Spring 5-1-2021

A Component-Wise Approach to Smooth Extension Embedding Methods

Vivian Montiforte

A COMPONENT-WISE APPROACH TO SMOOTH EXTENSION EMBEDDING METHODS

by

Vivian Ashley Montiforte

A Dissertation
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Mathematics and Natural Sciences
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved by:

Dr. James Lambers, Committee Chair
Dr. C.S. Chen
Dr. Haiyan Tian
Dr. Huiqing Zhu

May 2021

ABSTRACT

Krylov Subspace Spectral (KSS) Methods have demonstrated to be highly scalable methods for PDEs. However, a current limitation of these methods is the requirement of a rectangular or box-shaped domain. Smooth Extension Embedding Methods (SEEM) use fictitious domain methods to extend a general domain to a simple, rectangular or box-shaped domain. This dissertation describes how these methods can be combined to extend the applicability of KSS methods, while also providing a component-wise approach for solving the systems of equations produced with SEEM.

ACKNOWLEDGMENTS

Throughout my doctoral studies, research, and the writing of this dissertation, I have received an immense amount of support and assistance.

I would first like to thank my advisor, Dr. James Lambers, whose expertise was imperative in forming this research. Dr. Lambers has been an influential mentor throughout my years at USM and has played a crucial part in me completing my degree. Thank you for the years of guidance and help whenever I needed it. I woud like to acknowledge my committee members for their time and the feedback they have provided for this dissertation.

I would also like to thank my husband, Kyle, who has made all of this possible. He has continued to support me through all of my studies and has always pushed me to keep going. I also need to acknowledge my children, Kooper and Kinsley, who have kept me motivated to finish my dissertation and complete my studies. Thank you for giving me another reason to reach as high as the stars. Lastly, I would like to thank the rest of my family for always believing in me and helping in any way they can.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

BVP	-	Boundary Value Problem
DCT	-	Discrete Cosine Transform
FFT	-	Fast Fourier Transform
IDCT	-	Inverse Discrete Cosine Transform
KSS	-	Krylov Subspace Spectral Methods
ODE	-	Ordinary Differential Equation
PCG	-	Preconditioned Conjugate Gradient
PDE	-	Partial Differential Equation
SEEM	-	Smooth Extension Embedding Methods

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., Λ for a diagnonal 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 typset 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 abolute 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

Consider the second-order boundary value problem,

$$\begin{cases} \mathcal{A}u = f & \text{in } \Omega \\ \mathcal{B}u = g & \text{on } \Gamma = \partial\Omega \end{cases} \tag{1.1}$$

where $\mathcal{A}$ represents a second-order differential operator, such as the Laplace operator, and $\mathcal{B}$ represents a boundary operator, such as the trace Dirichlet operator. In this boundary value problem, Ω represents the general domain while Γ represents the boundary. There exists numerical methods that are efficient and accurate when Ω is a simple domain, such as a rectangular or box-shaped domain. These numerical methods, for instance, spectral methods, are not directly applicable for a more complicated domain Ω. A different approach to these intricate general domains, Ω, is to use a fictitious domain. Fictitious domain methods allow the general domain, Ω, to be embedded into a simpler fictitious domain, $\mathbb{B}$, which is larger than the original general domain. The simpler domain, $\mathbb{B}$, allows numerical methods, such as spectral methods, to be easily implemented.

As with many approaches, there is a drawback with fictitious domain methods. The original boundary value problem (BVP) is only defined on the general domain Ω. Since the general domain has been embedded into a larger, simpler domain $\mathbb{B}$, the general domain has now become a proper subset of the fictitious domain $\mathbb{B}$, ($\Omega \subset \mathbb{B}$). As a result, the original BVP, after spatial discretization, provides an underdetermined set of equations for unknowns on the larger fictitious domain $\mathbb{B}$. Some approaches extend the solutions, u, of the original BVP to the underdetermined system resulting in discretizations not being able to accurately approximate them. Other methods have approached this matter by smoothly extending the BVP to the entire fictitious domain in such a way that the resulting system was no longer underdetermined. This second approach introduces new difficulties of appropriately extending the data while ensuring that the original BVP equations are satisfied, [4].

An alternative approach for extending the data to the fictitious domain comes from a Smooth Extension Embedding Method (SEEM), which uses the BVP as a constraint to ensure a smooth solution. Background of SEEM and more details of this process are described in Section 2.1. While SEEM ensures a smooth solution within this fictitious domain, the method used to solve the solution has several weaknesses. A drawback of this

method is the lack of scalability. As the number of grid points increases, the systems of equations become ill-conditioned. These ill-conditioned systems of equations cause iterative methods to converge slowly, while the overall computational expense increases substantially. To avoid the drawbacks of solving the solution using the current method of SEEM, we introduce Krylov Subspace Spectral (KSS) Methods. Background of KSS and further details are given in Section 2.2. Using this method allows the systems of equations to be solved with a component-wise approach. KSS methods have been proven to be highly scalable on simple domains. With the benefits of SEEM properly extending a general domain to a simple domain, we can broaden the applicability of KSS methods to general domains, while also improving the scalability of SEEM.

Chapter 2

BACKGROUND

In order to extend the capability of KSS methods to general domains and improve the scalability of SEEM, we need an understanding of the backgrounds of each method. In this chapter, we will first discuss the background of SEEM and how this method extends general domains to fictitious domains in such a way that a smooth solution is ensured using an optimization problem. This smooth solution is important in order to apply spectral methods. We will further discuss the spectral discretization process and how the optimization problem reduces to the normal equation that will be used for approximating the solution. Next, we will discuss the background of KSS methods and how solutions are approximated using a component-wise approach.

2.1 Smooth Extension Embedding Method

The Smooth Extension Embedding Method (SEEM), by Daniel Agress [2], uses fictitious domain methods with a different approach for smoothly extending the BVP. SEEM utilizes the BVP as a constraint to ensure a smooth solution by minimizing the constrained optimization problem defined on the entire fictitious domain $\mathbb{B}$. Like many numerical methods, this approach has its own limitations. While SEEM ensures that a smooth solution is chosen, it requires the solution of the BVP to have a smooth extension to a rectangular domain that includes the original domain. For simplification, the BVP from (1.1) is written as one equation, $\mathcal{C}u = b$, where

$$\mathcal{C} = \begin{pmatrix} \mathcal{A} \\ \mathcal{B} \end{pmatrix} \text{ and } b = \begin{pmatrix} f \\ g \end{pmatrix}.$$

Given a smoothing norm $\|\cdot\|_{\mathbb{S}}$ on $\mathbb{B}$, we attempt to solve the following constrained optimization problem

$$\text{argmin}_{\{\mathcal{C}u=b\}} \frac{1}{2}\|u\|_{\mathbb{S}}^2. \tag{2.1}$$

The operators $\mathcal{A}$ and $\mathcal{B}$ are left in their original form; the operators are only constrained on their original domains Ω and Γ ($\partial\Omega$), respectively. The norm $\|\cdot\|_{\mathbb{S}}$ on $\mathbb{B}$ is chosen to enforce a smooth solution that satisfies the constraint. Since a smooth solution has been selected, a spectral discretization can be used to approximate the solution with a high degree of accuracy.

