Spring 5-2018

# Automatic Construction of Scalable Time-Stepping Methods for Stiff PDES

Vivian Ashley Montiforte
*University of Southern Mississippi*

# AUTOMATIC CONSTRUCTION OF SCALABLE TIME-STEPPING METHODS FOR STIFF PDES

by

Vivian Ashley Montiforte

A Thesis
Submitted to the Graduate School,
the College of Science and Technology,
and the Department of Mathematics
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

May 2018

AUTOMATIC CONSTRUCTION OF SCALABLE TIME-STEPPING METHODS FOR STIFF PDES

by Vivian Ashley Montiforte

May 2018

Approved by:

Dr. James Lambers, Committee Chair
Associate Professor, Mathematics

Dr. Haiyan Tian, Committee Member
Professor, Mathematics

Dr. Huiqing Zhu, Committee Member
Associate Professor, Mathematics

Dr. Bernd Schroeder
Chair, Department of Mathematics

Dr. Karen S. Coats
Dean of the Graduate School

ABSTRACT

# AUTOMATIC CONSTRUCTION OF SCALABLE TIME-STEPPING METHODS FOR STIFF PDES

by Vivian Ashley Montiforte

May 2018

Krylov Subspace Spectral (KSS) Methods have been demonstrated to be highly scalable time-stepping methods for stiff nonlinear PDEs. However, ensuring this scalability requires analytic computation of frequency-dependent quadrature nodes from the coefficients of the spatial differential operator. This thesis describes how this process can be automated for various classes of differential operators to facilitate public-domain software implementation.

# ACKNOWLEDGMENTS

I would first like to thank my advisor and professor, Dr. James Lambers. Without him, this research would have been impossible. I would also like to thank my committee members, Dr. Haiyan Tian and Dr. Huiqing Zhu, for allowing me their precious time and immense knowledge.

I owe my time here at the University of Southern Mississippi to my amazing husband, Kyle. He has helped make my educational and career goals a possibility and has always supported my dreams. Lastly, I would like to thank my son, Kooper, for making me want to become the best version of myself.

# TABLE OF CONTENTS

# LIST OF ILLUSTRATIONS

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| **ODE** | - | Ordinary Differential Equation |
| **PDE** | - | Partial Differential Equation |
| **KSS** | - | Krylov Subspace Spectral |
| **FFT** | - | Fast Fourier Transform |

# NOTATION AND GLOSSARY

## General Usage and Terminology

The notation used in this text represents fairly standard mathematical and computational usage. In many cases these fields tend to use different preferred notation to indicate the same concept, and these have been reconciled to the extent possible, given the interdisciplinary nature of the material. In particular, the notation for partial derivatives varies extensively, and the notation used is chosen for stylistic convenience based on the application. While it would be convenient to utilize a standard nomenclature for this important symbol, the many alternatives currently in the published literature will continue to be utilized.

The blackboard fonts are used to denote standard sets of numbers: $\mathbb{R}$ for the field of real numbers, $\mathbb{C}$ for the complex field, $\mathbb{Z}$ for the integers, and $\mathbb{Q}$ for the rationals. The capital letters, $A, B, \cdots$ are used to denote matrices, including capital Greek letters, e.g., $\Lambda$ for a diagonal matrix. Functions which are denoted in boldface type typically represent vector valued functions, and real valued functions usually are set in lower case Roman or Greek letters. Caligraphic letters, e.g., $\mathcal{V}$, are used to denote spaces such as $\mathcal{V}$ denoting a vector space, $\mathcal{H}$ denoting a Hilbert space, or $\mathcal{F}$ denoting a general function space. Lower case letters such as $i, j, k, l, m, n$ and sometimes $p$ and $d$ are used to denote indices.

Vectors are typeset in square brackets, e.g., $[\cdot]$, and matrices are typeset in parenthesese, e.g., $(\cdot)$. In general the norms are typeset using double pairs of lines, e.g., $||\cdot||$, and the absolute value of numbers is denoted using a single pairs of lines, e.g., $|\cdot|$. Single pairs of lines around matrices indicates the determinant of the matrix.

# Chapter 1

# INTRODUCTION

Advancements in computing power allow the solutions of mathematical models to be approximated at a much higher spatial resolution. While there is positive progress in approximating the solution, there are also downfalls. A greater stiffness occurs in some models that have a system of ordinary differential equations (ODEs) stemming from the spatial discretization of time-dependent partial differential equations (PDEs). This stiffness presents a challenge when applying both implicit and explicit time-stepping methods. The challenge when using an implicit method comes from solving increasingly ill-conditioned systems of equations. These systems of equations increase the number of iterations needed. When using an explicit method, the challenge arises from the necessary CFL condition. The CFL condition forces this method to use a smaller time-step in order to have convergence.

Consider a spatially discretized parabolic PDE; the solution is approximated by finding the exact solution of the system of ODEs. Computing the exact solution of this system of ODEs requires computing a matrix function and vector product. The use of time stepping methods is prevalent because they can approximate matrix function and vector products. This product is approximated by a polynomial function, in the case of an explicit method, or a rational function, in the case of an implicit method.

A previous approach, Krylov Projection [5] used Arnoldi or Lanczos iterations to compute a polynomial that was then used to approximate the exponential function. However, due to the character of time-stepping methods, this approach was not practical. The use of this polynomial to approximate the exponential function on such a large interval required a large number of terms. Also, due to the significantly varying eigenvalues, approximating this matrix exponential function required a great deal of Lanczos vectors. The varying of these eigenvalues occurs in the instance where $A$ comes from a stiff system of equations, such as the system of ODEs previously discussed. For these reasons, this approach is only sensible when the eigenvalues are clustered and, therefore, not suitable for our methods.

An alternative approach uses a component-wise approximation. This approach approximates high frequency and low frequency components of the solution individually. Separating the frequency components allows the approximating polynomials to be tailored to best suit the individual components. From this alternative technique, block Krylov Subspace Spectral

(KSS) methods emerged [10]. These block KSS methods are favorable due to explicit time stepping with high-order accuracy. Explicit methods generally need a small time-step to ensure stability but previous, simple KSS methods have been proven to be unconditionally stable. These previous KSS methods generated results that were favorable in accuracy and stability. Unfortunately, these results were obtained using a highly, computationally expensive algorithm. This algorithm uses a different initial block in the block Lanczos iteration for each Fourier component of the solution. Previous work has been completed in [9] proving that due to the wave number $\omega$ being used to parameterize the initial blocks, there is much redundant computation that can be eliminated. This resulted in an algorithm that requires $\mathcal{O}(N \log N)$ operations per time step.

The majority of the computational expense emerges from computing block Gaussian quadrature with component-dependent nodes and weights. A previous KSS method saved some computational expense by using estimates of the extremal nodes from block Gaussian to acquire the quadrature nodes [7]. These extremal nodes are acquired from an asymptotic analysis of the recursions coefficients created by block Lanczos. The advantage of this method is that these nodes are computed quite rapidly since we are not actually carrying out the block Lanczos iteration for any Fourier component. This method is much faster compared to the previous approach that obtained the nodes by computing the eigenvalues of each block tridiagonal matrix produced by the block Lanczos iteration. Although this approach of using the extremal nodes improves the efficiency, it does not, however, preserve the same accuracy as the previous block KSS methods. Accuracy is not preserved in this method due to the extremal nodes for each Fourier component being the only precise approximations of the block Gaussian quadrature nodes.

In a recent paper, a more thorough asymptotic analysis of the recursion coefficients was performed to provide a new insight into the computation of the block Gaussian nodes [14]. This new insight provided a simple method that made block KSS methods a drastically more efficient implementation. During the analysis of the recursion coefficients, it was discovered that it could be used to show how eigenvalue problems for computing the quadrature nodes approximately decoupled for high frequencies. Decoupling of the eigenvalue problem was then exploited to rapidly estimate the nodes. Unlike the previous approach, this new approach preserved the accuracy of the original KSS methods. The preservation of accuracy comes from all nodes for each Fourier component being accurately estimated instead of only the extremal nodes. This KSS approach was compared against traditional Krylov subspace-based approaches and, when accelerated in this way, the KSS approach exhibited highly favorable behavior. This behavior was not only advantageous in terms of accuracy and stability, but was also scalable with respect to the amount of grid points used when spatially discretizing our underling PDE.

In prior work, it was shown that when using the block Lanczos algorithm to produce the Gaussian quadrature rules, half of the nodes were frequency independent and the other half were strongly frequency-dependent [11, 14]. Due to using a self-adjoint differential operator, the frequency-independent nodes are acquired by performing the symmetric Lanczos iteration with the initial vector being the solution from the previous time-step. The output of this algorithm is a symmetric tridiagonal matrix which we then take the eigenvalues of. These eigenvalues are our frequency-independent nodes. In this paper, we introduce a simpler, yet still efficient, method for computing the frequency-dependent Gaussian quadrature nodes. In this new approach, we choose only a few frequencies, three to five frequencies per spatial dimension, that are roughly equally spaced and use them as our interpolation points. True Gaussian quadrature nodes are computed from these few selected frequencies, which then allows us to use polynomial interpolation to approximate the remaining frequency-dependent nodes.

The outline of this thesis is as follows. Section 2 provides background information on block KSS methods. In Section 3, the idea and implementation of this new approach will be thoroughly described. Section 4 consists of numerical results that compare the accuracy and efficiency to previous KSS methods. Lastly, Section 5 holds the conclusion and ideas for future research.

# Chapter 2

# BACKGROUND

## 2.1 Krylov Subspace Spectral Methods

The parabolic PDE on the interval (0, $2\pi$) is considered,

$$u_t + Lu = 0, \ t > 0, \tag{2.1}$$

with initial data, $\mathbf{u}(x,0) = \mathbf{u}_0(x)$. $L$ is a self-adjoint, positive definite, second-order differential operator. Using the spatial discretization of the parabolic PDE (2.1), a system of ODEs can be created. The resulting system of ODEs is as follows,

$$\mathbf{u}'(t) + A\mathbf{u} = 0,$$

with the initial condition, $\mathbf{u}(t_0) = \mathbf{u}_0$, coming from the initial data of the PDE. Both $\mathbf{u}(t)$ and $\mathbf{u}_0$ are N-vectors, and $A$ is an $N \times N$ matrix. The solution of the PDE can be approximated by solving the previous system of ODEs, where $\mathbf{u}(t) = e^{-At}\mathbf{u}_0$. After applying suitable initial conditions and periodic boundary conditions, the solution is represented using a Fourier series

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-\infty}^{\infty} e^{i\omega x} \hat{u}(\omega,t),$$

where the Fourier coefficients, $\hat{u}(\omega,t)$, are given by

$$\hat{u}(\omega,t) = \left\langle \frac{1}{\sqrt{2\pi}} e^{i\omega x}, u(x,t) \right\rangle = \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} e^{-i\omega x} u(x,t) \, dx$$

and the wave number of each Fourier component is represented by $\omega$. Note that $\langle ., . \rangle$ signifies the standard inner product of functions on $(0,2\pi)$.

