convergence of spectral methods for the one spatial dimension situation while keeping in mind that the conditions must be satisfied for both the x- and y- discretizations. In addition, we will also limit our analysis to only considering functions that are smooth and can be extended into the complex plane for which they are analytic in some open neighborhood of [-1,1]. The convergence depends heavily on the accuracy of our Chebyshev spectral differentiation.

Theorem IV.1 *Suppose that our analytic function of interest is u and that w is the νth Chebyshev spectral derivative of u ($\nu \geq 1$), then*

$$|w_j - u^{(\nu)}(x_j)| = O(K^{-N}) \tag{14}$$

as $N \to \infty$ and where K is a constant.

It is important to note that the constant, K, above is independent of v and is derived with the help of some complex analysis [2]. This is what is known as exponential convergence. In fact, Theorem IV.1 can best be seen in an example. For this example, we use the Chebyshev polynomials to approximate the first and second derivatives of a smooth, analytic, and separable function (Eq. 15); a smooth but non-analytic function (Eq. 16); and a smooth, analytic, but non-separable function in x and y (Eq. 17) on the square $[-1, 1] \times [-1, 1]$. Figure 1 shows the error in the L_2-norm for each of the derivatives.

$$f(x, y) = e^x \sin y \tag{15}$$

$$g(x, y) = e^{\frac{-1}{\sin^2(x/2+y)}} \tag{16}$$

$$h(x, y) = sin(xy + 1) \tag{17}$$

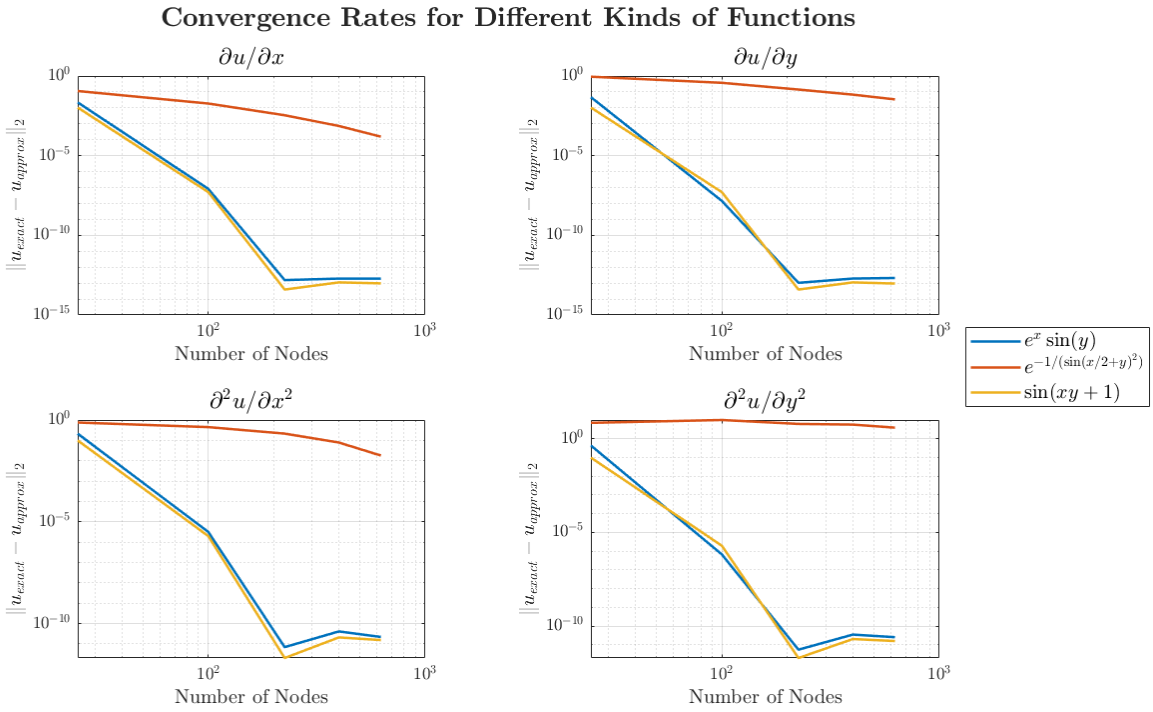

Fig. 1 L_2-error in spectral approximations of first and second derivatives different kinds of functions

As seen in Figure 1, for the smooth but non-analytic function, we lose the spectral convergence rate. On the other hand, we see that the convergence rate for Eq. 15 is faster than any fixed power of N and that the flat part of the each of the plots in Figure1 is due to errors in machine precision as the number of nodes used increases, as expected from Theorem IV.1. Additionally, we also see that our approximation also works well for the analytic but non-separable function, even though it was initially assumed that we are using a separable basis for our approximation (Eq. 10. In relation to solving BVPs, the spectral collocation method will be using the Chebyshev basis (Eq. 10) to

approximate the solution to the governing partial differential equation and the convergence rate of the method (as seen in Figure 1) will depend on the analyticity of the solution.

C. Algorithm

As subtly noted earlier, Eq. 13 enforces that the solution u satisfies the differential equation at each of the grid points x_j, Eq. 13. However, it does not enforce the boundary conditions in the BVP for the grid points that lie on the boundary. In order to account for the boundary conditions in the BVP, either a different basis of functions that satisfy the boundary conditions can be used or equations in the original system of equations can be replaced with ones that enforce the boundary condition [2]. For simplicity, we will consider replacing equations in the original system with ones that satisfy the boundary conditions.

For a systematic way of enforcing the boundary conditions, we first identify the nodes on the tensor product grid that lie on the boundary of the domain and the boundary nodes' corresponding row equations. An example of this for a square domain for $(x, y) \in [-1, 1] \times [-1, 1]$, is seen in Fig. 2.

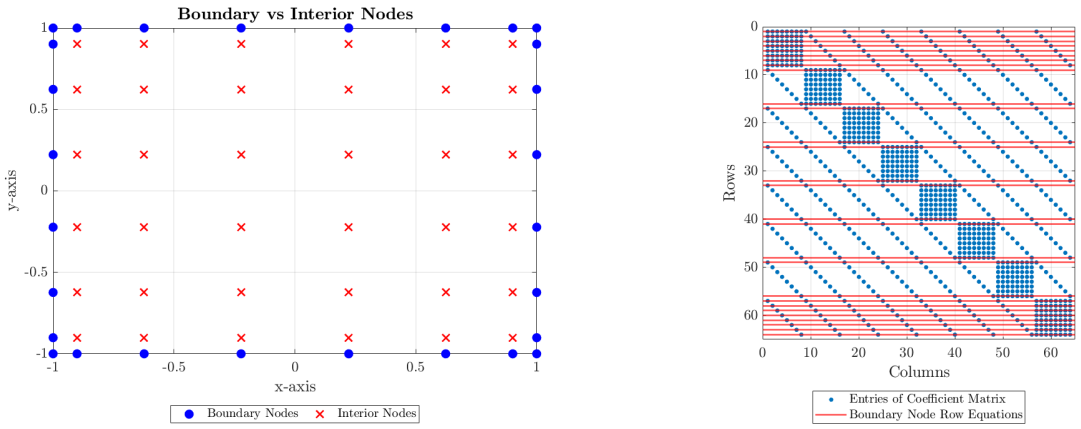

Fig. 2 Separating nodes on the boundary from interior nodes (left), Corresponding row equations to be replaced (right)

Furthermore, in the case where each side of the square domain has a different boundary condition, the nodes on the boundary can be further distinguished from each other by grouping them by the sides of the square domain (i.e bottom, right, top, and left sides of the square) and ordering the nodes in a counter-clockwise manner. We show this separation in Figure 3 for the same square domain example in Figure 2. It is worth noting that an inevitable issue with the corner points of the square being double counted. For simplicity, we avoid this issue by assuming that the corner point belongs to one point only (as seen in Figure 3) and enforce the appropriate boundary condition for the corner node. However, it should also be mentioned that the solutions at the sharp corners of a domain (such as the corner nodes in the square domain) are likely to be singular and can slow the convergence rate to algebraic convergence (convergence rate has a fixed power of N) [1]. Boyd also other methods for dealing with such singularities such as coordinate transformations [1].

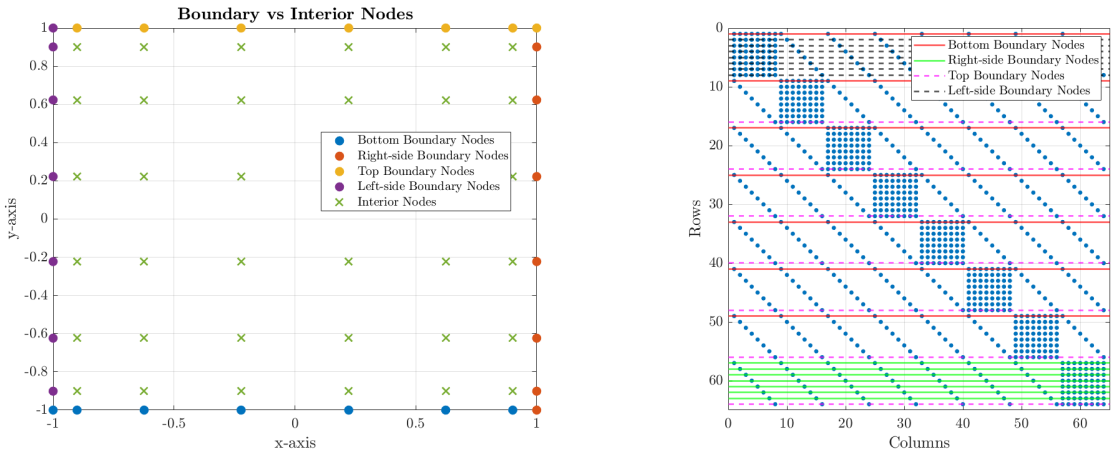

Fig. 3 Distinguishing nodes on the boundary by sides of the square domain (left), Corresponding row equations (right)