Since the solution can be approximated using spectral methods, we can now briefly discuss the discretization process and solving the minimization problem. The fictitious domain $\mathbb{B}$ is discretized by a Chebyshev grid $\mathbb{B}^m$. It is important to note here the choice of using a Chebyshev grid. The previous method of SEEM, [1], was implemented using a Fourier series with a periodic extension. This approach allowed the use of the Fast Fourier Transform (FFT) which made discretizations simple and was computationally efficient. Although this method proved to effectively solve the BVP, it had a few drawbacks. The first problem was due to the periodic extension of the BVP on $[-\pi,\pi)^d$, where d is the dimension. The original domain Ω occupied only a small fraction of the extended domain, while the entire periodic domain was essential in order to smoothly extend the BVP into a periodic function. Due to this, computational resources were wasted from solving the extension on a much larger grid than required for the solution of the BVP. Another drawback impacted the accuracy of the discretized solution due to the 2π-periodic extension of the solution needing a larger optimization norm than the solution itself. In order to force the extension to be periodic, the derivatives outside of the original domain were required to be large. This requirement negatively affected the accuracy. These drawbacks were mitigated when switched to a Chebyshev extension instead of a Fourier series. Using a Chebyshev grid allows the extension to be on a non-periodic box where the domain $\mathbb{B} = [-1,1]^d$. Since the extension is no longer on a periodic domain, much more of the original domain Ω is included in the new domain $\mathbb{B}$, which saves computational expense. Also, the optimization norm used for the extension of the solution is not significantly larger than the norm for the solution itself. This increases the accuracy of the discretized solution compared to the method implementing the Fourier series. In [2], numerical experiments were performed showing that the accuracy was equivalent on significantly smaller grids when compared with the Fourier series discretizations. Lastly, it is relevant to mention the Chebyshev method maintains the same efficiency of SEEM based on the Fourier method since the FFT can also be used to carry out computations on the Chebyshev grid.

Now that we understand the purpose of the fictitious domain $\mathbb{B}$ being discretized using a Chebyshev grid $\mathbb{B}^m$, we can continue the brief dicussion of the discretization process. The original domain Ω can be discretized as

$$\Omega^m = \Omega \cap \mathbb{B}^m,$$

with Ω^m representing the interior points. The original boundary Γ can be discretized as

$$\Gamma^m = \{\mathbf{y}_1, \ldots, \mathbf{y}_{N_m^\Gamma}\},$$

with Γ^m representing the boundary points where N_m^Γ is the number of boundary points. The

operators $\mathcal{C}$ and $\mathcal{S}$ are discretized as

$$C^m = \begin{bmatrix} A^m \\ B^m \end{bmatrix}$$

and S^m. The interior discrete differential operator A^m comes from discretizing the second order differential operator $\mathcal{A}$ at the interior points Ω^m. B^m is a discrete form of the boundary operator $\mathcal{B}$ from discretizing at the boundary points Γ^m. Lastly, the matrix S^m represents a discretization that approximates the smoothing norm $\mathcal{S}$. In order to uphold the accuracy of the method, spectral discretization is selected for these operators. The original minimization problem (2.1) can now be represented using the previously discretized operators. After dropping indexes for clarity, the optimization problem reduces to the following regularized normal equation

$$\mathbf{u} = S^{-1}C^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{2.2}$$

The linear system can now be efficiently approximated with chosen spectral methods. As will be shown later in experiments, SEEM can be used to attain spectrally accurate methods that work with general domains. The same can also be shown for general boundary conditions and differential operators with constant or variable coefficients. The approach using SEEM has a quite simple implementation, requiring only discretization of the BVP matrix $\mathcal{C}$ and the regularizing norm $\mathcal{S}^{-1}$ on a rectangular grid.

2.1.1 Discretization of Domain

The use of Chebyshev polynomials discretized on the Chebyshev roots' grid allows functions to be approximated while maintaining spectral accuracy on the fictitious domain $\mathbb{B}$. The domain $\mathbb{B}$ is discretized by a regular grid $\mathbb{B}^m$ using Chebyshev roots, where the BVP is posed in $\Omega \subseteq [-1,1]^d$. Here, d represents the number of dimensions for general purpose. The domain $[-1,1]^d$ is discretized by a product set of the Chebyshev grid $\mathbb{B}^m$, given by

$$\mathbb{B}^m = \left\{ (\mathbf{x}^1, \ldots, \mathbf{x}^d) \;\middle|\; \mathbf{x}^i \in \mathbb{C}^m \text{ for } 1 \leq i \leq d \right\},$$

where

$$\mathbb{C}^m = \left\{ \cos\left(\pi \frac{2k+1}{2m} \right) \;\middle|\; 0 \leq k \leq m-1 \right\}.$$

The discretized interior, Ω^m, is then defined as the points of the original domain Ω that intersect with the discretized fictitious domain $\mathbb{B}^m$, ($\Omega^m = \Omega \cap \mathbb{B}^m$). The discretized interior contains $N_\Omega^m := |\Omega^m|$ points. The discretized boundary, Γ^m, is defined by choosing equally spaced points along the boundary Γ, creating a set $\Gamma^m = \{\mathbf{y}_i, \ldots, \mathbf{y}_{N_\Gamma^m}\}$ where N_Γ^m represents the number of boundary points ($N_\Gamma^m := |\Gamma^m|$ points). It is important to note that the boundary

points do not need to exist on the regular grid $\mathbb{B}^m$ but instead must be a portion of the boundary Γ. Increasing the number of discretized points on the boundary improves the accuracy, which leaves the choice of the density of the boundary points. However, the matrix can become severely ill-conditioned if the regular grid cannot distingush the boundary points. This occurs when the boundary points are spaced closer together than the regular Chebyshev grid points, [2]. In the numerical results, $\frac{m}{2}$ boundary points are placed per unit length for a two-dimensional problem. Figure 2.1 displays an example of a general domain, in this case, a disc-shaped domain that has been embedded into a simpler, fictitious domain which is then discretized using a Chebyshev grid.

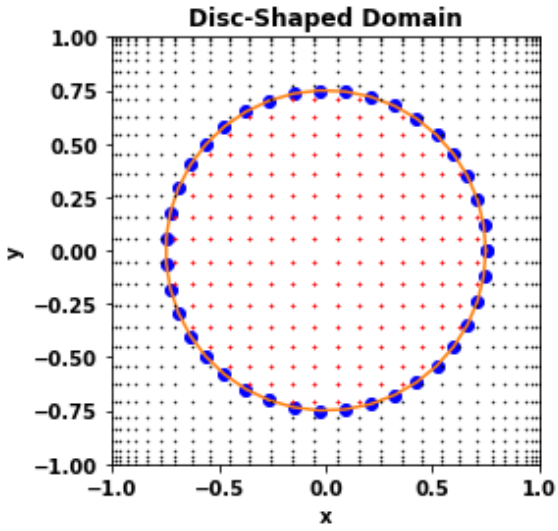


Figure 2.1: Disc-Shaped Domain
The Chebyshev grid, $\mathbb{B}^m$, is represented by the smaller **black** dots covering the entire box. The solid **orange** line of the disc-shaped domain represents the boundary Γ where $\{(r,\theta)|r<0.75\}$. The larger **blue** circles represent the discretized boundary or boundary points, Γ^m. The interior points, Ω^m, are represented by the **red** crosses where the Chebyshev grid points lie inside the boundary.