The general approach of KSS methods is to approximate each Fourier coefficient independently of each other. This approach allows a different approximation of the solution operator $e^{-L\Delta t}$ that is tailored to best suit each Fourier coefficient of the solution. The computed solution $u(x,t_n)$ at time $t_n = n\Delta t$ is given. Using the previous time-step, the Fourier coefficients of the solution can be computed at time $t_{n+1}$. Each Fourier coefficient is given by

$$\hat{u}(\omega,t_{n+1}) = \left\langle \frac{1}{\sqrt{2\pi}} e^{i\omega x}, \exp[-L\Delta t] u(x,t_n) \right\rangle.$$

Spatial discretization of each Fourier coefficient at time $t_{n+1}$ yields a bilinear form of the form

$$\mathbf{u}^H f(A)\mathbf{v},$$

where $\mathbf{u}$ and $\mathbf{v}$ are $N$-vectors on a uniform $N$-point grid. The vector $\mathbf{u}$ consists of the values of $\frac{1}{\sqrt{2\pi}}e^{i\omega x}$ and the vector $\mathbf{v}$ consists of the values of $u(x,t_n)$. Also, $f(\lambda) = \exp(-\lambda\Delta t)$ and the matrix $A$ comes from discretizing the operator $L$, where $A = L_N$ is an $N \times N$ symmetric positive definite matrix. This matrix $A$ has real positive eigenvalues

$$b = \mu_1 \geq \mu_2 \geq \cdots \geq \mu_N = a > 0,$$

and associated orthonormal eigenvectors $\mathbf{q}_j$, where $j = 1,\ldots,N$. As a result, $\mathbf{u}^H f(A)\mathbf{v}$ can be rewritten in terms of its spectral decomposition,

$$\mathbf{u}^H f(A)\mathbf{v} = \sum_{j=1}^{N} f(\mu_j)\mathbf{u}^H\mathbf{q}_j\mathbf{q}_j^H\mathbf{v}.$$

Previous research from Golub and Meurant [2, 3] allow us to express the bilinear form as a Reimann-Stieltjes integral

$$\mathbf{u}^H f(A)\mathbf{v} = I[f] = \int_a^b f(\lambda)\,d\alpha(\lambda).$$

where

$$\alpha(\lambda) = \begin{cases} 0, & \text{if } \lambda < a \\ \sum_{j=i}^{N} \alpha_j\beta_j, & \text{if } \mu_i \leq \lambda < \mu_{i-1}, \; \alpha_j = \mathbf{u}^H\mathbf{q}_j, \; \beta_j = \mathbf{q}_j^H\mathbf{v}. \\ \sum_{j=i}^{N} \alpha_j\beta_j, & \text{if } b \leq \lambda \end{cases}$$

Gaussian quadrature is used to approximate $I[f]$, resulting in the following form

$$I[f] = \sum_{j=1}^{K} \omega_j f(\lambda_j) + R[f].$$

The nodes $\lambda_j$, $j = 1,\ldots,K$, and weights $\omega_j$, $j = 1,\ldots,K$, are obtained by using the Lanczos algorithm. When $\mathbf{u}$ and $\mathbf{v}$ are real vectors, this Gaussian quadrature rule is exact for polynomials up to degree $2K-1$. It can then be generalized to the complex case with the appropriate complex conjugation.

Now, the case where $\mathbf{u} \neq \mathbf{v}$ is considered. In this case, the weights are typically not positive real numbers. As a result, the quadrature rule can numerically destabilize [1]. As another option, the block approach is considered [4] ,

$$[\; \mathbf{u} \;\; \mathbf{v} \;]^H f(A) [\; \mathbf{u} \;\; \mathbf{v} \;].$$

The nodes and weights needed for the quadrature rule are acquired by applying the block Lanczos algorithm:

```
$X_0 = 0, X_1 = [\ \mathbf{u}\ \ \mathbf{v}\ ]$ (QR factorization)
for $j = 1, 2, \ldots, K$
    $V = AX_j$
    $M_j = X_j^H V$
    if $j < K$
        $R_j = V - X_{j-1}B_{j-1}^H - X_j M_j$
        $R_j = X_{j+1}B_j$ (QR factorization)
    end
end
```

The outcome of executing the block Lanczos algorithm is two $2 \times 2$ matrices, $M_j$ and $B_j$, where $B_j$ is upper triangular. Together these matrices form the block tridiagonal matrix, $T_K$:

$$T_K = \begin{bmatrix} M_1 & B_1^H & & & \\ B_1 & M_2 & B_2^H & & \\ & \ddots & \ddots & \ddots & \\ & & B_{K-2} & M_{K-1} & B_{K-1}^H \\ & & & B_{K-1} & M_K \end{bmatrix}, \tag{2.2}$$

This matrix can be viewed as a matrix-valued Riemann Stieltjes integral

$$\int_a^b f(\lambda)\, d\mu(\lambda) = \begin{bmatrix} \mathbf{u}^H f(A)\mathbf{u} & \mathbf{u}^H f(A)\mathbf{v} \\ \mathbf{v}^H f(A)\mathbf{u} & \mathbf{v}^H f(A)\mathbf{v} \end{bmatrix}.$$

Let the following equations be defined as

$$E_{12} = [\ \mathbf{e}_1\ \ \mathbf{e}_2\ ],$$

$$X_1 = [\ \mathbf{u}\ \ \mathbf{v}\ ].$$

Then, use the block approach,

$$\begin{aligned} &[\ \mathbf{u}\ \ \mathbf{v}\ ]^H f(A)[\ \mathbf{u}\ \ \mathbf{v}\ ] \\ &= X_1^H f(A) X_1 \\ &= E_{12}^H \underline{X}^H f(A) \underline{X} E_{12} \\ &= E_{12}^H f(\underline{X}^H A \underline{X}) E_{12} + \text{error} \end{aligned} \tag{2.3}$$

where

$$\begin{aligned} T_K &= \underline{\overline{X}}^H A \underline{\overline{X}} \\ T_K &= U_K \Lambda_K U_K^H \\ T_K &= \sum_{j=1}^{2K} \lambda_j \mathbf{u}_j \mathbf{u}_j^H \\ f(T_K) &= \sum_{j=1}^{2K} f(\lambda_j) \mathbf{u}_j \mathbf{u}_j^H \end{aligned}$$

It is important to note that $\lambda_j$ is a scalar and by using $f(T_K)$, (2.3) can be rewritten as

$$E_{12}^H f(T_K) E_{12} + \text{error} = \sum_{j=1}^{2K} f(\lambda_j) E_{12}^H \mathbf{u}_j \mathbf{u}_j^H E_{12} + \text{error}$$

After setting $\mathbf{v}_j = E_{12}^H \mathbf{u}_j$, where $\mathbf{v}_j$ is a 2-vector, the following quadrature formula is obtained.

$$\int_a^b f(\lambda)\, d\mu(\lambda) = \sum_{j=1}^{2K} f(\lambda_j) \mathbf{v}_j \mathbf{v}_j^H + \text{error}\,. \tag{2.4}$$

The block tridiagonal matrix, $T_K$, produces the nodes and the weights required from (2.4). The nodes are $\lambda_j$ and consist of the eigenvalues of $T_K$. The "weights" are the 2 x 2 matrices $\mathbf{v}_j \mathbf{v}_j^H$, where $\mathbf{v}_j$ is a 2-vector containing the first two components of each eigenvector of $T_K$.

The block KSS method for the parabolic PDE (2.1) starts by first, defining

$$R_0 = [\ \hat{\mathbf{e}}_\omega \quad \mathbf{u}^n\ ],$$

where the first column in the matrix, $\hat{\mathbf{e}}_\omega$, is a discretization of $\frac{1}{\sqrt{2\pi}} e^{i\omega x}$ and the second column, $\mathbf{u}^n$, is a discretization of the approximate solution $u(x,t)$ at time $t_n = n\Delta t$. The second step is to compute the QR Factorization of $R_0$,

$$R_0 = X_1(\omega) B_0(\omega)$$

which then gives the output

$$X_1(\omega) = [\ \hat{\mathbf{e}}_\omega \quad \frac{\mathbf{u}_\omega^n}{\|\mathbf{u}_\omega^n\|_2}\ ]$$

and

$$B_0(\omega) = \begin{bmatrix} 1 & \hat{\mathbf{e}}_\omega^H \mathbf{u}^n \\ 0 & \|\mathbf{u}_\omega^n\|_2 \end{bmatrix},$$

where

$$\mathbf{u}_\omega^n = \mathbf{u}^n - \hat{\mathbf{e}}_\omega \hat{\mathbf{e}}_\omega^H \mathbf{u}^n = \mathbf{u}^n - \hat{\mathbf{e}}_\omega \hat{u}(\omega, t_n). \tag{2.5}$$

From this, the next step is to apply the block Lanczos algorithm to the matrix $L_N$ with initial block $X_1(\omega)$. The matrix $L_N$ comes from the discretization of $L$. This algorithm constructs a block tridiagonal matrix $T_K$ (2.2). Every entry of $T_K$ is a function of $\omega$. After that, at time $t_{n+1}$, each Fourier coefficient of the solution is

$$[\hat{\mathbf{u}}^{n+1}]_\omega = [B_0(\omega)^H E_{12}^H \exp[-T_K(\omega)\Delta t] E_{12} B_0(\omega)]_{12}.$$

## 2.2 Asymptotic Analysis of Lanczos Iteration

The main idea behind KSS methods is computing each Fourier component of the solution independently. This is done by using an approximation that is tailored to best suit the Fourier component. From this, every component has a polynomial approximation that is best suited for that specific component. These polynomials are approximations of $S(L_N;\Delta t)$. The function $S$ is determined by the solution operator of the given PDE. In the case of the previously discussed PDE (2.1), the function $S(L_N;\Delta t) = e^{-L_N \Delta t}$. $L_N$ comes from the spatial discretization of the differential operator. The polynomial approximations are acquired from interpolating the function $S(\lambda;\Delta t)$. This interpolation is executed at the nodes chosen for each Fourier component. The form of the computed solution in Fourier space is as follows,

$$\mathbf{u}^{n+1} = S(L_N;\Delta t)\mathbf{u}^n = \sum_{j=0}^{2K} D_j(\Delta t) A^j \mathbf{u}^n, \tag{2.6}$$

where $D_j(\Delta t)$ is a diagonal matrix. Each row of this matrix corresponds to a specific component and the diagonal entries represent the coefficients of the interpolating polynomials. This section covers previous work demonstrating a much faster way of computing interpolation points. This approach was discovered while studying the block Lanczos behavior when $\omega$ is the wave number and the limit of $|\omega| \to \infty$.