We then sort the system of equations (as in Eq. 13) such that the equations are grouped into ones that are for the nodes on the boundary and ones that solve the differential equation in the interior of the boundary. Supposing that $\mathbf{A}$ is the coefficient matrix representing the discretized versions of the operators in the differential equation (such as the $(D_X^2 \otimes I_Y + I_X \otimes D_Y^2)$ in Eq. 13) and $\vec{u}$ is the vector of unknowns in linear system of equations, we can be think of reordering of the equations as solving the linear system described by Eq. 18, where the subscript ext and int denotes the boundary node and interior node parts.

$$\left[\begin{array}{c} \mathbf{A}_{ext} \\ \hline \mathbf{A}_{int} \end{array}\right] \left[\begin{array}{c} \vec{u}_{ext} \\ \hline \vec{u}_{int} \end{array}\right] = \left[\begin{array}{c} \vec{f}_{ext} \\ \hline \vec{f}_{int} \end{array}\right] \tag{18}$$

Then, to enforce boundary conditions for the left hand side, the entries of the matrix $\mathbf{A}_{ext}$ and vector $\vec{f}_{ext}$ is replaced by appropriate values depending on the type of boundary condition. In the case of a Dirichlet boundary condition, the rows of $\mathbf{A}_{ext}$ are replaced by rows of the identity matrix corresponding to the boundary nodes. For a Neumann type boundary condition, $\mathbf{A}_{ext}$ is replaced by corresponding rows of the differentiation matrix representing a partial derivative along a given direction (e.g. $D_X \otimes I_Y$ for $\partial u/\partial x$ and $I_X \otimes D_Y$ for $\partial u/\partial y$) for the boundary nodes. As Robin boundary conditions are linear combinations of Dirichlet and Neumann boundary conditions, these types of boundary conditions are simply implemented by also doing a linear combination of the corresponding rows of the identity matrix and partial differentiation matrix for the nodes on the boundary. A visual representation of reordering the linear system of equations into $\mathbf{A}_{ext}$ concatenated with $\mathbf{A}_{int}$ and assigning Robin boundary conditions involving a partial derivative in x for the top and bottom sides of the square domain and robin boundary conditions involving a partial derivative in y for the left and right sides of the square domain is given in Figure 4.

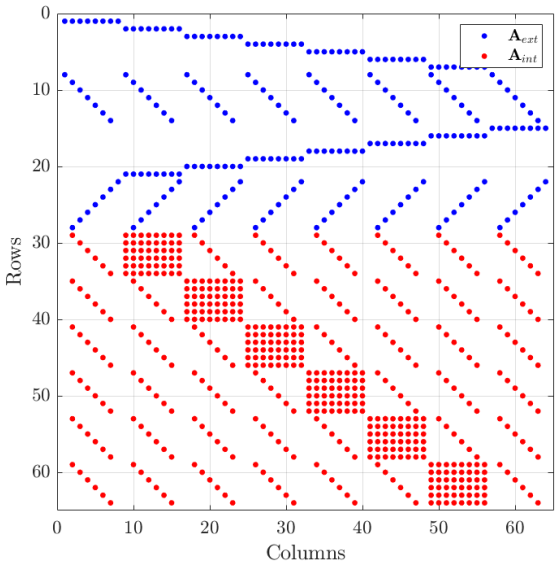

Fig. 4 Reordered coefficient matrix for linear system to solve a BVP with Robin boundary conditions

It is worth noting that many other kinds of boundary conditions exist (periodic boundary conditions, symmetry boundary conditions, etc). For the case of periodic boundary conditions, it is common to use the Fourier basis insteady of the Chebyshev polynomials as the Fourier basis is already periodic [1]. However, generally, implementing these other boundary conditions follow the same kind of process of distinguishing the boundary nodes from the rest of the nodes and applying the appropriate boundary condtion. Lastly, for the hand side of the linear system, the entries of $\vec{f}_{ext}$ are replaced by values that the boundary condition take on at the boundary nodes.

Now, knowing how to apply boundary conditions for a 2D BVP and approximate the differential equation, we summarize the above with a basic algorithm (Algorithm 1) for using Chebyshev spectral collocation to solve a 2D BVP with Dirichlet or Neumann boundary conditions on a square defined by $[-1,1] \times [-1,1]$.

Algorithm 2: Solving 2D Boundary Value Problem with Chebyshev spectral collocation

Solving the BVP of the form: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$

Begin

Input :N, Boundary Conditions

Output x, y, u

:

$x \leftarrow x_j = \cos(\frac{j\pi}{N}), \quad j = 0, 1, \ldots, N$

$x \leftarrow \frac{(b-a)}{2}x + \frac{b+a}{2} \leftarrow$ Linear map from $[-1,1]$ to $[a,b]$

$y \leftarrow x$

$xx \leftarrow$ vector x-coordinates in tensor product grid

$yy \leftarrow$ vector y-coordinates in tensor product grid

8

9 $Ext_{ID} \leftarrow$ indices of xx (or yy) for the points that lie on the boundary

10 $Int_{ID} \leftarrow$ indices of xx (or yy) for the points that lie inside the boundary

11 $D_1 \leftarrow D_N$ Approximation of first derivative operator in one dimension (Eqs. 5-7) for each $x_j \in [-1, 1]$

12 $D_1 \leftarrow \frac{2}{b-a} D_1$ First derivative operator on $[a, b]$

13 $(D_2) \leftarrow (D_N)^2$ Approximation of second derivative in one dimension at each $x_j \in [a, b]$

14 $D_{yy} \leftarrow I_N \otimes D_2$ Approximation of second partial derivative in the y direction

15 $D_{xx} \leftarrow D_2 \otimes I_N$ Approximation of second partial derivative in the x direction

16 $A \leftarrow D_{xx} + D_{yy}$, coefficient matrix for left hand side of linear system

17 $f_{int} \leftarrow f(xx_i, yy_j)$, Evaluate forcing term for each (x_i, y_j) lying interior to the boundary
$f_{ext} \leftarrow$ value of boundary condition at exterior points

18 $A_{int} \leftarrow A(INt_ID)$, grab all rows of A corresponding to interior points

19 **if** *Dirichlet Boundary Condition* **then**

20 Rows of A_{ext} corresponding to equation for a boundary node is assigned the corresponding row of $(N+1)^2 \times (N+1)^2$ identity matrix

21 **else**

22 **if** *Neumann Boundary Condition* **then**

23 Row of A_{ext} corresponding to equation for a boundary node gets assigned by corresponding row of partial derivative matrix,($I_N \otimes D_1$ or $D_1 \otimes I_N$)

24 **end**

25 **end**

26 Vertically concatenate matrices A_{ext} and A_{int} and vectors $\vec{f}_{ext}$ and $\vec{f}_{int}$.

27 Solve linear system to get $\vec{u}$

28 **end**

V. Initial Boundary Value Problems

The partial differential equations in boundary value problems (e.g. Eq. 8) are steady, meaning that the solution u does not have a time dependence. However, many physical systems and problems in engineering typically have a time dependence. As such, this leads to problems with derivatives in time and in space supplemented with boundary conditions and initial conditions, also known as initial boundary value problems.

A. Time discretization

As a prototype example of an initial boundary value problem, we will consider the heat conduction in two dimensions of a material with uniform material properties and without any internal heat generation occurring. This problem is supplemented with homogeneous Dirichlet boundary conditions and an initial condition and is solved on a square domain, summarized by Eq.

19

$$\begin{cases} \dfrac{\partial u}{\partial t} = \kappa \nabla^2 u \qquad \forall (x, y, t) \in R_{TSD} \\ u(x = 0, y, t) = 0, \quad u(x, y = 0, t) = 0 \qquad \forall t \in [0, T] \\ u(x = 1, y, t) = 0, \quad u(x, y = 1, t) = 0 \qquad \forall t \in [0, T] \\ u(x, y, t = 0) = u_0(x, y) \qquad \forall (x, y) \in R \end{cases} \tag{19}$$

where R is defined as above in Sec. IV.A and R_{TSD} is defined as

$$R_{TSD} \equiv \{(x, y, t) \mid (x, y) \in [-1, 1] \times [-1, 1], t \in [0, T]\} \tag{20}$$

In order to solve the initial boundary value problem, the differential equation must be discretized in both space and in time. In practice, to use the Chebyshev spectral collocation method for solving approximating the solution to an initial boundary value problem, the spatial variables (x and y) are discretized spectrally using the tensor product grid of Chebyshev nodes and the time variable is discretized using a finite difference (FD) formula [2]. As with the assumption made when extending to 2D BVPs, we also assume the approximation to the solution to the BVP can be separated temporally and spatially via Eq. 21.

$$u_N(x, y, t) = \sum_{j=0}^{N} \sum_{k=0}^{N} \hat{u}_N(x_j, y_k, t) T_j(x) T_k(y) \tag{21}$$

In a similar fashion, it is clear to see that basis functions and coefficients are unaffected by differentiation w.r.t x or y. However, due to the fact that we are choosing the coefficients to the basis functions to be time dependent, a derivative w.r.t time will result in each coefficient being differentiated in time. If we collect all of the coefficients to the basis functions into a vector and use the same spatial discretization technique for 2D BVPs, we can arrive at a set of ordinary differential equations that represents a semi-discrete form of the partial differential equation in Eq. 19, as seen in Eq. 22. It is worth noting that the vector $\vec{u}$ in Eq. 22 should consist of only the coefficients corresponding to the interior nodes since the nodes on the boundary have to satisfy the appropriate boundary conditions, as was done for the 2D BVPs.

$$\frac{\partial \vec{u}}{\partial t} = \kappa (D_X^2 \otimes I_Y + I_X \otimes D_Y^2) \vec{u} \tag{22}$$

To arrive at the fully-discrete form of Eq. 19 on the nodes interior to the boundary, a time-stepping finite difference formula (such as Euler's method, fourth-order Runge-Kutta, Adams-Bashforth multistep methods, etc.) is used to discretize the partial derivative in time. For simplicity, we show the fully-discrete form of the 2D heat equation using the explicit Euler's method, as seen in Eq. 23 (where the superscript n denotes the nth time-step in the time-stepping scheme). Furthermore, by denoting $\kappa(D_X^2 \otimes I_Y + I_X \otimes D_Y^2)$ with the matrix $\mathbf{A}$ to represent the spatial discretization operator, we further simplify Eq. 23 into Eq. 24.

$$\frac{\vec{u}^{(n+1)} - \vec{u}^{(n)}}{\Delta t} = \kappa (D_X^2 \otimes I_Y + I_X \otimes D_Y^2) \vec{u}^{(n)} \tag{23}$$

$$\vec{u}^{(n+1)} = (I + \Delta t \mathbf{A}) \vec{u}^{(n)} \tag{24}$$

In relation to the algorithm for solving 2D BVPs, the initial boundary value problem (Eq. 19) only presents a few more modifications. Namely, the output $\vec{u}$ will now be an array of values with each column of the array corresponding to the approximation of the solution on the tensor product grid at a particular time-step. Additionally, the solution output is first initialized by the values of the initial condition $u_0(x, y)$. Then, following the same separation of boundary nodes and interior nodes done for 2D BVPs, the interior nodes are advanced to the next time-step with the time-stepping finite difference formula as in Eq. 24 while the boundary nodes at the next time step is the value of the boundary conditions at a grid point (x_j, y_k) at the time-step t_n. The time advancement repeats until the end time T is reached.