2.1.2 Discretization of Differential Operators

Recall that we are solving a boundary value problem of the following form,

$$\begin{cases} \mathcal{A}u = f & \text{in } \Omega \\ \mathcal{B}u = g & \text{on } \Gamma \end{cases},$$

where SEEM uses the entire BVP as a system of constraints to an optimization problem over the fictitious domain $\mathbb{B}$. This approach requires the discrete solution to satisfy the discretized

interior differential equation and the discretized boundary condition. The discretized interior differential equation corresponds to the discretization A of the operator $\mathcal{A}$ at the interior points Ω^m. The discretized boundary condition corresponds to the discretization B of the operator $\mathcal{B}$ at the boundary points Γ^m. These discretizations are constructed in the setting of the Chebyshev grid.

We first discuss the discretization of the interior differential operator $\mathcal{A}$. The discretization A is an $N_\Omega^m \times m^d$ matrix constructed from values over the entire Chebyshev grid $\mathbb{B}^m$ to approximate the second order differential operator $\mathcal{A}$ at the interior points Ω^m. Recall that N_Ω^m represents the number of interior points and d represents the dimension. Therefore, the matrix A has as many rows as the number of interior points and as many columns as the number of points in the Chebyshev grid $\mathbb{B}^m$. The interior operator for a general second order elliptic BVP has the following form,

$$\mathcal{A}u = -a_{ij}u_{x_ix_j} + b_iu_{x_i} + cu.$$

The derivatives are evaluated using discrete differentiation matrices for a Chebyshev grid.

Next, we discuss the discretization of the boundary operator $\mathcal{B}$. The discretization B is an $N_\Gamma^m \times m^d$ matrix constructed from values on the entire Chebyshev grid $\mathbb{B}^m$ to approximate the boundary operator $\mathcal{B}$ at the boundary points Γ^m. Recall that N_Γ^m represents the number of boundary points. Therefore, the matrix B has as many rows as the number of boundary points and as many columns as the number of points in the Chebyshev grid $\mathbb{B}^m$. For some smooth functions a and b defined on the boundary Γ ($\partial\Omega$), the boundary operator has the following form

$$[B]_{i_\bullet} = a(\mathbf{y}_i)\delta_{\mathbf{y}_i} + b(\mathbf{y}_i)(\delta_{\mathbf{y}_i} \circ \nabla) \cdot \nu_{\mathbf{y}_i},$$

where $\mathbf{y}_i$ is a point from the set of discretized boundary points, $\Gamma^m = \{\mathbf{y}_i, \ldots, \mathbf{y}_{N_\Gamma^m}\}$. The unit outward pointing normal vector to Γ at the point $\mathbf{y}_i$ is represented with $\nu_{\mathbf{y}_i}$. The matrix B is constructed by each row where the i-th row of B, denoted by $i_\bullet$, represents the evaluation of the boundary condition at the i-th point of Γ^m. The spectral interpolation operators δ_y and $\delta_y \circ \nabla$ are used to evaluate the trace operator and the normal derivative on the boundary, respectively. It is important to note that Dirichlet boundary conditions corresponds to $a \equiv 1$ and $b \equiv 0$, while $a \equiv 0$ and $b \equiv 1$ corresponds to Neumann boundary conditions.

2.1.3 Chebyshev Smoothing Norm

The smoothing or regularizing norm $\mathcal{S}$ in (2.1) is chosen so that it can be efficiently computed on a Chebyshev grid. Implementing SEEM on a Chebyshev grid first requires defining the operator $\mathcal{D}$. By letting the operator $\mathcal{D} = M\left[\sqrt{1-x^2}\right] \circ \frac{\partial}{\partial x}$, the Chebyshev polynomials

satisfy the following eigenvalue equation

$$-\mathcal{D}^2 T_j(\mathbf{x}) = j^2 T_j(\mathbf{x}).$$

The benefit of the eigenvalue equation is to make the matrix used to define the norm below easily diagonalizable. This allows the norm to be readily computed and the inverse to be quickly found. Therefore, we have the following smoothing norm

$$\|\cdot\|_{\mathcal{S}_p}^2 = \left\| \left(1 - \sum_{i=1}^{d} \mathcal{D}_i^2\right)^{p/2} \mathbf{u} \right\|_{L_2}^2$$

which uses the operator $\mathcal{D}$ along each dimension number i. Because of this, the smoothing operator

$$\mathcal{S}_p^{-1} u = \mathfrak{C}^{-1} \circ M\left[(1+|k|^2)^{-p}\right] \circ \mathfrak{C} u$$

can be applied efficiently. Here, $\mathfrak{C}$ is the Chebyshev transform, $(k)_{k\in\mathbb{N}^d}$ is the Chebyshev frequency vector, or the degrees of the Chebyshev polynomial, and $M[\bullet]$ represents multiplication by the function. Similarly, given the Chebyshev grid $\mathbb{B}^m$, the discrete Chebyshev functions also satisfy a discrete eigenvalue equation

$$-\mathcal{D}_m^2 T_j(x_\bullet) = j^2 T_j(x_\bullet), \quad j \in \{0,\dots,m-1\},$$

where $x_\bullet$ represents a vector of grid points. The discrete norm is defined in an analogous way, resulting in the discrete smoothing operator

$$S_p^{-1}\mathbf{u} = \mathfrak{C}_m^{-1} \circ M\left[(1+|k_\bullet|^2)^{-p}\right] \circ \mathfrak{C}_m \mathbf{u}. \tag{2.3}$$

Here, $\mathfrak{C}_m$ is the discrete Chebyshev transform denoting the diagonalization of the operator $\mathcal{S}_p$ from the eigenvalue equation, $(k_\bullet)_{k\in\{0,\dots,m-1\}^d}$ is the discrete Chebyshev frequency vector on the d-dimensional box $\mathbb{B}^m$, and $M[\bullet]$ represents multiplication by the function or entry-wise multiplication of vectors.

2.1.4 Previous Method for Solving

Recall the form of the solution in (2.2) for the BVP using SEEM

$$\mathbf{u} = S^{-1}C^T(CS^{-1}C^T)^{-1}\mathbf{b}.$$

The previous approach used Preconditioned Conjugate Gradient Method (PCG). The linear operators C, C^T, and S^{-1} are easily implemented using the FFT which leads to fast computation for iterative methods. The matrix $CS^{-1}C^T$ is a symmetric positive definite matrix.

A drawback of PCG is that the smoothing operator is very ill-conditioned for large grids, which in return, requires good preconditioning. However, preconditioning is more effective for lower order smoothers. PCG was chosen since it is an iterative solver which relies on the implicit form of the linear operator. This choice allowed solving the system on a larger grid than compared to the use of explicit matrices with a QR decomposition, [1].

2.2 Krylov Subspace Spectral Methods

We now introduce KSS methods in order to combine with and improve the approaches used in SEEM. Unlike the current method implemented in SEEM, KSS methods use a component-wise approach to compute matrix function-vector products such as the regularized normal equation in (2.2). Beyond improving the scalabilty of SEEM with this component-wise approach, the method used in SEEM for general domains extends the current capabilities of KSS methods. KSS methods are highly scalable on a simple, rectangular domain. Using the fictitious domain approach in SEEM allows KSS to expand its applicability for general domains.

Consider the parabolic PDE on the interval $(0,\, 2\pi)$,

$$u_t + Lu = 0,\ t > 0, \tag{2.4}$$

with initial data, $u(x,0) = u_0(x)$. L is a self-adjoint, positive definite, second-order differential operator. Using the spatial discretization of the parabolic PDE in (2.4), 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 ω. 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

$$\mathbf{u}^H f(A)\mathbf{v}, \tag{2.5}$$

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.