### 2.2.1 The Block Case

In KSS Methods, $R_0 = [\ \hat{\mathbf{e}}_\omega \quad \mathbf{u}^n\ ]$ is used as the initial block for each $\omega = \frac{-N}{2}+1,\dots,\frac{N}{2}$. The vector $\mathbf{u}^n$, defined in the previous section, is the discretization of the approximate solution on a uniform $N$-point grid. The approximate solution is $u(x,t)$ at time $t_n = n\Delta t$. The first block Lanczos iteration starts with the $QR$-factorization of $R_0$:

$$R_0 = X_1 B_0, \tag{2.7}$$

where

$$X_1 = [\ \hat{\mathbf{e}}_\omega \quad \frac{\mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2}\ ] \text{ and } B_0 = \begin{bmatrix} 1 & \hat{u}(\omega,t_n) \\ 0 & \|\mathbf{u}^n_\omega\|_2 \end{bmatrix} \tag{2.8}$$

The vector $\mathbf{u}^n_\omega$ is defined from (2.5). The statement is made that as $|\omega| \to \infty$ , $|\hat{\mathbf{u}}(\omega)|^n \to 0$, if the solution $\mathbf{u}$ is continuous. From the previous statement, $B_0$ converges to a diagonal matrix.

The next step is to compute

$$M_1 = X_1^H L_N X_1, \tag{2.9}$$

The matrix $L_N$ is the spectral discretization of $L$, where $L$ is the operator defined by $L_u = pu_{xx} + q(x)u$. By substituting the value of $X_1$ from (2.8) into (2.9), the following is obtained

$$M_1 = \begin{bmatrix} \omega^2 p + \bar{q} & \frac{\widehat{L_N \mathbf{u}^n_\omega}(\omega)}{\|\mathbf{u}^n_\omega\|_2} \\ \frac{\overline{\widehat{L_N \mathbf{u}^n_\omega}(\omega)}}{\|\mathbf{u}^n_\omega\|_2} & R(L_N, \mathbf{u}^n_\omega) \end{bmatrix}.$$

Previous expressions are defined as: $\bar{q}$ is the mean of $q(x)$ on $(0, 2\pi)$. $\widehat{L_N \mathbf{u}^n_\omega}(\omega) = \hat{\mathbf{e}}^H_\omega L_N \mathbf{u}^n_\omega$ is the Fourier coefficient, of the gridfunction $L_N \mathbf{u}^n_\omega$, associated to the wave number $\omega$. Lastly, $R(L_N, \mathbf{u}^n_\omega) = \frac{\langle \mathbf{u}^n_\omega, L_N \mathbf{u}^n_\omega \rangle}{\langle \mathbf{u}^n_\omega, \mathbf{u}^n_\omega \rangle}$ is the Rayleigh quotient of $L_N$ and $\mathbf{u}^n_\omega$.

It was previously discussed that if we have a continuous function, then the Fourier coefficients go to zero as $|\omega|$ increases. With this, the non-diagonal entries of $M_1$ become negligible as long as our solution is sufficiently regular. Therefore,

$$M_1 \approx \begin{bmatrix} \omega^2 p + \bar{q} & 0 \\ 0 & R(L_N, \mathbf{u}^n) \end{bmatrix}.$$

Since it is known that the Fourier coefficients go to zero, they can be neglected. Terms that are of lower order in $\omega$ can also be neglected. Proceeding with the iteration results in the following,

$$R_1 = L_N X_1 - X_1 M_1 \approx [\ \tilde{\mathbf{q}} \hat{\mathbf{e}}_\omega \quad \frac{L_N \mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2} - R(L_N, \mathbf{u}^n_\omega \frac{\mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2}\ ],$$

where multiplication of vectors is component-wise. Also, $\mathbf{q}$ is a vector comprised of the values of $q(x)$ at the grid points ($\tilde{\mathbf{q}} = \mathbf{q} - \bar{q}$).

This process was continued in the paper from [14]. In this high-frequency limit, it was discovered that the block tridiagonal matrix $T_K$, produced by applying block Lanczos to $R_0$ (2.7), converges to a simpler matrix. This simpler matrix is obtained from first, applying the "non-block" Lanczos iteration to the columns of $R_0$ separately and second, alternating the columns and rows of the tridiagonal matrices produced from these iterations. Since $T_K$ converges to this previously discussed simpler matrix, the columns and rows of $T_K$ can be reordered. They are reordered to group together the even-numbered and odd-numbered columns and rows. With this reordering, the eigenvalue problem for this matrix was found to approximately decouple. The block Gaussian quadrature nodes can be obtained by computing the eigenvalues of these smaller, tridiagonal matrices. For an $\omega$ that is finite, this non-block Lanczos algorithm can be used to, at the minimum, estimate the true block Gaussian quadrature nodes.

### 2.2.2 The Non-Block Case

Previous discussion revealed that due to the matrix decoupling, approximations of half of the block Gaussian quadrature nodes are acquired by executing the "non-block" Lanczos iteration. This iteration is on the matrix $L_N$ with initial vector $\mathbf{u}$, the computed solution. This example of Krylov Subspace Spectral Methods is defined in [11, 14]. These frequency-dependent nodes are estimated by performing an asymptotic analysis of the Lanczos iteration. This iteration is on the matrix $L_N$ with initial vector $\hat{\mathbf{e}}_\omega$. This Lanczos iteration algorithm is given below:

```
$B_0 = 0$, $\mathbf{m}_0 = 0$, $\mathbf{m}_1 = \mathbf{u}/\|\mathbf{u}\|_2$
**for** $j = 1, 2, \dots, K$
    $\mathbf{v}_j = A\mathbf{m}_j$
    $\alpha_j = \mathbf{m}_j^H \mathbf{v}_j$
    **if** $j < K$
        $\mathbf{v}_j = \mathbf{v}_j - \beta_{j-1}\mathbf{m}_{j-1} - \alpha_j \mathbf{m}_j$
        $\beta_j = \|\mathbf{v}_j\|_2$
        $\mathbf{m}_{j+1} = \mathbf{v}_j/\beta_j$
    **end**
**end**
```

The case being considered is when $p$ is a constant. After carrying out three iterations and neglecting lower-order terms, the following recursion coefficients are obtained. It is also worth noting that by carrying out three iterations, it corresponds to a fifth-order accurate KSS method for parabolic PDEs.

$$\begin{bmatrix} \alpha_1 & \overline{\beta_1} & 0 \\ \beta_1 & \alpha_2 & \overline{\beta_2} \\ 0 & \beta_2 & \alpha_3 \end{bmatrix} \approx \begin{bmatrix} p\omega^2 & \|\tilde{\mathbf{q}}\|_2 & 0 \\ \|\tilde{\mathbf{q}}\|_2 & p\omega^2 & 2p|\omega|\|\mathbf{q}_x\|_2/\|\tilde{\mathbf{q}}\|_2 \\ 0 & 2p|\omega|\|\mathbf{q}_x\|_2/\|\tilde{\mathbf{q}}\|_2 & p\omega^2 \end{bmatrix}.$$

The frequency-dependent nodes can then be estimated easily and efficiently as

$$\lambda_{1,\omega} = p\omega^2,\ \lambda_{2,\omega}, \lambda_{3,\omega} = p\omega^2 \pm \sqrt{(\beta_1^2 + \beta_2^2)}. \tag{2.10}$$

# Chapter 3

# FREQUENCY-DEPENDENT NODES BY INTERPOLATION

## 3.1 One-Dimensional Parabolic PDE

We are going to use polynomial interpolation to accurately approximate the frequency-dependent nodes for PDEs. We will be interpolating from $\omega = \frac{-N}{2}+1,\ldots,\frac{N}{2}$. We choose a few $\omega$ that are equally spaced in our interval of $[\frac{-N}{2}+1,\frac{N}{2}]$. We used three $\omega$ values in Figure 3.1 and five $\omega$ values in Figure 4.1. We want to make sure that we choose $\omega$ values to cover all frequencies from low to medium to high. We know that $\omega$ spans $\frac{-N}{2}+1,\ldots,\frac{N}{2}$ and so we choose the smallest and largest $\omega$ and choose an $\omega$ that is between.

After our $\omega$ have been chosen, we carry out the non-block Lanczos algorithm for each $\omega$. This algorithm produces a tridiagonal matrix $T_K$ for each case of $\omega$. We then compute the eigenvalues of each tridiagonal matrix $T_K$ produced for each $\omega$. To get the frequency dependent nodes for all other $\omega$ in between the specifically chosen $\omega$, we use polynomial interpolation. Our frequency-dependent nodes are then stored in our matrix $nf$.

From (2.10), we determined that the eigenvalues, $\lambda_{1,\omega}$, would have the dominant term of $p\omega^2$. The dominant term was concluded from $B_1$ not depending on $\omega$ and $B_2$ having only a linear term of $\omega$. This leaves the highest degree of our eigenvalues similar to that of a quadratic.

Using this information, our first attempt to approximate our curve was using a second degree polynomial interpolation. For this case, we chose three values of $\omega$, and then interpolated to approximate the frequency dependent nodes for the rest of the $\omega$ in between. The results from this first attempt are shown in Figure 3.1.