B. Stability Requirements

When applying an explicit time-stepping method to solve the initial boundary value problem, the explicit time-stepping algorithm is limited by computational instability known as the Courant-Friedrichs-Lewy (CFL) instability [1]. In fact, according to Boyd [1], we have the following definition:

Definition V.1 *When the timestep τ for an EXPLICIT time-marching method exceeds a limit τ_{max} that the depends on the time-marching algorithm, the physics of the underlying partial differential equation, and the spatial resolution, the error grows exponentially with time. This exponential error growth is the CFL Instability. All explicit methods have a finite τ_{max}, but many implicit methods are stable for arbitrarily large timesteps.*

One way to determine this CFL condition is by requiring that the time-stepping algorithm to be operating within its region of stability. More specifically, Trefethen notes this is usually satisfied when the chosen timestep scaled by the eigenvalues of the spatial discretization operator (e.g $\mathbf{A}$ in Eq. 24) lies within the time-stepping algorithms region of stability [2]. While each time-stepping algorithm has its own stability region, we will consider the explicit Euler method's stability region, which is the interior of the unit circle in the complex plane centered at -1 on the real axis. For this time-stepping method, the CFL condition is given by Eq. 25, where λ will be the largest eigenvalue of the spatial discretization operator [1].

$$\Delta t < \frac{1}{|\lambda|} \tag{25}$$

For Chebyshev spectral collocation, it is determined that the largest eigenvalues of the second derivative matrix in two dimensions will scale like $-0.096N^4$. Consequently, considering the 2D heat equation example (Eq. 19), the time-step requirement to ensure stability is given by Eq. 26. As a conservative estimate, for BVPs containing higher derivatives, Boyd notes that the time-step need to ensure that the CFL condition is satisfied should be on the order of $O\left(\frac{1}{N^{2j}}\right)$ for the Chebyshev spectral collocation method, where j is the highest order derivative in the differential equation.

$$\Delta t < \frac{1}{|0.096N^4|} \tag{26}$$

VI. Numerical Experiments

Having covered the relevant background for the Chebyshev spectral collocation method to solve 2D BVPs, we now conduct some numerical experiments to see the theory in practice. In particular, in the following sections, we will solve the 2D Helmholtz equation and the 2D Heat equation with heat generation and observe the error with the exact solution.

A. Helmholtz Equation

Following the pseudocode for the solving 2D BVPs with Chebyshev spectral collocation, we now test this on a 2D BVP involving the Helmholtz equation. Ma, et al. [3] gives a relatively general form of the 2D Helmholtz equation and solves the partial differential equation with a set of Robin boundary conditions on a $[-1, 1] \times [-1, 1]$ square, given by Eq. 27, using Chebyshev spectral collocation and a Chebyshev-Galerkin method. Letting R denote our domain defined in Sec. IV.A, then we shall consider the following BVP:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \alpha(x,y)\dfrac{\partial u}{\partial x} + \beta(x,y)\dfrac{\partial u}{\partial y} + \gamma(x,y)u = f(x,y) \qquad \forall (x,y) \in R \\ \left(u + k_1(x,y)\dfrac{\partial u}{\partial x}\right)\Big|_{|x|=1} = 0, \quad \left(u + k_2(x,y)\dfrac{\partial u}{\partial y}\right)\Big|_{|y|=1} = 0 \end{cases} \tag{27}$$

B. Convergence study for Helmholtz Equation

Ma, et al. considered a test case for solving Eq. 27 with $\alpha = 1, \beta = 1,\ \gamma = 1$, and $f(x, y)$ given by Eq. 28. Furthermore, the exact solution to this BVP is given by Eq. 29. Since Ma, et al. solved the BVP with Chebyshev spectral collocation, we compare our results with the exact solution as well as the results provided by [3] by examining the relative error in the L_2-norm.

$$\begin{aligned} f(x,y) = \left(-5\pi^2 + 1\right)\sin\left(\pi x + \frac{\pi}{4}\right)\sin\left(2\pi y + \frac{\pi}{4}\right) + \pi\cos\left(\pi x + \frac{\pi}{4}\right)\sin\left(2\pi y + \frac{\pi}{4}\right) \\ + 2\pi\sin\left(\pi x + \frac{\pi}{4}\right)\cos\left(2\pi y + \frac{\pi}{4}\right) \end{aligned} \tag{28}$$

$$u(x,y) = \sin\left(\pi x + \frac{\pi}{4}\right)\sin\left(2\pi y + \frac{\pi}{4}\right) \tag{29}$$

Figure 5 shows the relative error in the approximation of the solution to Eq. 27 in the L_2-norm for our implementation of the Chebyshev spectral collocation method and the L_2-norm errors reported in [3]. As seen in Fig. 5, our implementation of the Chebyshev collocation yields similar relative error results as in [3], indicating that the implementation is done correctly. It is worth noting that the number of nodes in Figure 5 is actually the square of the number of Chebyshev nodes used in one direction, since a 2D BVP is being solved. Additionally, we observe the exponential convergence rate in Figure 5 of the Chebyshev collocation method, as expected from using a spectral method to approximate a solution that is smooth and analytic. Lastly, it can also be seen that relative error starts to level off as the total number of nodes is approximately larger than 600, which is likely due to the set of linear equations being solved being more ill-conditioned as the total number of nodes increases.

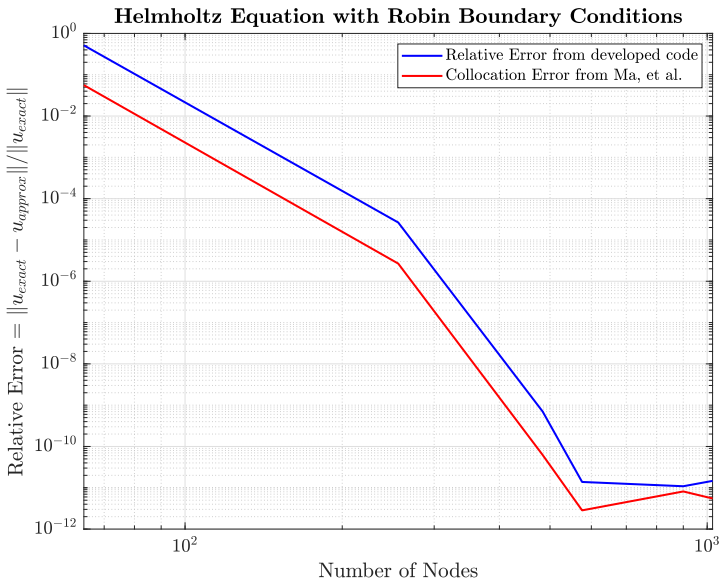

Fig. 5 Relative Error vs Number of Nodes

C. Heat Equation

After incorporating a FD formula to discretize our PDEs in time into our Spectral Method code, see code below, we are now able to approximate the solutions to the following two dimensional heat equation with constant coefficients. Note that we have utilized our shorthand, R and R_{TSD}, again, which was presented in SubsectionIV.A and Eq.20 respectively.

$$\begin{cases} \dfrac{\partial u}{\partial t} - \kappa(x,y)\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) = f(x,y,t), & \forall (x,y,t) \in R_{TSD} \\ \\ u(x=-1,y,t) = -e^{-t^2} sin(\pi y), \quad u(x,y=-1,t) = -e^{-t^2} sin(\pi x), & \forall t \in [0,T] \\ u(x=1,y,t) \quad = e^{-t^2} sin(\pi y), \quad u(x,y=1,t) \quad = e^{-t^2} sin(\pi x), & \forall t \in [0,T] \\ u(x,y,t=0) = sin(\pi xy), & \forall (x,y) \in R \end{cases} \tag{30}$$

where we will consider the situation when $\kappa(x,y) = 1$ and $f(x,y,t)$ is as follows

$$f(x,y,t) = \left[-2t + \pi^2(x^2+y^2)\right] e^{-t^2} sin(\pi xy) \tag{31}$$

D. Convergence study for Heat Equation

Now let's take a look at the stability and convergence of our spectral method for the 2D Heat Equation described via Eq. 30 and Eq.31 just above. We know the exact solution to this particular PDE happens to be

$$u(x,y,t) = e^{-t^2} sin(\pi xy), \tag{32}$$

meaning we can compare our approximated solution with the exact one to see how the error is behaving, see Figure 6. The difference between these two is that the good approximation had a time

step size in accordance with the CFL Condition, making sure it was satisfied, see Def. V.1, [1] and [4]. Giving us insight into the stability of our numerical method. Now that we have checked for accuracy and stability, we can now move on to looking into the convergence of each of our discretizations, both in time and space, of our spectral method. Aligned with the results presented in [4] and [1], we were able to test the stability, including our conditioning number, as well as the convergence of our time-stepping spectral method in 2D. Figure 7 is comprised of the same plot on two different axes: linear and semi-log axes. As seen in Fig. 7, because Euler's method is a first order approximation, our FD discretization in time is performing as expected with decent accuracy since the error decreases by half every time the time step size is halved. Now that we have examined how our time discretization behaves, lets now focus our attention to our spatial discretizations. Note, we do not have to check both spatial discretizations since we collapsed our vector of interest, u, into a 1D vector of columns.