2.2.1 Approximating Bilinear Forms

We now discuss how to approximate the bilinear form in (2.5) where the definitions provided after this equation still hold. The 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 [5, 6] 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 λ_j, $j = 1,\ldots,K$, and weights ω_j, $j = 1,\ldots,K$, are obtained 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 [3]. As another option, the following block approach is considered [7],

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

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

$X_0 = 0$, $X_1 = [\ \mathbf{u} \quad \mathbf{v}\ ]$ (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 2×2 matrices, M_j and B_j, where B_j is upper triangular. Together these matrices form the block tridiagonal matrix, T_K:

$$T_K = \begin{bmatrix} M_1 & B_1^H & & & \\ B_1 & M_2 & B_2^H & & \\ & \ddots & \ddots & \ddots & \\ & & B_{K-2} & M_{K-1} & B_{K-1}^H \\ & & & B_{K-1} & M_K \end{bmatrix}. \tag{2.6}$$

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

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

$$X_1 = [\ \mathbf{u} \quad \mathbf{v}\ ],$$
$$\underline{\overline{X}}_K = [X_1, X_2, \ldots, X_K].$$

Then, using the block approach,

$$\begin{aligned}
&[\ \mathbf{u} \quad \mathbf{v}\ ]^H f(A)[\ \mathbf{u} \quad \mathbf{v}\ ] \\
&= X_1^H f(A) X_1 \\
&= E_{12}^H \underline{\overline{X}}_K^H f(A) \underline{\overline{X}}_K E_{12} \\
&= E_{12}^H f(\underline{\overline{X}}_K^H A \underline{\overline{X}}_K) E_{12} + \text{error}
\end{aligned} \tag{2.7}$$

where

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

It is important to note that λ_j is a scalar and by using $f(T_K)$, (2.7) 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,

$$\int_a^b f(\lambda)\, d\mu(\lambda) = \sum_{j=1}^{2K} f(\lambda_j) \mathbf{v}_j \mathbf{v}_j^H + \text{error}\ . \tag{2.8}$$

The block tridiagonal matrix, T_K, produces the nodes and the weights required from (2.8). The nodes are λ_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.

2.2.2 Block KSS

The block KSS method for the parabolic PDE in (2.4) 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

$$X_1(\omega) = \begin{bmatrix} \hat{\mathbf{e}}_\omega & \frac{\mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2} \end{bmatrix}$$

and

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

where

$$\mathbf{u}^n_\omega = \mathbf{u}^n - \hat{\mathbf{e}}_\omega \hat{\mathbf{e}}^H_\omega \mathbf{u}^n = \mathbf{u}^n - \hat{\mathbf{e}}_\omega \hat{u}(\omega, t_n). \tag{2.9}$$

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 (2.6). Every entry of T_K is a function of ω. After that, at time t_{n+1}, each Fourier coefficient of the solution is approximated by

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

2.2.3 Asymptotic Analysis of Block 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 in (2.4), 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,

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

where $D_j(\Delta t) = T_N^{-1}\hat{D}_j(\Delta t)T_N$. Here, $\hat{D}_j(\Delta t)$ is a diagonal matrix and T_N is the matrix of the N-point FFT. Each row of $\hat{D}_j(\Delta t)$ 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 ω is the wave number in the limit as $|\omega| \to \infty$.

In KSS Methods, $R_0 = \left[\begin{array}{cc} \hat{\mathbf{e}}_\omega & \mathbf{u}^n \end{array} \right]$ is used as the initial block for each $\omega = \frac{-N}{2}+1, \ldots, \frac{N}{2}$. 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:

$$R_0 = X_1 B_0, \tag{2.10}$$

where

$$X_1 = \left[\begin{array}{cc} \hat{\mathbf{e}}_\omega & \frac{\mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2} \end{array} \right] \text{ and } B_0 = \left[\begin{array}{cc} 1 & \hat{u}(\omega, t_n) \\ 0 & \|\mathbf{u}^n_\omega\|_2 \end{array} \right] \tag{2.11}$$

The vector $\mathbf{u}^n_\omega$ is defined from (2.9). As $|\omega| \to \infty$, $|\hat{\mathbf{u}}(\omega)|^n \to 0$, if the solution $\mathbf{u}$ is at least piecewise continuous. From the previous statement, B_0 converges to a diagonal matrix.

The next step is to compute

$$M_1 = X_1^H L_N X_1. \tag{2.12}$$

The matrix L_N is the spectral discretization of L, where L is the operator defined by $Lu = pu_{xx} + q(x)u$. By substituting the value of X_1 from (2.11) into (2.12), the following is obtained:

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

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 grid function $L_N \mathbf{u}^n_\omega$ associated to the wave number ω, and 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 the function is continuous, then the Fourier coefficients go to zero as $|\omega|$ increases. With this, the non-diagonal entries of M_1 become negligible as long as the solution is sufficiently regular. Therefore,

$$M_1 \approx \left[\begin{array}{cc} \omega^2 p + \bar{q} & 0 \\ 0 & R(L_N, \mathbf{u}^n) \end{array} \right].$$

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

$$R_1 = L_N X_1 - X_1 M_1 \approx \left[\begin{array}{cc} \tilde{\mathbf{q}} \hat{\mathbf{e}}_\omega & \frac{L_N \mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2} - R\left(L_N, \mathbf{u}^n_\omega \frac{\mathbf{u}^n_\omega}{\|\mathbf{u}^n_\omega\|_2}\right) \end{array} \right],$$

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 [18]. In this high-frequency limit, it was discovered that the block tridiagonal matrix T_K, produced by applying block Lanczos to R_0 (2.10), 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. This non-block Lanczos algorithm can be used to, at the minimum, estimate the true block Gaussian quadrature nodes.

Chapter 3

COMPONENT-WISE APPROACH

3.1 Introduction

In this chapter, we will introduce the new component-wise approach for solving the regularized normal equation in (2.2). Now that we have discussed the backgrounds for each method (KSS and SEEM), we can further discuss how the two will be combined in a way that brings benefits to both methods. Recall the form of the solution from (2.2),

$$\mathbf{u} = S^{-1}C^T(CS^{-1}C^T)^{-1}\mathbf{b}, \tag{3.1}$$

where S is the discretized Chebyshev smoothing norm, C contains the discretized differential operator A and the discretized boundary operator B, and $\mathbf{b}$ is the right hand side from the BVP. KSS methods approximate these forms of matrix function-vector products such as the bilinear form discussed in (2.5). For this research, $\mathbf{u}^T f(A)\mathbf{v}$ is the bilinear form used in the following algorithms.

3.2 Algorithm

The solution from (3.1) can be represented as

$$\mathbf{u}_m = S^{-1}\mathbf{y}$$

where

$$\mathbf{y} = C^T(CS^{-1}C^T)^{-1}\mathbf{b} \tag{3.2}$$

Each component of $\mathbf{y}$ is a bilinear form $\mathbf{u}^T f(A)\mathbf{v}$ where the vector $\mathbf{u}$ represents the columns of C ($\mathbf{u} = \mathbf{c}_j$), the matrix A represents the matrix being inverted ($A = CS^{-1}C^T$), the vector $\mathbf{b}$ represents the right hand side ($\mathbf{v} = \mathbf{b}$), and the function f performs the inverse of the matrix A, where $f(\lambda) = 1/\lambda$ for A^{-1}. Each bilinear form is approximated using KSS. After implementing KSS, a post-process is performed by applying the inverse of the Chebyshev smoothing norm, S^{-1}, to the computed $\mathbf{y}$ to obtain the approximate solution. This original approach will be referred to as '*KSS Solve 1*'.