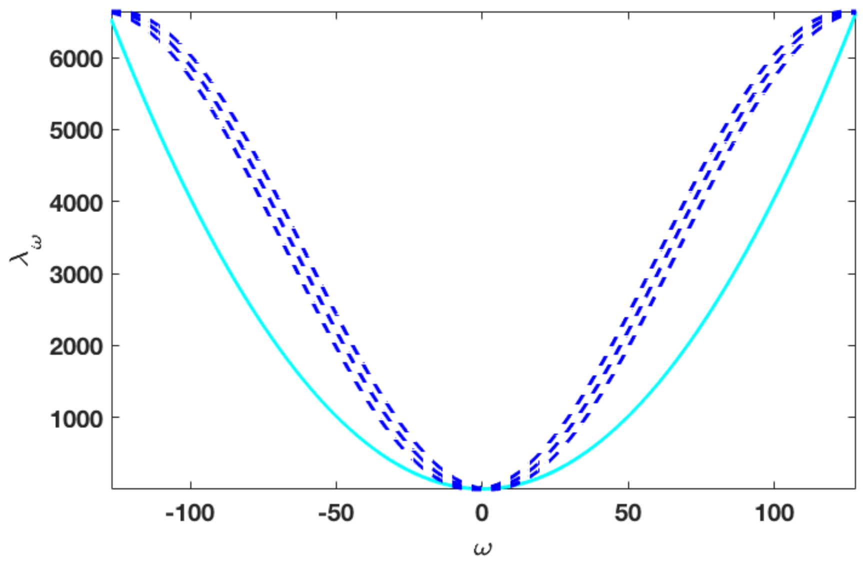

*Figure 3.1*: *First attempt using a second degree polynomial interpolation with three chosen $\omega$. The blue dashed curves represent all $\omega$ from $[\frac{-N}{2}+1, \frac{N}{2}]$ evaluated using the Lanczos algorithm. The cyan curves represent our method of evaluating only the three chosen $\omega$ in the Lanczos algorithm and using polynomial interpolation to approximate the curve.*

Looking at the graph above, it is easily noticed that our quadratic interpolant did not fit the curve as accurately as desired. Our second attempt starts with the fact that the matrix $L_N$ is a finite-difference representation of the differential operator. When this is true and formulas for the eigenvalues of symmetric Toeplitz tridiagonal matrices are used for the leading-order terms in the nodes, the block Gaussian quadrature nodes can be represented in a more accurate manner. Realizing the eigenvalues were behaving in this form, we used this direction and replaced $p\omega^2$ with $2p(N/\pi)^2(1-\cos(\pi\omega/N))$ where $N$ is the number of grid points.

Now that we have a new representation for our eigenvalues, we try interpolating again. This time, we need to use a higher degree polynomial interpolation. Using a higher degree differs from the previous approach by choosing five $\omega$ values to approximate versus the previous three $\omega$ values. Again we interpolate to approximate the frequency-dependent nodes for the rest of the $\omega$ in between the five chosen $\omega$. The results of this fourth degree interpolation are shown in the next chapter in Figure 4.1.

Algorithm for One-Dimensional Parabolic PDEs with Periodic Boundary Conditions:

It is important to note that $J$ is our Jacobian matrix, $m$ is the number of interpolation points, $K$ is the number of non-block Lanczos iterations, $\mathbf{x}$ contains the actual grid points, $N$ is the number of grid points, and our basis function is $e^{i\omega x}$.

```
1. Given that our $\omega$ range from $(-N/2+1,\dots,N/2)$, we choose $\omega$ values that are
   equally spaced, give or take, in the given interval.

2. We store the chosen $\omega$ values in the vector $\vec{\omega}$ and let $m$ represent the length of this
   vector.

3. **for** $i = 1,2,\dots,m$
       Run Lanczos algorithm for each chosen $\omega(i)$.
       Lanczos algorithm returns the tridiagonal matrix, $T_K$.
       Take the $K$ eigenvalues for each $T_K$.
       The eigenvalues are then stored in the rows of the matrix *mnodes* in increasing
           order, where each column is a different frequency.
   **end**

4. **for** $j = 1,2,\dots,K$
       Use fourth degree polynomial interpolation with vector $\vec{\omega}$ and rows of the
           matrix *mnodes*. In this case, we use the MATLAB command polyfit.
       Let $P_j$ represent the polynomial from the previous step.
       Evaluate $P_j$ for all $\omega$ from $(-N/2+1,\dots,N/2)$ to approximate the remaining
           frequency dependent nodes.
   **end**
```

Each time we use polynomial interpolation, we use the `polyfit` command included in MATLAB. `polyfit` computes the coefficients of the interpolant by solving a system of linear equations involving a Vandermonde matrix. In the future, we will try to minimize our computational expense by using Lagrange interpolation. Lagrange would be more efficient if we needed to interpolate more than once using the same points. Newton interpolation would have been more efficient during our first two attempts. If we had used Newton interpolation, then we could add more $\omega$ values without having to rerun previous $\omega$ values, saving time, computationally. Other cases to think about would be: 1. If the PDE is linear, then we can compute the nodes once and use them over again. 2. If the coefficients of the PDEs are not dependent on time, then we can compute the nodes once and reuse them for frequency dependent nodes. 3. If the frequency independent nodes tend to look the same or not changing much then we can reuse those to save on computational expense.

Summary of algorithm for computing $\mathbf{u}^{n+1}$ from $\mathbf{u}^n$ in One Space Dimension:

The vector $\mathbf{u}^n$ is a discretization of the approximate solution $u(x,t)$ at time $t_n = n\Delta t$ and $\mathbf{u}^{n+1}$ is described in (4.1) .

1. Perform $K$ iterations of the symmetric Lanczos algorithm on our matrix $L_N$, discretized from our spatial differential operator, and the initial vector $\mathbf{u}^n$. The Lanczos algorithm outputs the symmetric tridiagonal matrix, $T_K$, which is then used to compute the $K$ eigenvalues $\tilde{\lambda}_1, \dots, \tilde{\lambda}_K$. These $K$ eigenvalues are the frequency-independent nodes.

2. Compute polynomial interpolation with the few specifically chosen $\omega$ frequencies as our interpolation points. A fourth degree polynomial interpolation is used to approximate our curve. We then evaluate all $\omega$ values in our polynomial to approximate the remaining frequency-dependent nodes. These frequency-dependent nodes are stored in $\lambda_{j,\omega}$ where $j = 1, \dots, K$.

3. Combine the frequency-independent nodes, $\tilde{\lambda}_1, \dots, \tilde{\lambda}_K$, with the frequency-dependent nodes, $\lambda_{1,\omega}, \dots, \lambda_{K,\omega}$, for a total of $2K$ nodes that are used to compute the coefficients of our interpolating polynomials, $P_{2K-1,\omega}(\lambda)$ of degree $2K-1$. These polynomials will interpolate $e^{-\lambda\Delta t}$ at the nodes $\lambda_{0,\omega}, \dots, \lambda_{2K-1,\omega}$ for $\omega_1, \omega_2 = 1, \dots, N$.

   For $\omega = -N/2+1, \dots, N/2$,we have the following:

$$p_{2K-1,\omega}(\lambda) = \sum_{j=0}^{2K-1} c_{j,\omega}\lambda^j, \quad p_{2K-1,\omega}(\lambda_{j,\omega}) = e^{-\lambda_{j,\omega}\Delta t}$$

4. For $j = 0, 1, \dots, 2K-1$ let $C_j = \operatorname{diag}(c_{j,-N/2+1}, \dots, c_{j,N/2})$. Then compute $\mathbf{u}^{n+1}$ as follows

$$\mathbf{u}^{n+1} = \mathscr{F}_N^{-1} \sum_{j=0}^{2K-1} C_j \mathscr{F}_N L_N^j \mathbf{u}^n,$$

   where the matrix $\mathscr{F}$ performs an $N$-point Fast Fourier Transfer (FFT) in 1-D.

## 3.2 Two-Dimensional PDE

Using the previous approach, we expand the method to accurately approximate the frequency-dependent nodes of a two-dimensional PDE with Dirichlet boundary conditions. Since our boundary conditions are different, we will be interpolating from $\omega = 1, \dots, N$. We choose $\omega$ values that are equally spaced over this interval and include low, medium, and high frequencies. Generally, we choose the smallest and largest $\omega$ and then choose $\omega$ values that are equally spaced in between. These values are stored in the vector $\vec{\omega}$ and $m$ represents the length of this vector.

Since this is a two-dimensional approach, we need all possible 2-D grid coordinates. The `meshgrid` command in MATLAB produces all possible 2-D coordinate combinations which are then stored in $[\vec{\omega}_1, \vec{\omega}_2]$. Similar to the one-dimensional case, we run Lanczos algorithm, but now for each two-dimensional coordinate. The Lanczos algorithm will return the tridiagonal matrix, $T_\omega$ that we then take the eigenvalues of. These eigenvalues are stored in MATLAB as an $m \times m \times K$ array. This is similar to our matrix *mnodes* from the 1-D case but must be stored differently to account for both dimensions and to easily access the nodes for a given frequency. These eigenvalues are stored in increasing order and are used as our interpolation points.

The next step is to make all possible combinations of all $\omega$ values, not only the chosen ones. The vector $\vec{\omega s}$ contains all $\omega$ from $(1,\dots,N)$. The MATLAB command `meshgrid` is called with the $\vec{\omega s}$ and the 2-D coordinates are stored in $[\vec{\omega s}_1, \vec{\omega s}_2]$. Our frequency-dependent nodes corresponding to the frequencies in $[\vec{\omega s}_1, \vec{\omega s}_2]$ are stored in each $m \times m$ layer of *mnodes*. We use the 2-D polynomial interpolation command in MATLAB, `interp2`, with cubic spline interpolation. This outputs the values of the cubic spline at the interpolation points. Similar to our frequency-dependent nodes being stored in $nf$ in the 1-D case, their values at $[\omega s_1, \omega s_2]$ are stored in $nf$. The values at $[\omega s_1, \omega s_2]$ are our frequency-dependent nodes.

Algorithm for Two-Dimensional Parabolic PDEs with Dirichlet Boundary Conditions:

Similar to our one-dimensional algorithm, $J$ is our Jacobian matrix, $m$ is the number of interpolation points, $K$ is the number of non-block Lanczos iterations, and $N$ is the number of grid points per dimension. In addition, $\mathbf{x}$ and $\mathbf{y}$ contain the actual grid points and our basis function is now $\sin(\omega_1 x)\sin(\omega_2 y)$.

1. Given that our $\omega$ range from $(1,\dots,N)$, we choose $\omega$ values that are equally spaced, give or take, in the given interval.

2. We store the chosen $\omega$ values in the vector $\vec{\omega}$ and let $m$ represent the length of this vector.

3. Use the MATLAB command `meshgrid` with the vector $\vec{\omega}$ to produce all possible two-dimensional grid coordinates. These coordinates are stored in $[\vec{\omega}_1, \vec{\omega}_2]$.