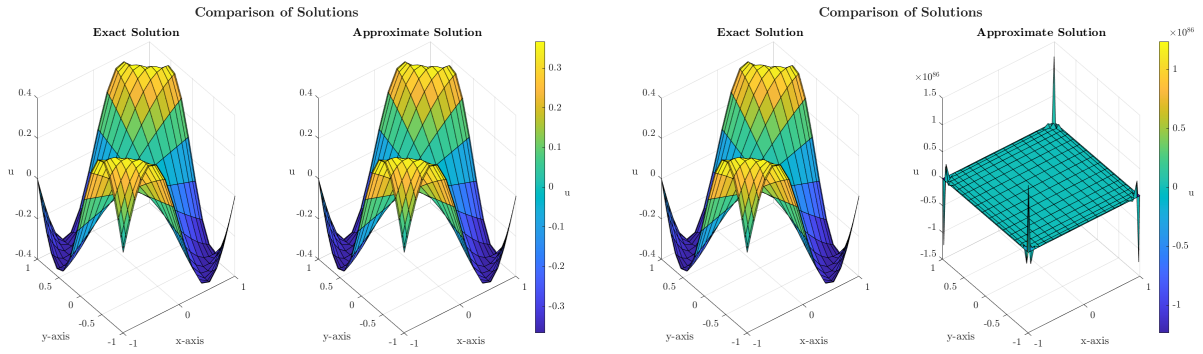

Fig. 6 Solution comparison: the good (left) and the bad (right)

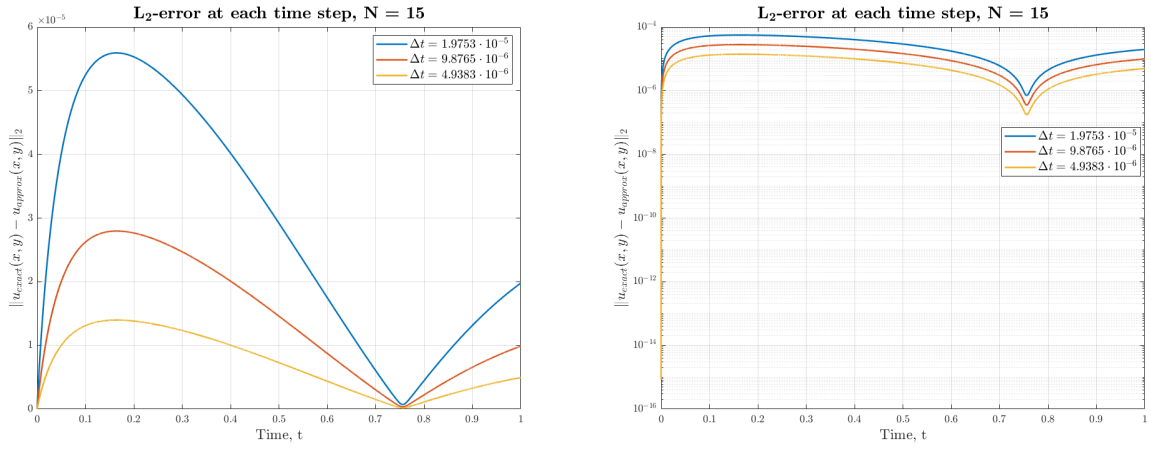

Fig. 7 The L_2 – Error at each time step with Linear Axes (left) and with Log-Linear Axes (right) both with N = 15

The important take away from Figure 8 is that a with an increased number of nodes it appears that the errors starts to slightly increase more. This could be explained by the spectral differentiation matrix conditioning number vs. the number of nodes plot to the right of Figure 8. Figure 8, shows that condition number is rising exponentially with increasing number of nodes, indicating that round-off error due to finite precision arithmetic can play a significant part the error. However, the plots in Fig. 8 were done using the time-step required for most number of nodes used in the study. Let's now investigate what would happen if we were to relax our time-step in such a way that it just satisfies the CFL condition for the given number of nodes used, see Figure 9.

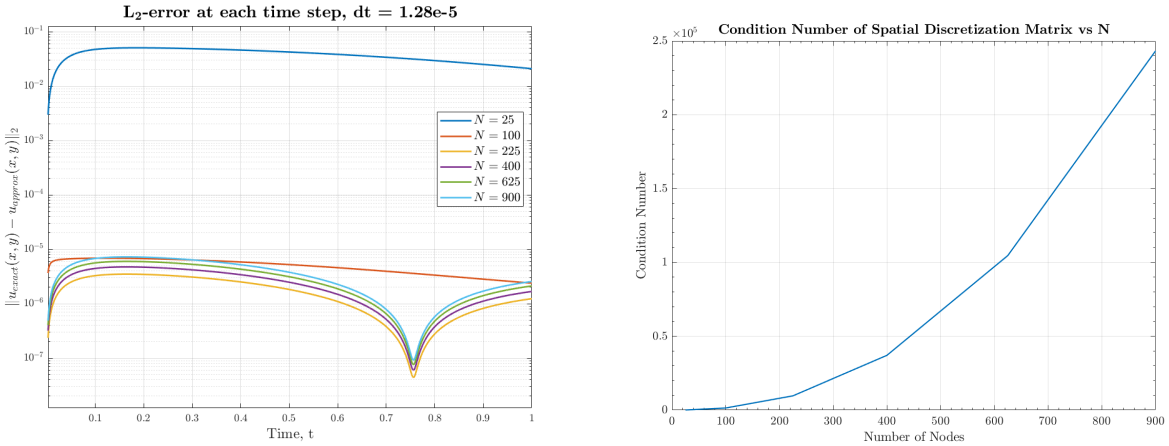

Fig. 8 The L_2 − Error at each time step with dt = $1.28 * 10^{-5}$ (left) and the Condition Number of Spectral Differentiation Matrix vs. Number of Nodes (right)

Below is Figure 9, showing the L_2-error at each time-step and taking a look at the behavior of different values of dt and N. Note that dt was chosen specifically such that it is to always satisfy the CFL condition for any number of nodes, N, used in the approximation. It is important to notice that now we are seeing better performance in terms of L_2-error when more nodes are used to approximate the solution. In comparison to Figure 8, we are now able to conclude that it was actually the smaller time-step that made the approximations with less nodes slightly more accurate as compared to the approximations with more nodes. Lastly, in Fig. 9, we also see that the spectral convergence is initially exhibited (when going from 25 to 100 nodes used in the spatial discretization), but the method is eventually slowed down to first order when further increasing the number of nodes used for the spatial discretization. This is expected as the temporal and spatial discretizations are of different orders and the convergence rate of our method will be limited by the lower order method (Euler's method).

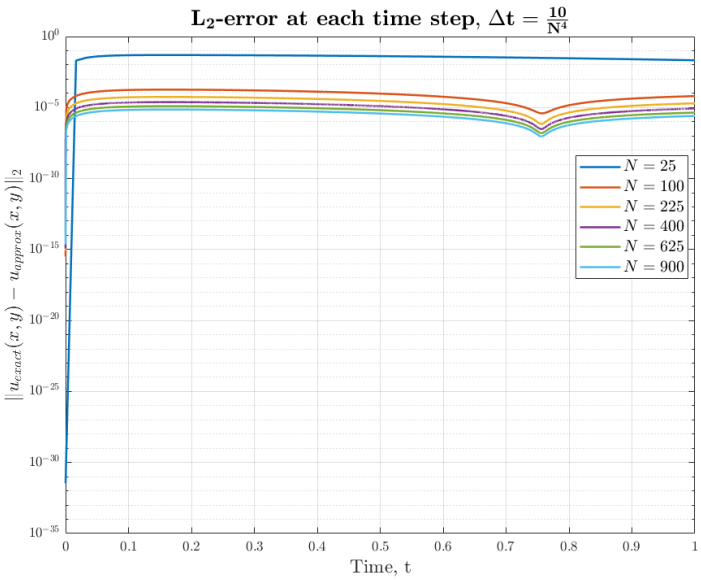

Fig. 9 The L_2 – Error at each time step, time-stepping aligned with CFL condition

VII. Conclusion

When challenged with solving a 2D BVP, there are many potential numerical methods one can employ to approximate a solution. Currently, the three main techniques for solving BVPs are FD, FEM, and spectral methods. We've seen that spectral methods converge at a much faster rate (exponentially) when compared to FD and FEM, but spectral methods require solving a dense linear system of equations, see right of Figure 3. In addition to solve BVPs in 2D, we also took a look into IBVPs, which are initial boundary value problems, Section V, as well as stability and convergence requirements when incorporating a time discretization into your approximation that already has two spatial discretizations. We found that the CFL condition is a non-negotiable condition that must be satisfied for us to achieve an accurate approximation. When solving both BVPs and IBVPs, incorporating the boundary conditions is the most difficult task when coding these numerical methods. Although Dirichlet BCs are easy to implement, when working with Neumann or Robin BCs it can be incredibly difficult to ensure your BCs are not only in the correct position that correspond to the boundary of our square, but are also incorporating the correct spectral differentiation matrices. For on-going and future work, we would be taking a look into conformal mapping to deal with more complicated geometries in 2D [1] or even potentially look into a spectral element method for complex geometries. We would also like to do some exploration into the Fourier collocation method and the spectral Galerkin method for both Chebyshev and Fourier expansions. Lastly, in addition to all of this, taking a look into nonlinear problems would wrap up our research of Spectral Methods for solving 2D BVPs very nicely.