3.3 Other Approaches

As with most research, different approaches are tested in order to improve the method. There are several factors that can improve between methods whether that be the error compared to the exact solution, the computational expense from storage to number of iterations or time, rate of convergence, and so on. To have an efficient, highly scalable method, there must be a choice in determining what is the trade-off. In this section, we discuss a few different approaches for approximating the solution using KSS methods.

A second approach is representing the solution from (3.1) as

$$\mathbf{u}_m = S^{-1}C^T\mathbf{y},$$

where

$$\mathbf{y} = (CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.3}$$

Each component of $\mathbf{y}$ is a bilinear form $\mathbf{u}^T f(A)\mathbf{v}$ with the same representations of $A = CS^{-1}C^T$, $\mathbf{v} = \mathbf{b}$, and $f(\lambda) = 1/\lambda$ for A^{-1}. The difference in this approach is the representation for the vector $\mathbf{u}$, where instead of $\mathbf{u}$ representing the columns of C, it represents the standard basis ($\mathbf{u} = \mathbf{e}_j$). Another difference in this approach is in the post-processing. Similarly, as in the first approach, each bilinear form is approximated using KSS. After implementing KSS, the post-processing is completed by applying $S^{-1}C^T$ to the computed $\mathbf{y}$ to obtain the approximate solution. This second approach will be referred to as '*KSS Solve 2*'.

For the next two approaches, the inverse smoothing operator, S^{-1}, is broken down into the individual components as defined in (2.3). The goal of writing the solution in this way is to be able to compute the coefficients of the solution in a basis of Chebyshev polynomials. Since higher-degree Chebyshev polynomials oscillate more rapidly, this makes the approach more similar to the Fourier KSS method, in which both cases are using bases of functions that are, to some extent, concentrated in frequency. Using the original definition and dropping indexes for clarity, the inverse smoothing operator can be rewritten as

$$S^{-1} = \mathfrak{C}^{-1}M\mathfrak{C}, \tag{3.4}$$

where $\mathfrak{C}$ is the discrete Chebyshev transform and M is the diagonalized matrix from the eigenvalue equation that performs dampening in frequency space. By substituting (3.4) into (3.1), the following is obtained

$$\mathbf{u}_m = \mathfrak{C}^{-1}M\mathfrak{C}C^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.5}$$

Using the decomposed version of the inverse smoothing operator in (3.5), the third approach can be defined. For this approach, $\mathfrak{C}C^T$ has been rewritten so that it is represented as a single transpose. This is necessary to have a bilinear form for KSS methods. The solution for the third approach is now expressed as the following equation

$$\mathbf{u}_m = \mathfrak{C}^{-1}M(C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.6}$$

The solution can be represented similarly to the first two approaches as

$$\mathbf{u}_m = \mathfrak{C}^{-1}M\mathbf{y},$$

where

$$\mathbf{y} = (C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.7}$$

Again, each component of $\mathbf{y}$ is a bilinear form $\mathbf{u}^T f(A)\mathbf{v}$ with the same representations of $A = CS^{-1}C^T$, $\mathbf{v} = \mathbf{b}$, and $f(\lambda) = 1/\lambda$ for A^{-1}. The difference in this approach comes from the representation of the vector $\mathbf{u}$ and the post-processing. For this method $\mathbf{u} = (C\mathfrak{C}^T)_j$. KSS is applied to solve the bilinear form. Post-processing implements $\mathfrak{C}^{-1}M$ on the computed $\mathbf{y}$ to approximate the solution. This approach will be referred to as '*KSS Solve 3*'.

The last approach is similar to the third approach in which the solution in (3.5) with the decomposed version of the inverse smoothing operator, S^{-1}, is used. For this approach, the matrix M that performs dampening in frequency space is included as part of the bilinear form instead of implementing M in the post-processing stage. To accomplish this, $M\mathfrak{C}C^T$ must be rewritten as a single transpose. This is represented in the following equation

$$\mathbf{u}_m = \mathfrak{C}^{-1}(C\mathfrak{C}^TM)^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.8}$$

The solution can now be expressed as

$$\mathbf{u}_m = \mathfrak{C}^{-1}\mathbf{y},$$

where

$$\mathbf{y} = (C\mathfrak{C}^TM)^T(CS^{-1}C^T)^{-1}\mathbf{b}. \tag{3.9}$$

Similary to the other approaches, each component of $\mathbf{y}$ is a bilinear form $\mathbf{u}^T f(A)\mathbf{v}$ with the same representations of $A = CS^{-1}C^T$, $\mathbf{v} = \mathbf{b}$, and $f(\lambda) = 1/\lambda$ for A^{-1}. The difference in this last approach is how the vector $\mathbf{u}$ is represented and the post-processing application. In this fourth approach $\mathbf{u} = (C\mathfrak{C}^TM)_j$. KSS is implemented to approximate the bilinear form. Post-processing applies $\mathfrak{C}^{-1}$ on the computed $\mathbf{y}$ to obtain the approximate solution. This approach will be referred to as '*KSS Solve 4*'.

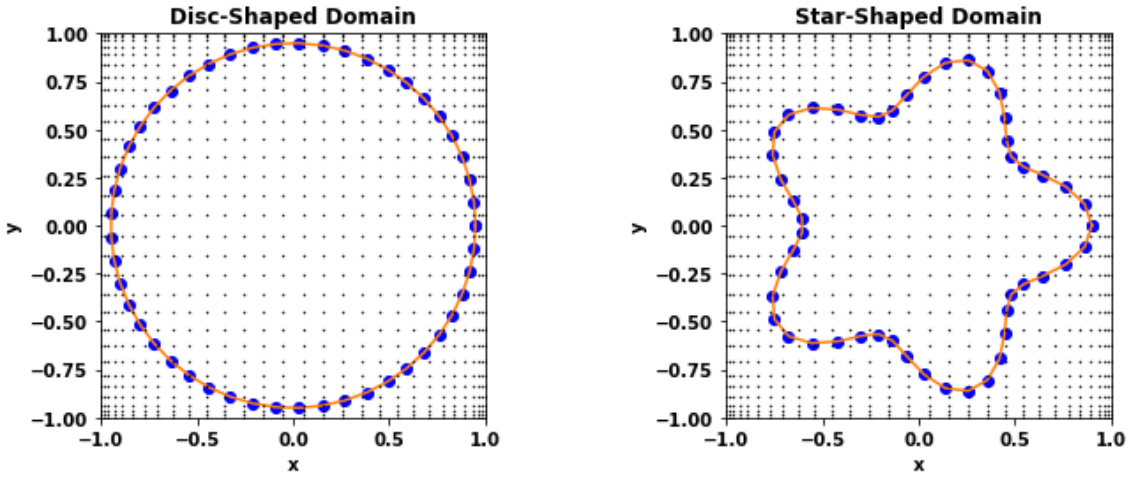
Chapter 4

RESULTS

In this chapter, we offer several numerical experiments to demonstrate the accuracy of the component-wise approach. The first section will cover results for each method discussed in the previous chapter. We will compare each KSS approach and discuss the most efficient approach of the four proposed algorithms. The following sections will use the most efficient algorithm compared with the previous PCG method. These numerical results are performed for Dirichlet, Neumann, and Robin boundary conditions on several different two-dimensional BVPs. All results will be computed on one of the following two-dimensional complex domains,

$$\Omega_1 = \{(r,\theta)|r < 0.95\}$$
$$\Omega_2 = \{(r,\theta)|r < 0.75(1+0.2\cos(5\theta))\},$$

where Ω_1 is a disc-shaped domain and Ω_2 is a star-shaped domain, also referred to as a flower-shaped domain with five petals. These complex domains are shown in Figure 4.1.