4. **for** $i = 1,2,\dots,m$

```
        for $j = 1, 2, \ldots, m$
            Run Lanczos algorithm for each two-dimensional coordinate $[\vec{\omega}_1(i,j), \vec{\omega}_2(i,j)]$.
            Lanczos algorithm returns the tridiagonal matrix, $T_K$.
            Take the $K$ eigenvalues for each $T_K$.
            The eigenvalues are then stored in the matrix *mnodes* in increasing order.
        end
    end
```

5. Again, use the MATLAB command `meshgrid` with the vector $\vec{\omega s}$ containing all $\omega$ from $(0, \ldots, N)$ to produce all possible two-dimensional coordinates. These coordinates are stored in $[\vec{\omega s}_1, \vec{\omega s}_2]$.

```
6. for $j = 1, \ldots, K$
       Let $NodesJ$ equal our matrix $mnodes$ and reshape $NodesJ$ so that it is a $m \times m$
           two-dimensional matrix.
       Use a two-dimensional polynomial interpolation with $\vec{\omega}, \vec{\omega}, NodesJ, \vec{\omega s}_1$,
           and $\vec{\omega s}_2$. In this case, we used the MATLAB command interp2 with
           spline interpolation.
       The values of each polynomial at all of the frequencies are reshaped as a vector and stored in th
   end
```

Summary of algorithm for computing $\mathbf{u}^{n+1}$ from $\mathbf{u}^n$ in Two Space Dimensions:

The vector $\mathbf{u}^n$ is a discretization of the approximate solution $u(x,t)$ at time $t_n = n\Delta t$ and $\mathbf{u}^{n+1}$ is described in (4.1).

1. Perform $K$ iterations of the symmetric Lanczos algorithm on our matrix $L_N$, discretized from our spatial differential operator, and the initial vector $\mathbf{u}^n$. The Lanczos algorithm outputs the symmetric tridiagonal matrix, $T_K$, which is then used to compute the $K$ eigenvalues $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_K$. These $K$ eigenvalues are the frequency-independent nodes.

2. Use polynomial interpolation with the few specifically chosen $[\vec{\omega}_1, \vec{\omega}_2]$ as our interpolation points and the previously found eigenvalues of $T_K$ as our function values. A two-dimensional cubic spline polynomial interpolation is used to approximate our curve. We then evaluate all $[\vec{\omega s}_1, \vec{\omega s}_2]$ frequencies in our polynomial to approximate the remaining frequency dependent nodes.

3. Combining the frequency-independent nodes, $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_K$, with the frequency-dependent nodes, $\lambda_{1,\vec{\omega}}, \ldots, \lambda_{K,\vec{\omega}}$, we have a total of $2K$ nodes that are used to compute the coeffi-

cients of our interpolating polynomials, $P_{2K-1,\omega_1,\omega_2}(\lambda)$ of degree $2K-1$. These polynomials will interpolate $e^{-\lambda\Delta t}$ at the nodes $\lambda_{0,\vec{\omega}},\ldots,\lambda_{2K-1,\vec{\omega}}$ for $\omega_1,\omega_2 = 1,\ldots,N$. For $\omega_1,\omega_2 = 1,\ldots,N$,we have the following:

$$p_{2K-1,\omega_1,\omega_2}(\lambda) = \sum_{j=0}^{2K-1} c_{j,\omega_1,\omega_2}\lambda^j \quad , p_{2K-1,\omega_1,\omega_2}(\lambda_{j,\vec{\omega}}) = e^{-\lambda_{j,\vec{\omega}}\Delta t}$$

4. For $j = 0,1,\ldots,2K-1$ let $C_j = diag(c_{j,1,1},\ldots,c_{j,N,N})$. Then compute $\mathbf{u}^{n+1}$.

$$\mathbf{u}^{n+1} = \mathscr{F}_N^{-1}\sum_{j=0}^{2K-1} C_j\mathscr{F}_N L_N^j\mathbf{u}^n,$$

where the matrix $\mathscr{F}$ performs an $N^2$-point Fast Fourier Transfer (FFT) in 2-D.

# Chapter 4

# NUMERICAL RESULTS

## 4.1 One-Dimensional Parabolic PDE

Our first numerical result is demonstrating the effectiveness when solving the 1-D Parabolic PDE,

$$u_t = pu_{xx} + q(x)u, \quad 0 < x < 2\pi \tag{4.1}$$

with periodic boundary conditions,

$$u(0,t) = u(2\pi,t)$$

where

$$q(x) = 1 + \frac{1}{2}\sin(3x).$$

The number of grid points used in this case is 256, our $\omega$ values are between $-\frac{N}{2}+1,\ldots,\frac{N}{2}$, and our constant $p$ is 1. Using this information, we create the Jacobian matrix using a centered finite difference. The Jacobian matrix is then passed to our algorithm, along with the values of $x$ and the number of grid points. Our chosen five equally spaced $\omega$ values are $-\frac{N}{4}, -\frac{N}{2}, 1, \frac{N}{2}$, and $\frac{N}{4}$. We then run the Lanczos algorithm for each chosen $\omega$ which outputs the symmetric tridiagonal matrix, $T_K$. The eigenvalues of this matrix $T_K$ are the nodes to our interpolation points, $\omega$. We then use a fourth degree polynomial interpolation to approximate the remaining frequency-dependent nodes to complete our curve. In Figure 4.1, we compared our method to this previous approach from [6]. However, this previous approach is far more computationally expensive compared to our method. This alternative approach must execute the Lanczos algorithm for all frequencies which then outputs the symmetric tridiagonal matrix $T_K$. It must then compute the eigenvalues for each $T_K$. This is computed for all $\omega$ between $[-\frac{N}{2}+1,\ldots,\frac{N}{2}]$. Our method is far less computationally expensive yet still holds accuracy.

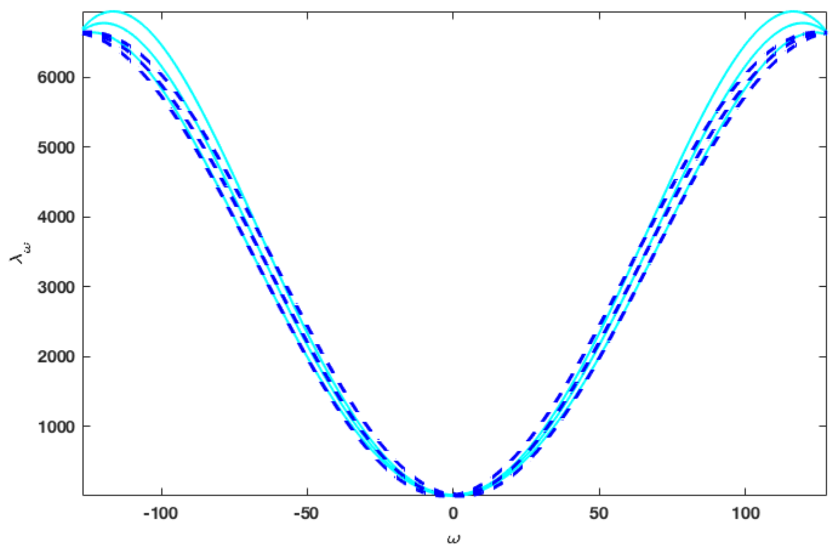

*Figure 4.1*: *One-Dimensional result using the parabolic PDE with periodic boundary conditions* (4.1). *The blue dashed curves represent all $\omega$ from $[1, \frac{N}{2}]$ evaluated in the Lanczos algorithm. The cyan curves represent our method of evaluating only five $\omega$ in the Lanczos algorithm and using polynomial interpolation to approximate the curve.*

## 4.2 Two-Dimensional PDE

Our second numerical result is with a 2-D PDE,

$$u_t = \Delta pu + q(x,y)u, \quad 0 < x < \pi$$

with Dirichlet boundary conditions,

$$u(x,0) = u(x,2\pi) = u(0,y) = u(2\pi,y) = 0$$

where

$$q(x) = 1 + \frac{1}{2}\sin(3x)\cos y.$$

Many of the steps are similar to the 1-D case but now we must account for the second dimension and different boundary conditions. The number of grid points used was 32, our $\omega$ values are between $[1,\ldots,N]$, and our constant $p$ is still 1. Since this is a 2-D case, we need 2-D coordinates. Using our x-values, we must use the MATLAB command `meshgrid` to produce all possible 2-D coordinate combinations. Our next step is to create our Jacobian matrix using a centered finite difference. This Jacobian is then passed to our 2-D algorithm described in the previous chapter, along with the $x$ values, $y$ values, and the number of grid points. Our chosen five $\omega$ in this case are $1, \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}$, and $N$. Again, we use the MATLAB command `meshgrid` to compute all possible 2-D coordinate combinations of our chosen $\omega$. The Lanczos algorithm is called with these coordinates and outputs a symmetric, tridiagonal matrix $T_K$ for all combinations of the chosen $\omega$. The eigenvalues of $T_K$ are our nodes for our chosen $\omega$. We then use the 2-D polynomial interpolation command in MATLAB, `interp2`, with the chosen coordinate values, the nodes computed from the eigenvalues of $T_K$, all possible 2-D coordinate combinations of all $\omega$ from $1,\ldots,N$, and specified cubic spline interpolation to compute the remaining frequency dependent nodes. It is important to note that all possible 2-D coordinate combinations of all $\omega$ were also produced by using the MATLAB command `meshgrid`.

Looking at the figures that are to follow, Figure 4.2 uses the alternative approach of computing the Lanczos algorithm for all possible 2-D coordinate combinations of all $\omega$ from $[1,\ldots,N]$ and then computing the eigenvalues for each $T_K$ produced from Lanczos. Figure 4.3 is our approach of computing the Lanczos algorithm for only the 2-D combinations of our chosen $\omega$ and then computing the eigenvalues of the significantly less $T_K$ matrices. Using the eigenvalues as nodes, we use polynomial interpolation to compute the remaining frequency dependent nodes. Much like our 1-D case, this 2-D example also showcases how far less computationally expensive our method is compared to the alternative method shown. Even with our method being less expensive, it stills holds the order of accuracy needed to be a competitive algorithm.