VIII. Appendix

Here is a useful, and very critical, mathematical definition that is pivotal in using Spectral Methods in 2D.

A. The Kronecker Product

For extending the ideas presented in the 1D situation to the 2D situation, we first need to know a little about the Kronecker product for matrices, denoted by "⊗". Suppose we have two matrices, A and B, with dimensions $M \times N$ and $P \times Q$ respectively, then $A \otimes B$ is a $MP \times NQ$ matrix of the following form:

$$A \otimes B = \begin{bmatrix} a_{1,1}B & \cdots & a_{1,N}B \\ \vdots & \ddots & \vdots \\ a_{M,1}B & \cdots & a_{M,N}B \end{bmatrix} \tag{33}$$

This is valuable in that it allows us to clearly separate all of the different operators that appear when dealing with higher dimensions.

Acknowledgements

We would like to acknowledge Professor Adrianna Gillman for providing references (literature and demo codes) and general background knowledge to help do this project.

References

[1] Boyd, J. P., "Chebyshev and Fourier Spectral Methods," 2000, p. 611.

[2] Trefethen, L. N., *Spectral methods in MATLAB*, Software, environments, tools, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[3] Ma, X., Wang, Y., Zhu, X., Liu, W., Lan, Q., and Xiao, W., "A Spectral Method for Two-Dimensional Ocean Acoustic Propagation," *Journal of Marine Science and Engineering*, Vol. 9, No. 8, 2021, p. 892. https://doi.org/10.3390/jmse9080892, URL https://www.mdpi.com/2077-1312/9/8/892.

[4] Gottlieb, D., "The Stability of Pseudospectral-Chebyshev Methods," *Mathematics of Computation*, 1981, p. 12.