(a) Ω_1 (b) Ω_2

Figure 4.1: Complex Domains
Figure 4.1a is the domain Ω_1 and Figure 4.1b is the domain Ω_2, both discretized on the Chebyshev grid.

The numerical results shown in this chapter were computed on a disc-shaped domain and a star-shaped domain using the number of interior points and the number of boundary points for each grid size shown in Figures 4.1 and 4.2, respectively.

Table 4.1: *Number of Points - Disc-Shaped Domain (Ω_1).* This table includes the number of interior points Ω^m and boundary points Γ^m for each grid size m^2 that were used to compute the numerical results on the disc-shaped domain, Ω_1.

Grid Size ($\mathbb{B}^m$)	8^2	16^2	24^2	32^2	40^2
# of Interior Points (N_Ω^m)	24	104	240	408	652
# of Boundary Points (N_Γ^m)	10	19	29	38	47

Table 4.2: *Number of Points - Star-Shaped Domain (Ω_2).* This table includes the number of interior points Ω^m and boundary points Γ^m for each grid size m^2 that were used to compute the numerical results on the star-shaped domain, Ω_2.

Grid Size ($\mathbb{B}^m$)	8^2	16^2	24^2	32^2	40^2
# of Interior Points (N_Ω^m)	16	54	132	228	358
# of Boundary Points (N_Γ^m)	9	18	27	36	44

The tables in this chapter containing the numerical results include information for each grid size m^2 used and each order p of the smoothing operator S in (2.3). In these tables, '*# iter*' represents the average number of iterations, '*error*' represents the L^2 norm of the approximate solution compared with the exact solution, and the last row of the table shows the rate of convergence for each order of the smoothing norm.

The figures in this chapter containing the error plots show the errors computed using the L^2 norm and L^∞ norm, respectively, compared with the exact solution for each grid size m^2. Each plotted line represents the error with respect to the order of the smoothing operator, where the index of S corresponds to the value of p.

4.1 Example 1 - Dirichlet Problem on Ω_1

In this section, we show the numerical results for each of the four proposed component-wise KSS approaches along with results from the previous method of PCG. These results were executed using the following Dirichlet BVP on a disc-shaped domain of radius 0.95 (Ω_1) shown in Figure 4.1a

$$\begin{cases} -\Delta u = -6x + 6y & \text{in } \Omega_1 \\ u = x^3 - y^3 & \text{on } \Gamma = \partial\Omega_1 \end{cases} . \tag{4.1}$$

The grid sizes used for this initial example vary compared to the grid sizes used for later examples. For this reason, Table 4.3 has been included to show the number of interior points and boundary points for each grid size used in this section. Recall that the grid sizes in Table 4.3 come from the disc-shaped domain Ω_1 discretized on a Chebyshev grid.

Table 4.3: *Example 1 - Number of Points - Disc-Shaped Domain (Ω_1).* This table includes the number of interior points Ω^m and boundary points Γ^m for each grid size m^2 that were used to compute the following numerical results for Equation (4.1).

Grid Size ($\mathbb{B}^m$)	8^2	12^2	16^2	20^2	24^2
# of Interior Points (N_Ω^m)	24	60	104	164	240
# of Boundary Points (N_Γ^m)	10	15	19	24	29

In the following results, the value p corresponds to the order of the smoothing operator S from (2.3). In the numerical results for Equation (4.1), we use the grid sizes m^2 where $m = 8, 12, 16, 20, 24$ for each order $p = 1, 2, 3$.

In addition to the numerical results tables and the error plot figures previously described, other figures are included in this first section of the original numerical experiments for each proposed algorithm. These figures, such as Figures 4.3, 4.4, 4.5 for the first approach, contain visualizations of the coefficients of $Y = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$ on a logarithmic scale for $p = 1, 2, 3$, respectively, and each grid size m^2, where $m = 8, 16, 24$. The visualizations in these figures help to understand the magnitude of the coefficients and how they are changing as the grid size increases and as the order of the smoothing operator increases. This plays an important role in understanding why one of the four algorithms is chosen to precede with for the remaining numerical results.

4.1.1 '*KSS Solve 1*'

The first numerical results are computed using the original algorithm, '*KSS Solve 1*' from Section 3.2, where $\mathbf{y} = C^T(CS^{-1}C^T)^{-1}\mathbf{b}$ in (3.2) is the bilinear form being approximated with KSS. The computed solution for this approach comes after post-processing where $\mathbf{u}_m = S^{-1}\mathbf{y}$. The results come from solving Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. The error plots are shown in Figure 4.2. The error for the highest order smoother, $p = 3$, jumps up at the grid size $m = 24$. By looking at the numerical results in Table 4.4, it can be seen that this jump in error causes the rate of convergence to become very low at -2.63. This low convergence rate does have some effect from only computing results up to the grid size of 24. Computing the solution for larger grid sizes does raise the convergence rate as the error decreases. However, the overall rate of convergence is still not as desired. The numerical results for the previous PCG method are shown at the end of this section in Table 4.8 along with the error plots in Figure 4.18. By comparing '*KSS Solve 1*' with PCG, it can be seen that KSS has a better rate of convergence for the lower order smoother $p = 1$ along with much fewer iterations. For $p = 2$, both methods have around the same number of iterations, where PCG still has fewer iterations for $p = 3$.

Before reviewing the results for the second approach, '*KSS Solve 2*', the visualizations of the magnitude of the coefficients of Y for '*KSS Solve 1*' are investigated. Recall that $Y = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$ on a logarithmic scale. The idea behind observing these values is that KSS methods when used with a Fourier series tend to have negligibly small higher frequency components. When that happens, we can neglect the higher frequency components and only compute the lower frequency components while maintaining accuracy. We hope to see a similar trend for KSS methods in a Chebyshev basis. Figures 4.3, 4.4, and 4.5 contain the visualizations of the magnitude of the coefficients for the bilinear form on a logarithmic scale before post-processing. Each figure contains the results for grid sizes $m = 8, 16, 24$ and smoothing order $p = 1, 2, 3$, respectively. Looking at the figures, it can be seen that they do not have the same benefits of small higher frequency components.