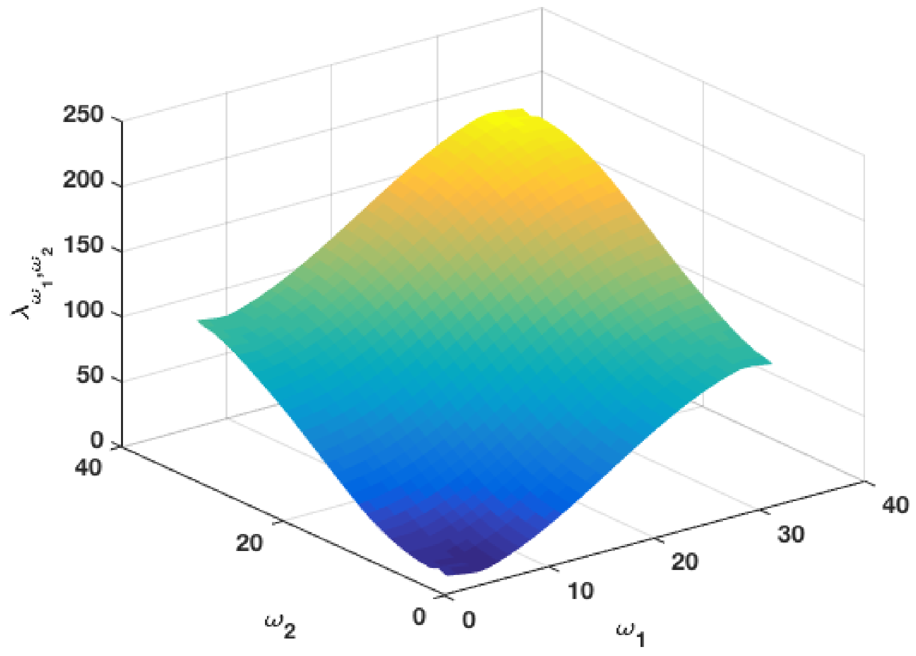

*Figure 4.2*: *Two-Dimensional PDE result using the Lanczos algorithm to evaluate all 2-D coordinates consisting of the possible combinations of* $\omega$ *between* $[1,N]$.

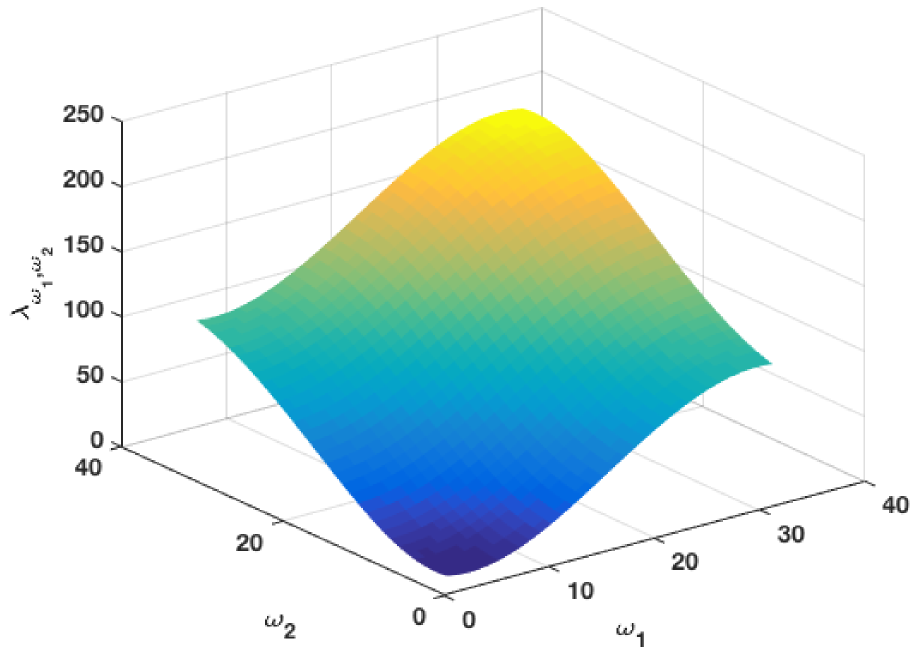

*Figure 4.3*: *Two-Dimensional PDE result using our method of choosing five* $\omega$ *values to evaluate using the Lanczos algorithm and using a 2-D polynomial interpolation to approximate the curve.*

## 4.3 Applying Our Method with KSS

After having success with a 1-D Parabolic PDE and a 2-D PDE, both having different boundary conditions, we decided to test our new approach within KSS methods. Below are the results from our first test case,

$$u_t = \alpha \Delta u + (1 - 3u_0^2)u, \quad 0 < t < 0.2 \tag{4.2}$$

on the rectangle $[0,1]^2$ with Neumann boundary conditions,

$$u_x(0,y) = u_x(1,y) = 0 \quad \text{and} \quad u_y(x,0) = u_y(x,1) = 0 \tag{4.3}$$

where

$$u_0(x,y) = 0.4 + 0.1\cos(2\pi x)\cos(5\pi y) \tag{4.4}$$

The steps for this case are very close to the 2-D case previously discussed above with a few differences. We use Neumann boundary conditions (4.3) and our basis function is now $\cos(\pi\omega_1 x)\cos(\pi\omega_2 y)$. This basis function also changes our interval of $\omega$ to $[0, N-1]$.

Figures 4.5 and 4.7 show results from our method using three $\omega$ frequencies with 50 grid points and 150 grid points per spatial dimension, respectively. Comparing 4.5 to Figure 4.4 and 4.7 to Figure 4.6, we can tell that our method captured the magnitude of the exact frequency-dependent nodes. However, it did not accurately represent the trend of the curves. From these results, we decided to change our chosen $\omega$ from three frequencies to five frequencies. Figures 4.9 and 4.11 show the results from our method using five $\omega$ with 50 grid points and 150 grid points per spatial dimension, respectively. Comparing 4.9 to Figure 4.8 and 4.11 to Figure 4.10, we can tell that our method again captured the magnitude of the exact frequency-dependent nodes; but in addition, using five $\omega$ frequencies also captured the trend of the curves. The slight discrepancy between these figures comes from the oversimplification of the formulas in Figures 4.8 and 4.10. The error Figures 4.12 and 4.13 compare the two approaches using five $\omega$ frequencies with 50 grid points and 150 grid points per spatial dimension, respectively, These figures compare relative error against execution time. These results show that our approach is slightly less efficient than the previous KSS method [14]; however, our method still holds the same order of accuracy. If the same error figures were computed using only three $\omega$ frequencies, as shown in 4.5 and 4.7, our method would be more efficient but less accurate.

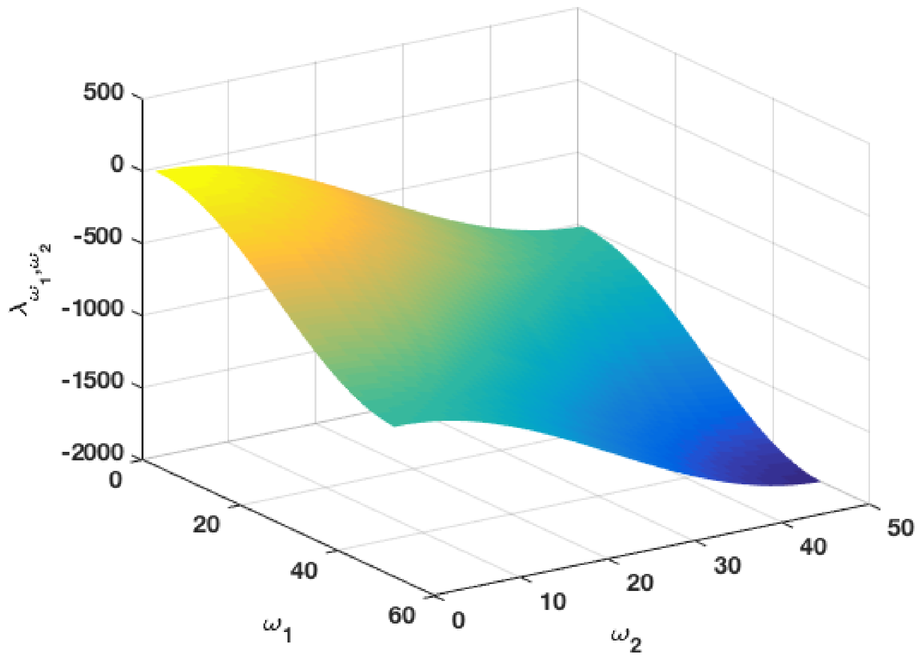

*Figure 4.4*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4)*. These results use the previous method for KSS [14] with three interpolation points and 50 grid points per dimension.*

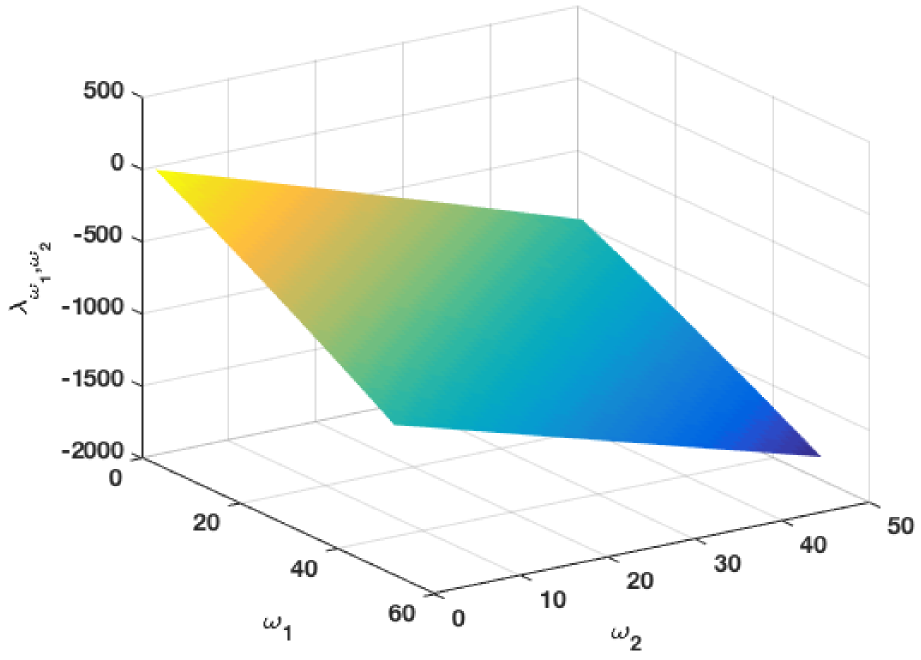

*Figure 4.5*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4)*. These results use our new method with three interpolation points and 50 grid points per dimension.*