Code

B. Main Script for solving problems

```
%% APPM 4660 Project - Main
% Purpose: Code developed for to solve 2D boundary value problems
    with
% Chebyshev Spectral Collocation
```

```
% Author: Derrick Choi
% Date: 4/5/2022
% Last Modified: 4/26/2022

% Things to do: check derivative operations with different kinds
   of
% functions that exhibit different convergence rates, do a
   problem with
% variable coefficients, check conditioning of stretching domains
   , compare
% convergence rates in space and in time
clc;clear;close all;
%% Check Derivative operators

N = 5:5:25;
a = -1;
b = 1;

error = zeros(length(N),4);
error2 = zeros(length(N),4);
error3 = zeros(length(N),4);

g = @(x,y) exp(x).*sin(y);
gx = @(x,y) exp(x).*sin(y);
gy = @(x,y) exp(x).*cos(y);
gxx = @(x,y) exp(x).*sin(y);
gyy = @(x,y) -exp(x).*sin(y);

h = @(x,y) exp(-1./(sin(x/2+y).^2));
hx = @(x,y) (cos(x/2 + y).*exp(-1./sin(x/2 + y).^2))./sin(x/2 + y
   ).^3;
hxx = @(x,y) (cos(x/2 + y).^2.*exp(-1./sin(x/2 + y).^2))./sin(x/2
    + y).^6 - (3*cos(x/2 + y).^2.*exp(-1./sin(x/2 + y).^2))./(2*
   sin(x/2 + y).^4) - exp(-1./sin(x/2 + y).^2)./(2*sin(x/2 + y)
   .^2);
hy = @(x,y) (2*cos(x/2 + y).*exp(-1./sin(x/2 + y).^2))./sin(x/2 +
    y).^3;
hyy = @(x,y) (4*cos(x/2 + y).^2.*exp(-1./sin(x/2 + y).^2))./sin(x
   /2 + y).^6 - (6*cos(x/2 + y).^2.*exp(-1./sin(x/2 + y).^2))./
   sin(x/2 + y).^4 - (2*exp(-1./sin(x/2 + y).^2))./sin(x/2 + y)
   .^2;

r = @(x,y) sin(x.*y+1);
rx = @(x,y) y.*cos(x.*y+1);
ry = @(x,y) x.*cos(x.*y+1);
```

```
rxx = @(x,y) -y.^2.*sin(x.*y+1);
ryy = @(x,y) -x.^2.*sin(x.*y+1);

for i = 1:length(N)
    [D,x] = cheb(N(i)-1);
%     x = x(end:-1:1);
    x = (b-a)/2*x+(b+a)/2;
    J = (b-a)/2; % jacobian
    D = 1/J*D;
    D2 = D*D;
    y = x;

    [xx,yy] = meshgrid(x,y);

    xx = xx(:);
    yy = yy(:);

    % Determine interior vs exterior points
    logic_idx = xx==a|xx==b|yy==a|yy==b;
    Ext_pt = find(logic_idx);
    Int_pt = find(~logic_idx);

    % Order exterior points so that it goes from bottom left
       corner and moves
    % counterclockwise
    xcenter = (b+a)/2;
    ycenter = (b+a)/2;
    % reference angle to determine point locations relative to
       center
    theta0 = atan2(yy(1)-ycenter,xx(1)-xcenter);
    theta = rem(4*pi+1e-12-theta0+atan2(yy(Ext_pt)-ycenter,xx(
       Ext_pt)-xcenter),2*pi);
    [~,order_ccw] = sort(theta);
    Ext_pt = Ext_pt(order_ccw);

%     xx = [xx(Ext_pt);xx(Int_pt)];
%     yy = [yy(Ext_pt);yy(Int_pt)];
    % Partial Derivative operators
    du_dydy = kron(eye(N(i)),D2);
%     du_dxdx = [du_dxdx(Ext_pt,:);du_dxdx(Int_pt,:)];
    du_dxdx = kron(D2,eye(N(i)));
%     du_dydy = [du_dydy(Ext_pt,:);du_dydy(Int_pt,:)];
    du_dy =   kron(eye(N(i)),D);
%     du_dy = [du_dy(Ext_pt,:);du_dy(Int_pt,:)];
    du_dx =   kron(D,eye(N(i)));
```

```
%     du_dx = [du_dx(Ext_pt,:);du_dx(Int_pt,:)];

    test_partialx = du_dx*g(xx,yy);
    test_partialy = du_dy*g(xx,yy);
    test_partialxx = du_dxdx*g(xx,yy);
    test_partialyy = du_dydy*g(xx,yy);

    exact_partialx = gx(xx,yy);
    exact_partialy = gy(xx,yy);
    exact_partialxx = gxx(xx,yy);
    exact_partialyy = gyy(xx,yy);

    test_partialx2 = du_dx*h(xx,yy);
    test_partialy2 = du_dy*h(xx,yy);
    test_partialxx2 = du_dxdx*h(xx,yy);
    test_partialyy2 = du_dydy*h(xx,yy);

    exact_partialx2 = hx(xx,yy);
    exact_partialy2 = hy(xx,yy);
    exact_partialxx2 = hxx(xx,yy);
    exact_partialyy2 = hyy(xx,yy);

    test_partialx3 = du_dx*r(xx,yy);
    test_partialy3 = du_dy*r(xx,yy);
    test_partialxx3 = du_dxdx*r(xx,yy);
    test_partialyy3 = du_dydy*r(xx,yy);

    exact_partialx3 = rx(xx,yy);
    exact_partialy3 = ry(xx,yy);
    exact_partialxx3 = rxx(xx,yy);
    exact_partialyy3 = ryy(xx,yy);

    error(i,:) = [ norm(test_partialx-exact_partialx) norm(
       test_partialy-exact_partialy)...
        norm(exact_partialxx-test_partialxx) norm(exact_partialyy
           -test_partialyy)];
    error2(i,:) = [ norm(test_partialx2-exact_partialx2) norm(
       test_partialy2-exact_partialy2)...
        norm(exact_partialxx2-test_partialxx2) norm(
           exact_partialyy2-test_partialyy2)];
    error3(i,:) = [ norm(test_partialx3-exact_partialx3) norm(
       test_partialy3-exact_partialy3)...
        norm(exact_partialxx3-test_partialxx3) norm(
           exact_partialyy3-test_partialyy3)];
end
```

```
figure('Position',[200 100 1000 600])
tiledlayout(2,2)
nexttile
loglog(N.^2,error(:,1),'LineWidth',1.5)
hold on
loglog(N.^2,error2(:,1),'LineWidth',1.5)
loglog(N.^2,error3(:,1),'LineWidth',1.5)
xlabel('Number of Nodes','FontSize',12,'Interpreter','latex')
ylabel('$\|u_{exact}-u_{approx}\|_2$','FontSize',12,'Interpreter'
   ,'latex')
title('${\partial u/\partial x}$','FontSize',14,'Interpreter','
   latex')
grid on
set(gca,'TickLabelInterpreter','latex')

nexttile
loglog(N.^2,error(:,2),'LineWidth',1.5)
hold on
loglog(N.^2,error2(:,2),'LineWidth',1.5)
loglog(N.^2,error3(:,2),'LineWidth',1.5)
xlabel('Number of Nodes','FontSize',12,'Interpreter','latex')
ylabel('$\|u_{exact}-u_{approx}\|_2$','FontSize',12,'Interpreter'
   ,'latex')
title('${\partial u/\partial y}$','FontSize',14,'Interpreter','
   latex')
grid on
set(gca,'TickLabelInterpreter','latex')

nexttile
loglog(N.^2,error(:,3),'LineWidth',1.5)
hold on
loglog(N.^2,error2(:,3),'LineWidth',1.5)
loglog(N.^2,error3(:,3),'LineWidth',1.5)
xlabel('Number of Nodes','FontSize',12,'Interpreter','latex')
ylabel('$\|u_{exact}-u_{approx}\|_2$','FontSize',12,'Interpreter'
   ,'latex')
title('${\partial^2 u/\partial x^2}$','FontSize',14,'Interpreter'
   ,'latex')
set(gca,'TickLabelInterpreter','latex')
grid on

nexttile
loglog(N.^2,error(:,4),'LineWidth',1.5)
hold on
loglog(N.^2,error2(:,4),'LineWidth',1.5)
```

```
loglog(N.^2,error3(:,4),'LineWidth',1.5)
xlabel('Number of Nodes','FontSize',12,'Interpreter','latex')
ylabel('$\|u_{exact}-u_{approx}\|_2$','FontSize',12,'Interpreter'
   ,'latex')
title('${\partial^2 u/\partial y^2}$','FontSize',14,'Interpreter'
   ,'latex')
grid on
set(gca,'TickLabelInterpreter','latex')
sgtitle('\textbf{Convergence Rates for Different Kinds of
   Functions}','Interpreter','latex','FontSize',16)
lgd = legend('$e^x\sin(y)$','$e^{-1/(\sin(x/2+y)^2)}$','$\sin(xy
   +1)$','Interpreter','latex','FontSize',12);
lgd.Layout.Tile = 'east';
%% Verify with poisson_spec (it works)
u_exact = @(x,y) exp(x+y/2);
alpha = @(x,y) 0.*x;
beta = @(x,y) 0.*x;
gamma = @(x,y) 0.*x;

a = 0;
b = 1;

f = @(x,y) 1.25*exp(x+y/2);
[xx,yy,u] = SpectralCollocation_2D_BVP(alpha,beta,gamma,f,7,a,b,'
   Dirichlet');
load('uapptest.mat')

xx = reshape(xx,7,7);
yy = reshape(yy,7,7);
u = reshape(u,7,7);

figure('Name','Test_Poisson')
surfc(xx,yy,u)
xlabel('x','FontSize',12,'Interpreter','latex')
ylabel('y','FontSize',12,'Interpreter','latex')
zlabel('u','FontSize',12,'Interpreter','latex')
set(gca,'TickLabelInterpreter','latex','FontSize',12)
title('\textbf{Solution to $\mathbf{u_{xx}+u_{yy} = 5/4e^{x+y
   /2}}$}','Interpreter','latex','FontSize',16)
%% Helmholtz Equation for Ocean Acoustics

% Set-up
N = [8 16 22 24 30 32];
a = -1;
b = 1;
```

```
alpha = @(x,y) 1*ones(length(x),1);
beta = @(x,y) 1*ones(length(x),1);
gamma = @(x,y) 1*ones(length(x),1);

f = @(x,y) (-5*pi^2 + 1)*sin(pi*x+pi/4).*sin(2*pi*y+pi/4)+...
    pi*cos(pi*x+pi/4).*sin(2*pi*y+pi/4)+2*pi*sin(pi*x+pi/4).*cos
       (2*pi*y+pi/4);
u_exact = @(x,y) sin(pi*x+pi/4).*sin(2*pi*y+pi/4);

rel_err = zeros(1,length(N));
paper_rel_err = [5.6e-2 2.68e-6 6.24e-11 2.84e-12 8.15e-12 5.51e
   -12];
for numnodes = 1:length(N)
    % Solve with spectral collocation
    [xgrid,ygrid,u] = SpectralCollocation_2D_BVP(alpha,beta,gamma
       ,f,N(numnodes),a,b,'Robin');
    rel_err(numnodes) = norm(u_exact(xgrid,ygrid)-u)/norm(u_exact
       (xgrid,ygrid));

end
fprintf(1,'||uex - uapp||/||uex|| = %17.8e\n',norm(u_exact(xgrid,
   ygrid)-u)/norm(u_exact(xgrid,ygrid)));

% surface plot
xgrid = reshape(xgrid,N(end),N(end));
ygrid = reshape(ygrid,N(end),N(end));
u_grid = reshape(u,N(end),N(end));


% Plot of solution and error
figure('Name','Test_Helmholtz')
surf(xgrid,ygrid,u_grid)
%fsurf(u_exact,[a b a b])

figure('Name','Convergence plot','Position',[400 100 800 600])
semilogy(N.^2,rel_err,'b','LineWidth',1.5);
hold on
semilogy(N.^2,paper_rel_err,'r','LineWidth',1.5)
legend('Relative Error from developed code','Collocation Error
   from Ma, et al.','Interpreter','latex','FontSize',12)
grid on
xlabel('Number of Nodes','Interpreter','latex','FontSize',12)
ylabel('Relative Error = $\|u_{exact}-u_{approx}\|/\|u_{exact}\|$
   ','FontSize',12,'Interpreter','latex')
title('\textbf{Helmholtz Equation with Robin Boundary Conditions}
```

```
   ','Interpreter','latex','FontSize',16)
set(gca,'TickLabelInterpreter','latex','FontSize',14)

%% 1D Heat Equation
close all
x0 = 0;
L = 1;
n = 1:1000;
bk = 320./((2*n-1)*pi);

% set-up
kappa = @(x) 0.0017*ones(length(x),1);
a = 0;
b = 1;
dt = 0.00001;
tend = 10;
IC = @(x) ones(length(x),1);
BC = [0; 0];
N = 10;
NT = 100;

[tout,xout,uout] = HeatEq1D(kappa,0,N,a,b,NT,IC,BC);

%% Animate Figure
test = load('u_test.mat');
f1 = figure('Position',[400 100 800 600]);
%plot(xout,uout(:,1));
axis tight manual
ax = gca;
ax.NextPlot = 'replaceChildren';
loops = length(uout(1,:));
M(loops) = struct('cdata',[],'colormap',[]);
f1.Visible = 'off';
mov = zeros(612,775,1,loops,'uint8');
for j = 1:loops
    plot(xout,uout(:,j),'LineWidth',1.5)
    hold on
    plot(test.x,test.u(:,j),'LineWidth',1.5)
    legend('Spectral Collocation','FEM','Interpreter','latex')
    xlabel('x-axis','FontSize',14,'Interpreter','latex')
    ylabel('y-axis','FontSize',14,'Interpreter','latex')
    title('\textbf{Solution to 1D Heat Equation}','Interpreter','
       latex','FontSize',16)
    set(gca,'TickLabelInterpreter','latex')
    grid on
```

```
    drawnow
    M(j) = getframe;
    hold off
    if j == 1
        [mov(:,:,1,j), map] = rgb2ind(M(j).cdata, 256, 'nodither'
           );
    else
        mov(:,:,1,j) = rgb2ind(M(j).cdata, map, 'nodither');
    end
end
f1.Visible = 'on';
movie(M);
% Create gif
imwrite(mov,map,'HeatEQ1D.gif','DelayTime',0,'LoopCount',Inf)
% approx exact
% u_exact = @(x,t) 100-sum(bk.*sin((2*n-1)*pi.*x)/L.*exp(-((2*n
   -1)*pi/L).^2*0.0017.*t));
% u_ap = zeros(length(xout),length(xout));
% Evaluate Exact
% for j = 1:length(xout)
%     for i = 1:length(tout)
%         u_ap(j,i) = u_exact(xout(j),tout(i));
%     end
% end

%% 2D-Heat Equation
K = @(x,y) ones(length(x),1);
% f = @(x,y,t) 0.*x; % test problem with no forcing
f = @(x,y,t) exp(-t.^2).*sin(pi.*x.*y).*(-2*t+pi^2*(x.^2+y.^2));
a = -1;
b = 1;
N = 5:5:30;

dt = min(1./(0.048*2*N.^4));
t0 = 0;
tend = 2;
% Time Error
t_err = cell(2,length(dt));
con = zeros(length(dt),1);

for m = 1:length(N)
    fprintf('Solving with Spectral Collocation: dt = %f\n',dt)
    [tout,xout,yout,uout,con(m)] = HeatEq2D(K,f,dt,t0,tend,N(m),a
       ,b,'Dirichlet');
```

```
    [xgrid,ygrid] = meshgrid(xout,yout);
    % Compare Error
    % u_exact = @(x,y,t) exp(-2*t).*sin(x).*sin(y); % exact
       solution for test problem
    u_exact = @(x,y,t) exp(-t.^2).*sin(pi*x.*y);
    fprintf('Computing Exact Solution!\n')
    exact = zeros(N(m),N(m),length(tout));
    for k = 1:length(tout)
        for i = 1:length(xout)
            for j = 1:length(yout)
                exact(i,j,k) = u_exact(xout(i),yout(j),tout(k));
            end
        end
    end
    fprintf('Computing Error in Time!\n')
    for id_t = 1:length(tout)
        approx = reshape(uout(:,id_t),N(m),N(m));
        err = reshape(exact(:,:,id_t)-approx,N(m)^2,1);
        t_err{1,m}(id_t,1) = norm(err,2);
    end
    t_err{2,m} = tout;
end

figure('Position',[400 100 1000 600])
tiledlayout(1,2)
nexttile

surf(xgrid,ygrid,exact(:,:,end))
xlabel('x-axis','Interpreter','latex','FontSize',12)
ylabel('y-axis','Interpreter','latex','FontSize',12)
zlabel('u','FontSize',12,'Interpreter','latex','Rotation',0);
set(gca,'TickLabelInterpreter','latex','FontSize',12)
title('\textbf{Exact Solution}','Interpreter','latex','FontSize'
   ,14)

nexttile
surf(xgrid,ygrid,reshape(uout(:,end),N(m),N(m)))
xlabel('x-axis','Interpreter','latex','FontSize',12)
ylabel('y-axis','Interpreter','latex','FontSize',12)
zlabel('u','FontSize',12,'Interpreter','latex','Rotation',0);
set(gca,'TickLabelInterpreter','latex','FontSize',12)
title('\textbf{Approximate Solution}','Interpreter','latex','
   FontSize',14)

sgtitle('\textbf{Comparison of Solutions}','FontSize',16,'
```

```
   Interpreter','latex')
cbar = colorbar;
cbar.Layout.Tile = 'east';
cbar.Label.String = 'u';
cbar.Label.Interpreter = 'latex';
cbar.TickLabelInterpreter = 'latex';
cbar.FontSize = 12;
cbar.Label.Rotation = 0;
% black = [0 0 0];
% gold = [207 184 124]/255;
% colors_p = [linspace(black(1),gold(1),1000)', linspace(black(2)
   ,gold(2),1000)', linspace(black(3),gold(3),1000)'];
% colormap(colors_p)
%{
%%
figure('Name','HeatEq2D_TimeError','Position',[400 100 800 600])
for m = 1:size(t_err,2)
semilogy(t_err{2,m},t_err{1,m},'LineWidth',1.5)
hold on
end
xlabel('Time, t','Interpreter','latex','FontSize',14)
ylabel('$\|u_{exact}(x,y)-u_{approx}(x,y)\|_2$','FontSize',14,'
   Interpreter','latex')
title('$\mathbf{L_2}$\textbf{-error at each time step}, $\Delta \
   mathbf{t = \frac{10}{N^{4}}}$','Interpreter','latex','FontSize
   ',16)
grid on
% legend('$N = 25$','$N = 100$','$N = 225$','$N = 400$','$N =
   625$','$N = 900$','Interpreter','latex','FontSize',12)
%legend('$\Delta t = 1.9753 \cdot 10^{-5}$','$\Delta t = 9.8765 \
   cdot 10^{-6}$','$\Delta t = 4.9383 \cdot 10^{-6}$'...
%   ,'Interpreter','latex','FontSize',12)
set(gca,'TickLabelInterpreter','latex')
%%
%}
%%
f2 = figure('Position',[400 100 800 600]);
axis tight manual
ax = gca;
% ax.View = [-12.2,51.22];
% ax.CameraPosition =
   [-2.311012994955346,-10.599436255823809,14.842021805308326];
% ax.CameraViewAngle = [-10.2541];
zlim([min(uout,[],'all')-0.1 max(uout,[],'all')+0.1])
grid on
```

```
ax.NextPlot = 'replaceChildren';
loops = length(uout(1,:));
M(loops) = struct('cdata',[],'colormap',[]);
f2.Visible = 'off';

it = 1:100:loops;
v = VideoWriter('HeatEq2DAnim.mp4');
open(v)
for j = 1:length(it)
    approx_u = reshape(uout(:,it(j)),N,N);
    surf(xgrid,ygrid,approx_u,'EdgeColor','flat')
    xlabel('x-axis','FontSize',14,'Interpreter','latex')
    ylabel('y-axis','FontSize',14,'Interpreter','latex')
    zlabel('u(x,y)','FontSize',14,'Interpreter','latex')
    title('\textbf{Solution to 2D Heat Equation with Forcing}','
       Interpreter','latex','FontSize',16)
    ax.TickLabelInterpreter = 'latex';
    xlim([a b])
    ylim([a b])
    grid on
    colorbar
    drawnow
    M(j) = getframe;
    im{j} = frame2im(M(j));
    writeVideo(v,M(j))
end
close(v)

for idx = 1:length(it)
    [A,map] = rgb2ind(im{idx},256);
    if idx == 1
        imwrite(A,map,'HeatEQ2D.gif','gif','LoopCount',Inf,'
           DelayTime',0)
    else
        imwrite(A,map,'HeatEQ2D.gif','gif','WriteMode','append','
           DelayTime',0)
    end
end
```