Table 4.4: *Numerical Results ('KSS Solve 1') - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 1*'.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	17	5.58E-02	18	3.42E-02	18	2.86E-02
$12^2=144$	33	1.01E-01	38	6.55E-03	54	5.30E-01
$16^2=256$	48	4.52E-02	58	3.93E-03	77	2.95E-03
$20^2=400$	57	1.32E-01	72	5.98E-02	92	3.82E-04
$24^2=576$	76	4.63E-02	99	1.55E-02	145	5.12E-01
Rate of Conv.	0.17		0.72		-2.63	

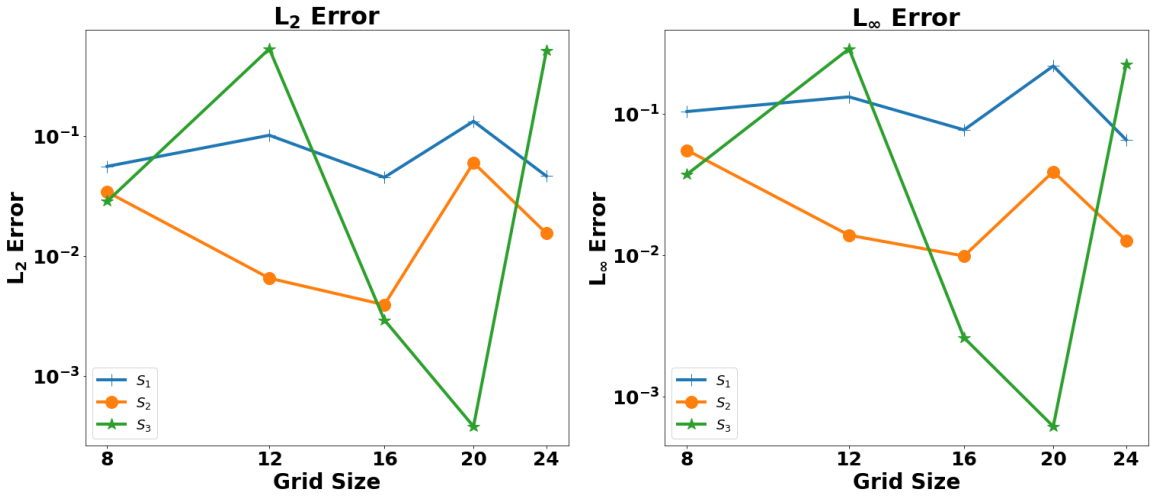
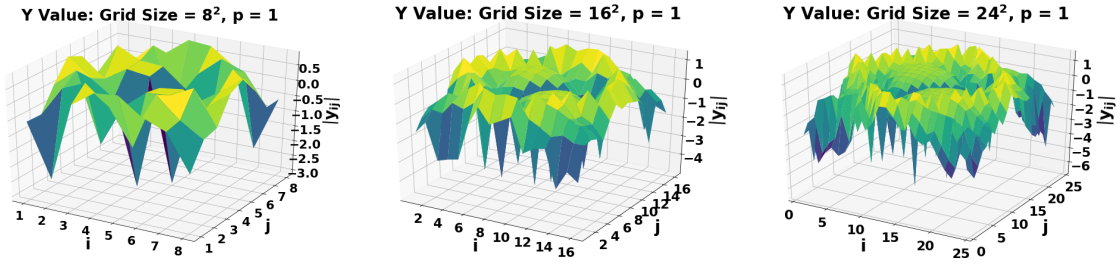


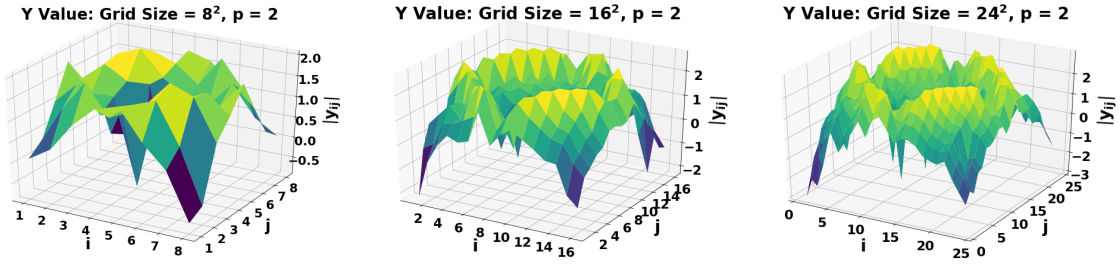
Figure 4.2: Error Plots ('*KSS Solve 1*') - Example 1 - Dirichlet Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.1) computed using '*KSS Solve 1*'. The index of S represents the value of p.



(a) (b) (c)

Figure 4.3: Y Value: '*KSS Solve 1*' with $p = 1$

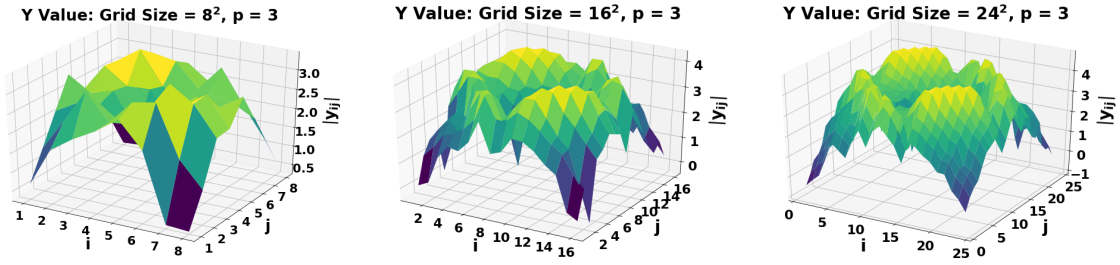
Figures 4.3a, 4.3b, and 4.3c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the grid point indices. Here, $\mathbf{y} = C^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.2) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.4: Y Value: '*KSS Solve 1*' with $p = 2$

Figures 4.4a, 4.4b, and 4.4c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the grid point indices. Here, $\mathbf{y} = C^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.2) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.5: Y Value: '*KSS Solve 1*' with $p = 3$

Figures 4.5a, 4.5b, and 4.5c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the grid point indices. Here, $\mathbf{y} = C^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.2) is used while approximating Equation (4.1).

4.1.2 '*KSS Solve 2*'

The next numerical results are computed using the second algorithm, '*KSS Solve 2*' from Section 3.3, where $\mathbf{y} = (CS^{-1}C^T)^{-1}\mathbf{b}$ in (3.3) is the bilinear form being approximated with KSS. The computed solution for this approach comes after post-processing where $\mathbf{u}_m = S^{-1}C^T\mathbf{y}$. The results come from solving Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. The error plots are shown in Figure 4.6 where it can already be seen that the convergence rate is better than that of the first approach, '*KSS Solve 1*'. The numerical results are shown in Table 4.5. From this table, a further comparison can be made with the numerical results in Table 4.4 for the first approach to see that the second algorithm has fewer iterations overall. If comparing with the PCG numerical results in Table 4.8, it can be seen that '*KSS Solve 2*' has fewer iterations for the lower order smoother $p = 1$. For $p = 2$, the number of iterations are roughly the same. The number of iterations for '*KSS Solve 2*' tend to grow for the higher order smoother $p = 3$. Overall, this second algorithm retains nearly the same accuracy as PCG. However, this approach is not getting the same benefits as in the KSS Fourier case since the number of iterations are increasing as the smoothing order increases.

Again, the visualizations of the magnitude of the coefficients of Y on a logarithmic scale before post-processing are considered. Due to the selection of the bilinear form, the dimensions of Y differ here than other methods. Instead of having the dimensions of the number of grid points as the other methods, this bilinear form only has dimensions of the number of interior points and boundary points. Each figure contains the results for grid sizes $m = 8, 16, 24$ and smoothing order $p = 1, 2, 3$, respectively. Looking at Figures 4.7, 4.8, and 4.9, the advantages of any frequency components being negligibly small is still not seen.