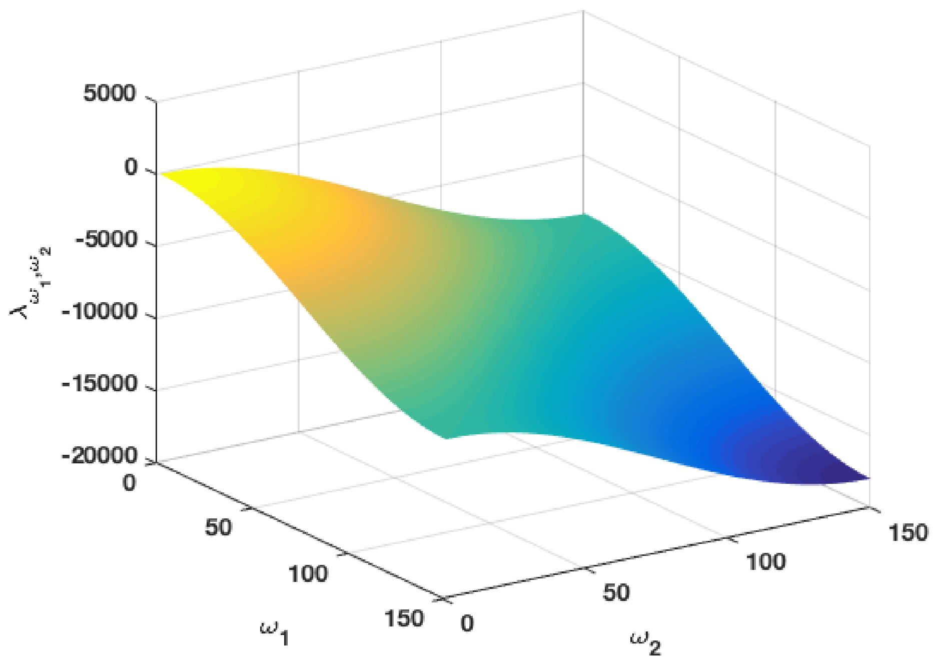

*Figure 4.6*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4). *These results use the previous method for KSS [14] with three interpolation points and 150 grid points per dimension.*

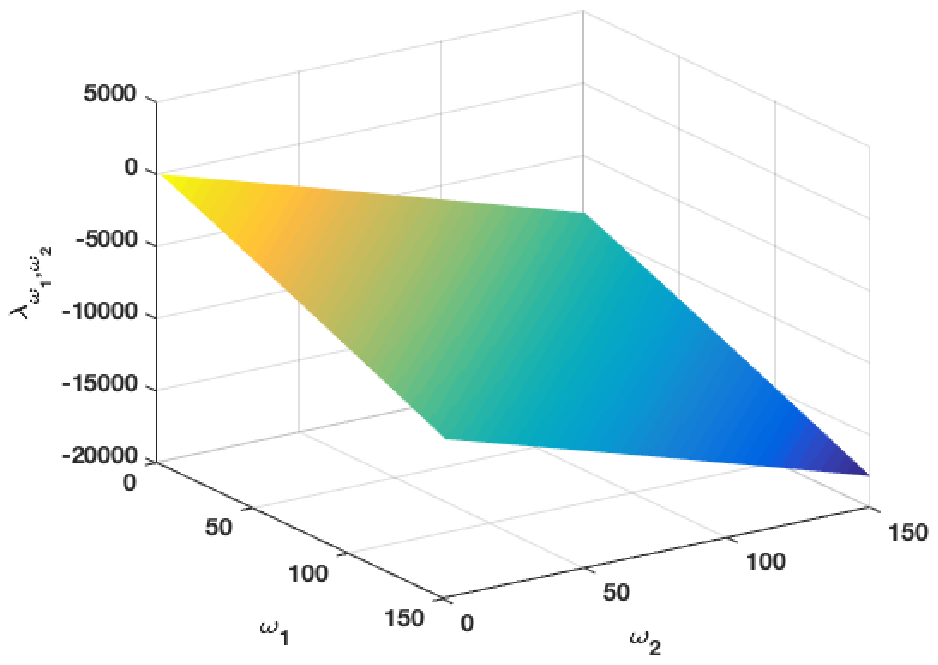

*Figure 4.7*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4). *These results use our new method with three interpolation points and 150 grid points per dimension.*

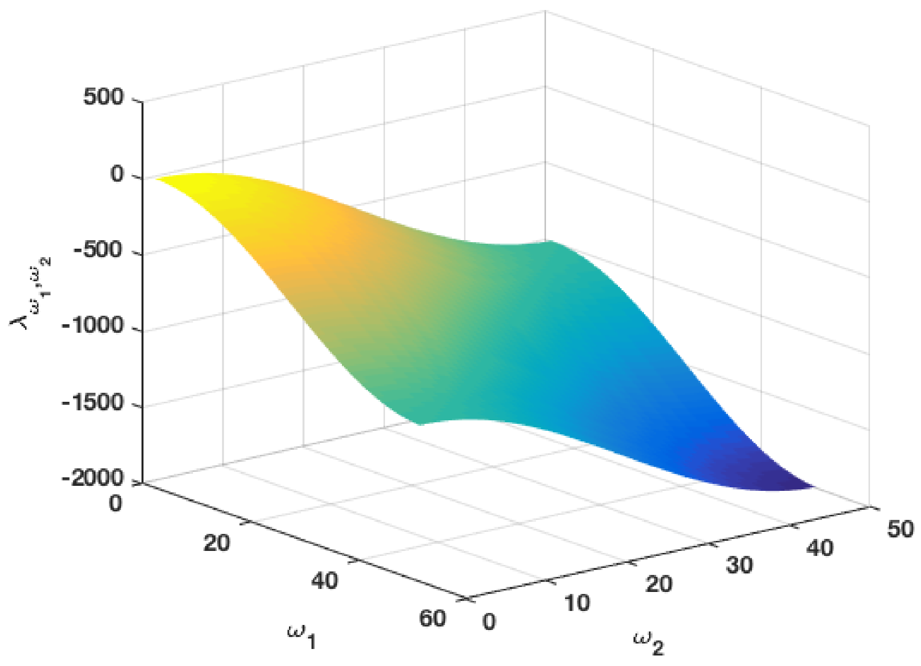

*Figure 4.8*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4). *These results use the previous method for KSS [14] with five interpolation points and 50 grid points per dimension.*

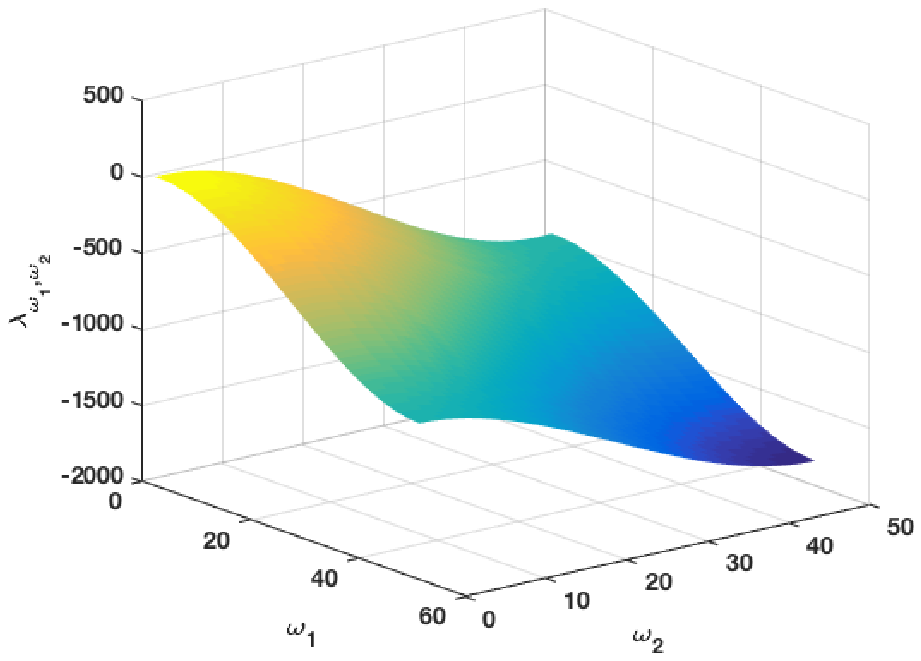

*Figure 4.9*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4). *These results use our new method with five interpolation points and 50 grid points per dimension.*

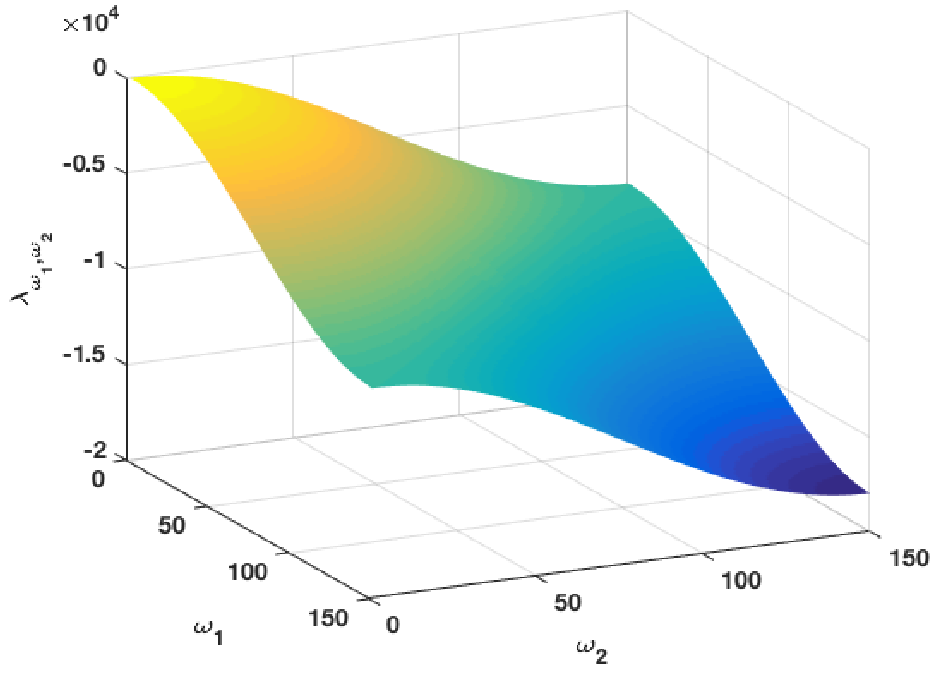

*Figure 4.10*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4)*. These results use the previous method for KSS [14] with five interpolation points and 150 grid points per dimension.*