C. Spectral Collocation BVP code

```
function [xx,yy,u] = SpectralCollocation_2D_BVP(alpha,beta,gamma,
   f,N,a,b,BC_type)
% Function to solve the 2D helmholtz equation for ocean acoustic
```

```
% propagation given by the form:
% uxx + uyy + a(x,y)ux + b(x,y)uy + c(x,y)u = f(x,y)
% on a square (a,b) x (a,b)
%
% Inputs:
%        alpha,beta,gamma = variable coefficient function handles
%        f = forcing term function handle
%        N = number of collocation points along one direction
%        a,b = left and right endpoints of x-dimension (
   respectively)
%        BC_type = type of Boundary Condition
% Outputs:
%        xx,yy = spectral collocation mesh
%        u = approximation to the solution of the
% Authors: Derrick Choi and Ryan Stewart
%
% Note: Method uses Chebyshev spectral collocation
%
%       cheb.m function is from the text Spectral Methods in
   Matlab by
%       Trefethen. Method for obtaining interior nodes are
   adapted from
%       poisson_spec code given by Professor Gillman
%
%       (modification for boundary conditions done by user in
   separate
%       subfunction and currently assumes one kind of Boundary
   condition for all
%       sides of the square)


% Get chebyshev differentiation matrix for interior points
[D,x] = cheb(N-1);
x = x(end:-1:1);
% map to interval of interest
x = (b-a)/2*x+(b+a)/2;
J = (b-a)/2; % jacobian
D = 1/J*D; %derivative operator on interval defined by (a,b)

% second derivative differentiation matrix
D2 = D*D; % 1D second derivative matrix

% Create tensor product grid
y = x;
[xx,yy] = meshgrid(x,y);
```

```
xx = xx(:);
yy = yy(:);

% Derivative approximations of the left hand side
du_dydy = kron(eye(N),D2);
du_dxdx = kron(D2,eye(N));
du_dy =   -kron(eye(N),D);
du_dx =   -kron(D,eye(N));

% Determine interior vs exterior points
logic_idx = xx==a|xx==b|yy==a|yy==b;
Ext_pt = find(logic_idx);
Int_pt = find(~logic_idx);

% Order exterior points so that it goes from bottom left corner
   and moves
% counterclockwise
xcenter = (b+a)/2;
ycenter = (b+a)/2;
% reference angle to determine point locations relative to center
theta0 = atan2(yy(1)-ycenter,xx(1)-xcenter);
theta = rem(4*pi+1e-12-theta0+atan2(yy(Ext_pt)-ycenter,xx(Ext_pt)
   -xcenter),2*pi);
[~,order_ccw] = sort(theta);
Ext_pt = Ext_pt(order_ccw);

% Contributions of variable coefficients
axy = diag(alpha(xx,yy));
bxy = diag(beta(xx,yy));
cxy = diag(gamma(xx,yy));

% check_partial_deriv(xx,yy,du_dx,du_dy,Ext_pt,N,du_dxdx,du_dydy)
   ;

% Equations for BCs
switch BC_type
    case 'Dirichlet'
        [F,fext] = getBC_Eqs(xx,yy,du_dx,du_dy,Ext_pt,N,BC_type);
    case 'Robin'
        [F,fext] = getBC_Eqs(xx,yy,du_dx,du_dy,Ext_pt,N,BC_type);
end

% Equations for interior points
A = du_dxdx+du_dydy+axy*du_dx+bxy*du_dy+cxy;
```

```
% Right hand side
fint = f(xx(Int_pt),yy(Int_pt));
b = [fext;fint];

% Solve linear system
FA = [F;A(Int_pt,:)];
u = FA\b;

end

function [F,fext] = getBC_Eqs(xx,yy,dudx,dudy,Ext_pt,N,BC_type)
% Inputs:
%        xx,yy = 2D grid points flattened as long 1D vectors
%        dudx, dudy = partial derivative matrices
%        Ext_pt = exterior point indices
%        N = number of points on a side
%        BC_type = type of BC
% Outputs:
%        F = Coefficient Matrix for boundary node equations
%        fext = right hand side for boundary nodes
%
Npt = length(xx);

% Matrices
switch BC_type
    case 'Dirichlet'
        F = zeros(length(Ext_pt),Npt);
        F(:,Ext_pt) = eye(length(Ext_pt));
    case 'Neumann'
        % This BC does not work right now (see Robin BC for
           correct implementation)
        idx = reshape(1:N,N/4,4);
        D_bottom = dudy(Ext_pt(idx(:,1)),:);
        D_right = dudx(Ext_pt(idx(:,2)),:);
        D_top = dudy(Ext_pt(idx(:,3)),:);
        D_left = dudx(Ext_pt(idx(:,4)),:);
        F = [D_bottom;D_right;D_top;D_left];

    case 'Robin'

        F = zeros(length(Ext_pt),Npt);
        F(:,Ext_pt) = eye(length(Ext_pt));

        nodeperside = length(Ext_pt)/4;
```

```
        % indices for bottom, right, top, and left sides of the
           square
        b = Ext_pt(1:N-1);
        r = Ext_pt(N:N+nodeperside-1);
        t = Ext_pt(N+nodeperside:N+2*nodeperside-1);
        l = Ext_pt(N+2*nodeperside:end);

        % for the specific problem in the paper, these constants
           are
        % multiplied to the derivative matrices
        D_bottom = -1/(2*pi)*dudy(b,:);
        D_right = -1/pi*dudx(r,:);
        D_top = -1/(2*pi)*dudy(t,:);
        D_left = -1/pi*dudx(l,:);

        % Combine corresponding dirichlet and Neumann entries
        F(1:nodeperside,:) = F(1:nodeperside,:) +D_bottom;
        F(nodeperside+1:2*nodeperside,:) = F(nodeperside+1:2*
           nodeperside,:)+D_right;
        F(2*nodeperside+1:3*nodeperside,:) = F(2*nodeperside+1:3*
           nodeperside,:)+D_top;
        F(3*nodeperside+1:end,:) = F(3*nodeperside+1:end,:)+
           D_left;

end

% Values of boundary condition (need to change depending on the
   problem you
% consider)
%     fext = exp(xx(Ext_pt)+yy(Ext_pt)/2);
    fext = zeros(length(Ext_pt),1);
end

function check_partial_deriv(xx,yy,dudx,dudy,ext,N,dudxdx,dudydy)

% function to check partial derivatives along side of a square
s = ext(1:N);
e = ext(N+1:N+5);
n = ext(N+6:N+11);
w = ext(N+12:end);

u = @(x,y) exp(x).*sin(y);
ux = @(x,y) exp(x).*sin(y);
uy = @(x,y) exp(x).*cos(y);
```

```
Dsx = dudx(s,:);Dex = dudx(e,:);
Dnx = dudx(n,:);Dwx = dudx(w,:);

Dsy = dudy(s,:);Dey = dudy(e,:);
Dny = dudy(n,:);Dwy = dudy(w,:);

ff = u(xx,yy);

test{:,1} = Dsx*ff; exact{:,1} = ux(xx(s),yy(s));
test{:,2} = Dwx*ff; exact{:,2} = ux(xx(w),yy(w));
test{:,3} = Dnx*ff; exact{:,3} = ux(xx(n),yy(n));
test{:,4} = Dex*ff; exact{:,4} = ux(xx(e),yy(e));
test{:,5} = Dsy*ff; exact{:,5} = uy(xx(s),yy(s));
test{:,6} = Dwy*ff; exact{:,6} = uy(xx(w),yy(w));
test{:,7} = Dny*ff; exact{:,7} = uy(xx(n),yy(n));
test{:,8} = Dey*ff; exact{:,8} = uy(xx(e),yy(e));

for i = 1:length(test)
    err(i) = norm(exact{:,i}-test{:,i});
end

figure
plot(err)
% figure
% plot(ux_ex)
% hold on
% plot(test)
% err = norm(ux_ex-test);
disp(err)
end
```