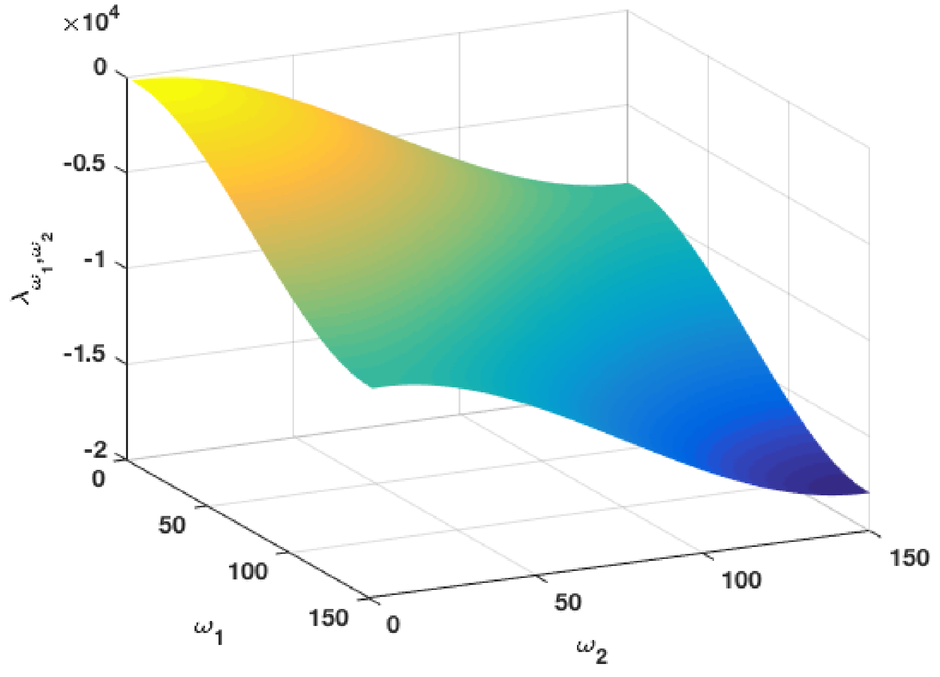

*Figure 4.11*: *Results for the PDE* (4.2) *with Neumann boundary conditions* (4.3) *using the equation* (4.4)*. These results use our new method with five interpolation points and 150 grid points per dimension.*

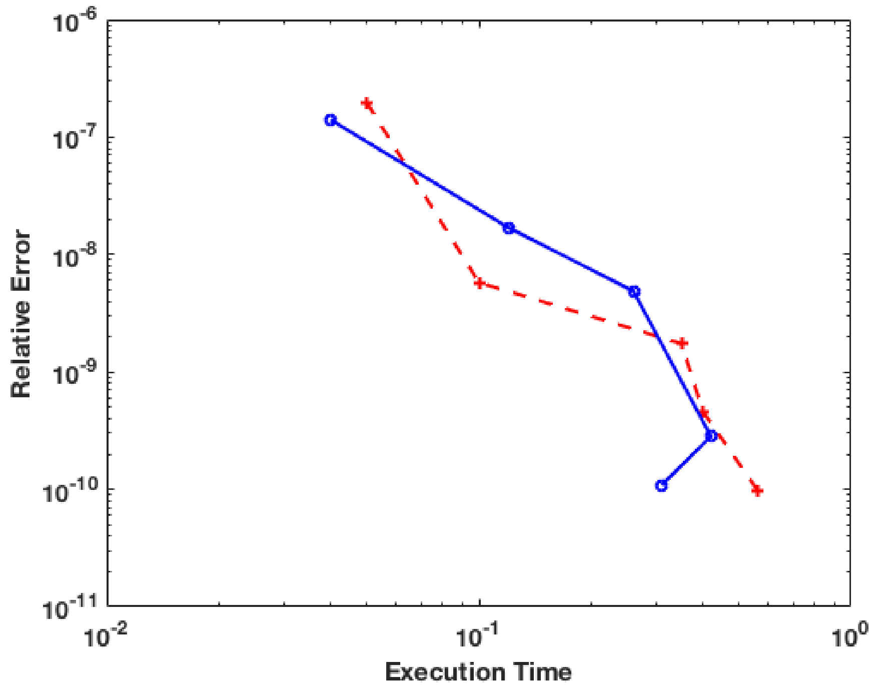

*Figure 4.12*: *Relative Error versus Execution Time with five interpolation points and 50 grid points per dimension. The blue curve represents the previous method for KSS [14] and the red dashed curve represents our new method.*

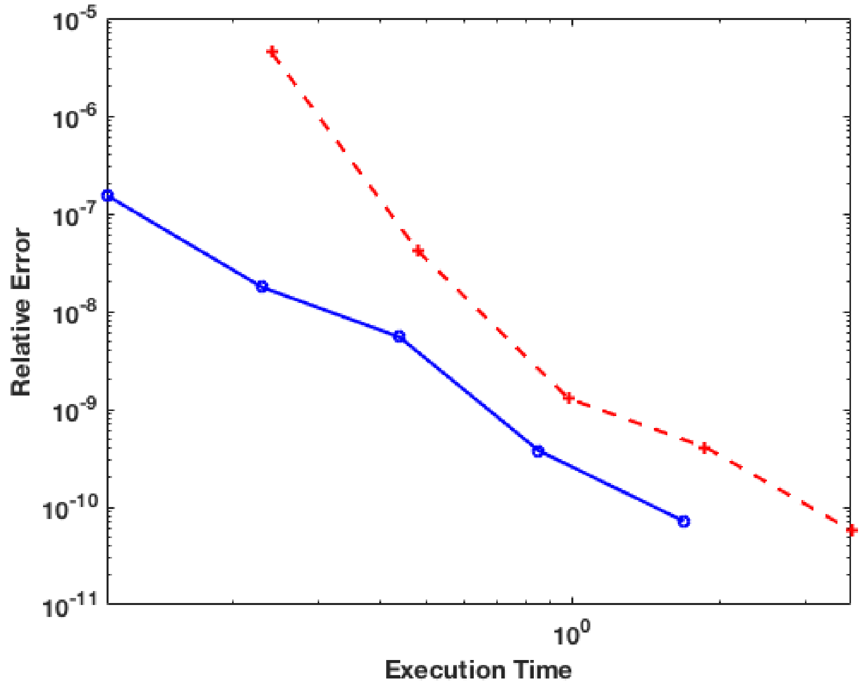

*Figure 4.13*: *Relative Error versus Execution Time with five interpolation points and 150 grid points per dimension. The blue curve represents the previous method for KSS [14] and the red dashed curve represents our new method.*

# Chapter 5

# CONCLUSION

Our approach of computing the frequency-dependent nodes is slightly less efficient then the most recent approach used in [11]. Even though our approach is slightly less efficient, it is more readily applicable since it requires much less information from the user than approach from [11]. The research of this method was intended to improve the overall user-friendly aspect of KSS methods while also preserving the advantages KSS has over other time-stepping methods, in terms of scalability to high resolution.. The previous approach of computing the frequency dependent nodes needed to know information about the coefficients of the PDE which, in turn, required the user to compute many tedious formulas by hand before being able to use the KSS method. Our new approach has shown to hold up to the accuracy of other approaches and requires minimal input from the user such as the type of PDE, boundary conditions, and number of grid points.

In the future, we plan to broaden our scope to cover more types of PDEs and different boundary conditions so that we can eventually create a complete, user-friendly, software package of KSS methods for public use. In this thesis, we could use symmetric Lanczos since we had a self-adjoint differential operator. If we use a differential operator that is not self-adjoint, then we would need to use Arnold iterations. Future research will include stiff PDEs with this type of differential operator that uses Arnold iterations. This work can be expanded to easily apply to other PDEs, such as the wave equation that has been previously applied with KSS methods [12, 13, 8].

# BIBLIOGRAPHY

[1] Atkinson, K.: *An Introduction to Numerical Analysis, 2nd Ed.* Wiley (1989)

[2] Golub, G. H., Meurant, G.: Matrices, Moments and Quadrature. *Proceedings of the 15th Dundee Conference*, June-July 1993, Griffiths, D. F., Watson, G. A. (eds.), Longman Scientific & Technical (1994).

[3] Golub, G. H., Meurant, G.: *Matrices, Moments and Quadrature with Applications*. Princeton University Press (2010).

[4] Golub, G. H., Underwood, R.: The block Lanczos method for computing eigenvalues. *Mathematical Software III*, J. Rice Ed., (1977) 361-377.

[5] Hochbruck, M., Lubich, C.: On Krylov Subspace Approximations to the Matrix Exponential Operator. *SIAM J. Numer. Anal.* **34** (1996) 1911-1925.

[6] Lambers, J. V.: Enhancement of Krylov Subspace Spectral Methods by Block Lanczos Iteration. *Electron. T. Numer. Ana.* **31** (2008) 86-109.

[7] Lambers, J. V.: Explicit High-Order Time-Stepping Based on Componentwise Application of Asymptotic Block Lanczos Iteration. *Numerical Linear Algebra with Applications* **19**(6) (2012) 970-991.

[8] Lambers, J. V.: An Explicit, Stable, High-Order Spectral Method for the Wave Equation Based on Block Gaussian Quadrature. *IAENG Journal of Applied Mathematics* **38** (2008) 333-348.

[9] Lambers, J. V.: Practical Implementation of Krylov Subspace Spectral Methods. *Journal of Scientific Computing* **32** (2007) 449-476.

[10] Lambers, J. V.: A Multigrid Block Krylov Subspace Spectral Method for Variable-Coefficient Elliptic PDE. *IAENG Journal of Applied Mathematics* **39**(4) (2009) 236-246.

[11] Cibotarica, A., Lambers, J. V., Palchak, E. M.: Solution of Nonlinear Time-Depdendent PDE Through Componentwise Approximation of Matrix Functions. *Journal of Computational Physics* **321** (2016) 1120-1143.

[12] Lambers, J. V.: A Spectral Time-Domain Method for Computational Electrodynamics. *Advances in Applied Mathematics and Mechanics* **1**(6) (2009) 781-798.

[13] Lambers, J. V. Krylov Subspace Spectral Methods for the Time-Dependent Schrödinger Equation with Non-Smooth Potentials. *Numerical Algorithms* **51** (2009) 239-280.

[14] Palchak, E. M., Cibotarica, A., Lambers, J. V.: Solution of Time-Dependent PDE Through Rapid Estimation of Block Gaussian Quadrature Nodes. *Lin. Alg. Appl.* **468** (2015) 233-259.