D. Heat Equation in 2D

```
function [tout,xout,yout,u,con] = HeatEq2D(K,f,dt,t0,tend,N,a,b,
   BC_type)
% function to solve 2D-Heat equation on square [a,b] x [a,b] from
    t = 0 to t ~ T
% Heat Eq: ut - k(x)(u_xx+uyy) = f(x,y)
%
%
% Inputs:
%        K = material thermal diffusivity
%        f = forcing function
%        t0 = start time
```

```
%        tend = stop time
%        dt = time step size
%        N = Number of nodes for spatial discretization in one
   dimension
%        a = "left" end point of side of square
%        b = "left" end point of side of square
%        BC_type = type of Boundary condition on the boundary of
   domain
% Outputs:
%        tout = vector of times solution is approximated at
%        xout,yout = spatial discretization mesh points
%        u = solution at (x,y) at time = t
%        con = condition number of the matrix to be solved
%
% Note: initial conditions and BCs applied in a subroutine
%
%       cheb.m function is from the text Spectral Methods in
   Matlab by
%       Trefethen. Method for obtaining interior nodes are
   adapted from
%       poisson_spec code given

% Get chebyshev differentiation matrix for interior points
[D,x] = cheb(N-1);
x = x(end:-1:1);
% map to interval of interest
x = (b-a)/2*x+(b+a)/2;
J = (b-a)/2; % jacobian
D = 1/J*D; %derivative operator on interval defined by (a,b)

% second derivative differentiation matrix
D2 = D*D; % 1D second derivative matrix

% Create tensor product grid
y = x;
[xx,yy] = meshgrid(x,y);
xx = xx(:);
yy = yy(:);

% Derivative approximations of the left hand side
du_dydy = kron(eye(N),D2);
du_dxdx = kron(D2,eye(N));
du_dy =   -kron(eye(N),D);
du_dx =   -kron(D,eye(N));
```

```
% Determine interior vs boundary points
logic_idx = xx==a|xx==b|yy==a|yy==b;
Ext_pt = find(logic_idx);
Int_pt = find(~logic_idx);

% Order exterior points so that it goes from bottom left corner
   and moves
% counterclockwise
xcenter = (b+a)/2;
ycenter = (b+a)/2;

% reference angle to determine point locations relative to center
theta0 = atan2(yy(1)-ycenter,xx(1)-xcenter);
theta = rem(4*pi+1e-12-theta0+atan2(yy(Ext_pt)-ycenter,xx(Ext_pt)
   -xcenter),2*pi);
[~,order_ccw] = sort(theta);
Ext_pt = Ext_pt(order_ccw); % gives indexing so that points are
   ordered from bottom left and moves ccw

% Contributions of variable coefficients
kappa = diag(K(xx,yy));

% Spatial Discretization operator
A = du_dx*kappa*du_dx+du_dy*kappa*du_dy;
Aint = A(Int_pt,:); % only for the interior points

F = zeros(length(Ext_pt),N^2);
F(:,Ext_pt) = eye(length(Ext_pt));

%eigval = eig([F;Aint]);

% Flow: initialize t = 0, advance in time the interior nodes,
   apply BCs to
% boundary at each timestep
tout = t0:dt:tend;

u = zeros(length(xx),length(tout));
con = cond([F;Aint]);

u(:,1) = applyICs(xx,yy,Ext_pt,Int_pt,N);

% check_partial_deriv(xx,yy,du_dx,du_dy,Ext_pt,N,du_dxdx,du_dydy)
   ;

% Begin time stepping scheme (Euler)
```

```
for it = 2:length(tout)
    u(Int_pt,it) = u(Int_pt,it-1) + dt*(Aint*u(:,it-1)+f(xx(
       Int_pt),yy(Int_pt),tout(it-1)));
    % apply BCs to boundary nodes
    [~,u(Ext_pt,it)] = applyBCs(tout(it),xx,yy,du_dx,du_dy,Ext_pt
       ,N,BC_type);
end

% output grid
xout = x;
yout = y;
end

%% Initial Conditions
function u0 = applyICs(xx,yy,boundary,interior,N)
% Inputs: xx,yy = grid
%         boundary nodes indices
%        interior nodes indices
%        N = number of nodes
% Outputs: u0 initial value of the solution

% function to apply initial conditions

% IC = @(x,y) sin(y).*sin(x); % initial condition function for
   test problem
IC = @(x,y) sin(pi*x.*y);
nodeperside = length(boundary)/4;

% identify bottom, right, top, and left indices of square
b = boundary(1:N-1);
r = boundary(N:N+nodeperside-1);
t = boundary(N+nodeperside:N+2*nodeperside-1);
l = boundary(N+2*nodeperside:end);

% apply IC to nodes
u0(1:nodeperside,:) = IC(xx(b),yy(b));
u0(nodeperside+1:2*nodeperside,:) = IC(xx(r),yy(r));
u0(2*nodeperside+1:3*nodeperside,:) = IC(xx(t),yy(t));
u0(3*nodeperside+1:length(boundary),:) = IC(xx(l),yy(l));
u0(length(boundary)+1:length(boundary)+length(interior),:) = IC(
   xx(interior),yy(interior));

u0 = zeros(length(boundary)+length(interior),1);
u0(boundary) = IC(xx(boundary),yy(boundary));
u0(interior) = IC(xx(interior),yy(interior));
```

```
end

%% Boundary Conditions
function [F,fext] = applyBCs(time,xx,yy,dudx,dudy,Ext_pt,N,
   BC_type)
% Inputs:
%        xx,yy = 2D grid points flattened as long 1D vectors
%        dudx, dudy = partial derivative matrices
%        Ext_pt = exterior point indices
%        N = number of points on a side
%        BC_type = type of BC

Npt = length(xx);

% Matrices
switch BC_type
    case 'Dirichlet'

        F = zeros(length(Ext_pt),Npt);
        F(:,Ext_pt) = eye(length(Ext_pt));

        nodeperside = length(Ext_pt)/4;

        % indices for bottom, right, top, and left sides of the
           square
        b = Ext_pt(1:N-1);
        r = Ext_pt(N:N+nodeperside-1);
        t = Ext_pt(N+nodeperside:N+2*nodeperside-1);
        l = Ext_pt(N+2*nodeperside:end);

end

% Values of boundary condition (need to change depending on the
   problem you
% consider)
% fbot = zeros(length(b),1);
% fright = zeros(length(r),1);
% ftop = zeros(length(t),1);
% fleft = zeros(length(l),1);
fbot = -exp(-time^2)*sin(pi*xx(b));
fleft = -exp(-time^2)*sin(pi*yy(l));
ftop = exp(-time^2)*sin(pi*xx(t));
fright = exp(-time^2)*sin(pi*yy(r));

fext = [fbot;fright;ftop;fleft];
```

```
end

function check_partial_deriv(xx,yy,dudx,dudy,ext,N,dudxdx,dudydy)

s = ext(1:N);
e = ext(N+1:N+5);
n = ext(N+6:N+11);
w = ext(N+12:end);

u = @(x,y) exp(x).*sin(y);
ux = @(x,y) exp(x).*sin(y);
uy = @(x,y) exp(x).*cos(y);

Dsx = dudx(s,:);Dex = dudx(e,:);
Dnx = dudx(n,:);Dwx = dudx(w,:);

Dsy = dudy(s,:);Dey = dudy(e,:);
Dny = dudy(n,:);Dwy = dudy(w,:);

ff = u(xx,yy);

test{:,1} = Dsx*ff; exact{:,1} = ux(xx(s),yy(s));
test{:,2} = Dwx*ff; exact{:,2} = ux(xx(w),yy(w));
test{:,3} = Dnx*ff; exact{:,3} = ux(xx(n),yy(n));
test{:,4} = Dex*ff; exact{:,4} = ux(xx(e),yy(e));
test{:,5} = Dsy*ff; exact{:,5} = uy(xx(s),yy(s));
test{:,6} = Dwy*ff; exact{:,6} = uy(xx(w),yy(w));
test{:,7} = Dny*ff; exact{:,7} = uy(xx(n),yy(n));
test{:,8} = Dey*ff; exact{:,8} = uy(xx(e),yy(e));

for i = 1:length(test)
    err(i) = norm(exact{:,i}-test{:,i});
end

figure
plot(err)
% figure
% plot(ux_ex)
% hold on
% plot(test)
% err = norm(ux_ex-test);
disp(err)
end
```

E. Chebyshev Differentiation Matrix

```
% CHEB  compute D = differentiation matrix, x = Chebyshev grid

  function [D,x] = cheb(N)
  if N==0, D=0; x=1; return, end
  x = cos(pi*(0:N)/N)';
  c = [2; ones(N-1,1); 2].*(-1).^(0:N)';
  X = repmat(x,1,N+1);
  dX = X-X';
  D  = (c*(1./c)')./(dX+(eye(N+1)));      % off-diagonal entries
  D  = D - diag(sum(D'));                 % diagonal entries
```