Table 4.5: *Numerical Results ('KSS Solve 2') - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 2*'.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	17	5.98E-02	19	3.53E-02	19	2.14E+00
$12^2=144$	24	6.65E-02	39	2.29E-02	49	1.33E-02
$16^2=256$	31	6.06E-02	48	1.05E-02	68	1.97E-01
$20^2=400$	35	9.76E-02	55	5.84E-03	83	2.49E-02
$24^2=576$	44	1.05E-01	62	7.34E-03	117	3.16E-02
Rate of Conv.	-0.51		1.43		3.84	

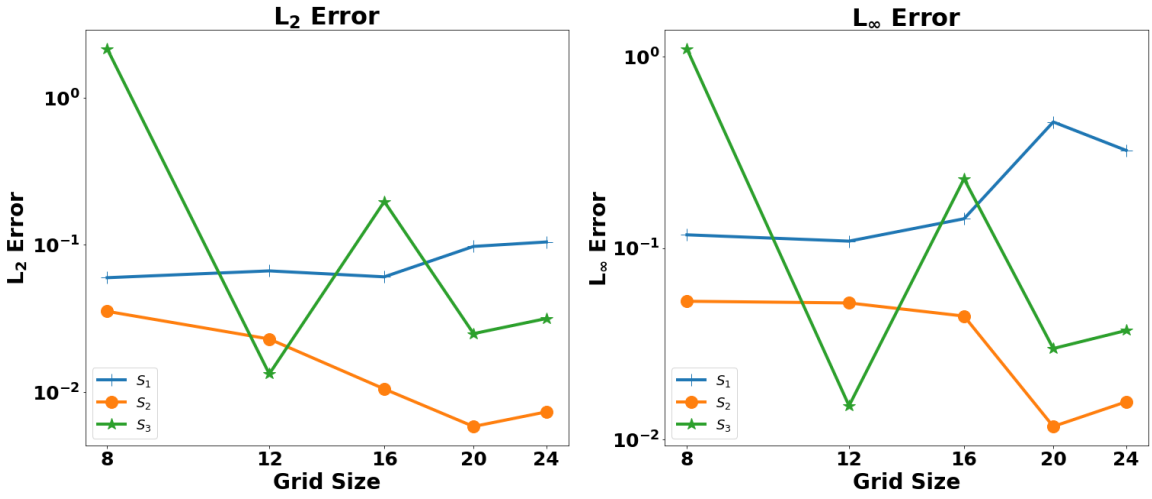
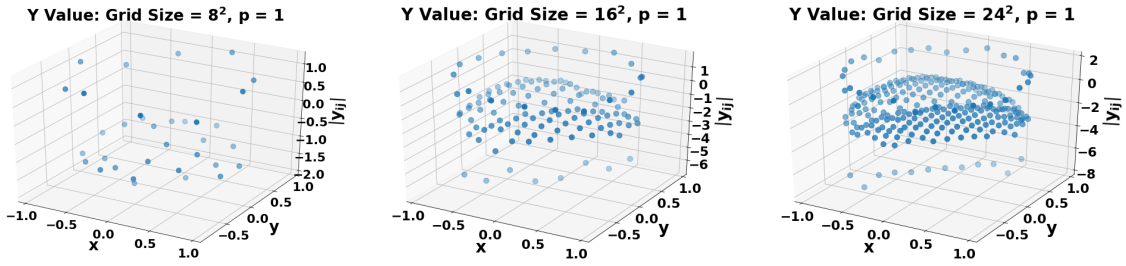


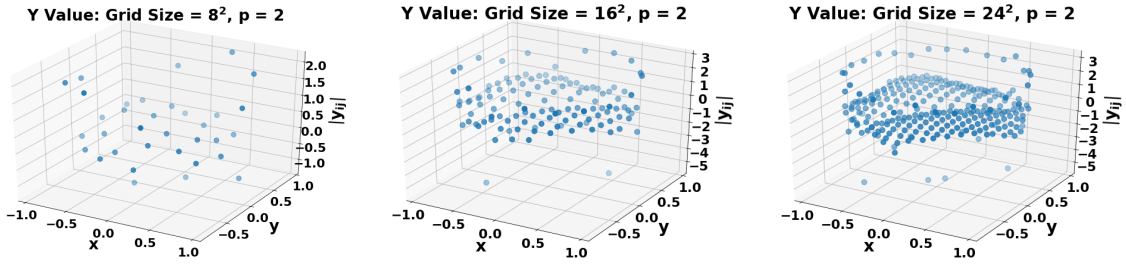
Figure 4.6: Error Plots ('*KSS Solve 2*') - Example 1 - Dirichlet Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.1) computed using '*KSS Solve 2*'. The index of S represents the value of p.



(a) (b) (c)

Figure 4.7: Y Value: '*KSS Solve 2*' with $p = 1$

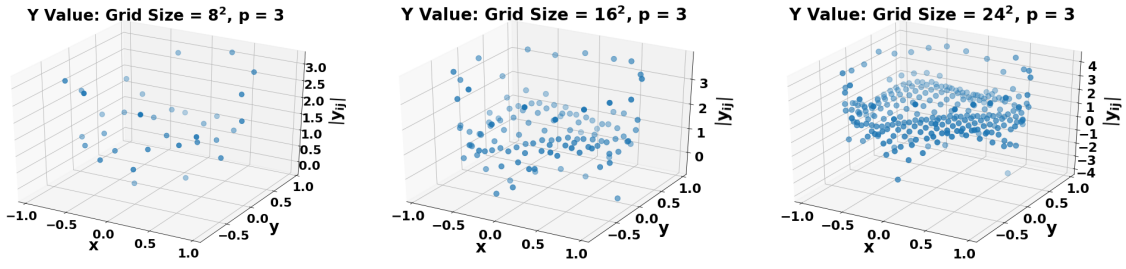
Figures 4.7a, 4.7b, and 4.7c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively. Here, $\mathbf{y} = (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.3) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.8: Y Value: '*KSS Solve 2*' with $p = 2$

Figures 4.8a, 4.8b, and 4.8c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively. Here, $\mathbf{y} = (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.3) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.9: Y Value: '*KSS Solve 2*' with $p = 3$

Figures 4.9a, 4.9b, and 4.9c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively. Here, $\mathbf{y} = (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.3) is used while approximating Equation (4.1).

4.1.3 '*KSS Solve 3*'

The following numerical results are computed using another approach, '*KSS Solve 3*' from Section 3.3, where $\mathbf{y} = (C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}$ in (3.7) is the bilinear form being approximated with KSS. The computed solution for this approach comes after post-processing where $\mathbf{u}_m = \mathfrak{C}^{-1}M\mathbf{y}$. The results come from solving Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. This is the first algorithm in which the coefficients of the solution are computed in a basis of Chebyshev polynomials. The error plots for this approach are shown in Figure 4.10. Looking at the error plots and comparing with the first two KSS approaches, it can be seen that this third approach, '*KSS Solve 3*', has performed the best so far. The numerical results for this third algorithm are shown in Table 4.6. Comparing with the PCG results in Table 4.8, it can be seen that '*KSS Solve 3*' performs better for the lower order smoother $p = 1$ with less iterations and a better convergence rate of 0.96 compared to that of -0.9 for PCG. This KSS approach maintains the same accuracy as PCG for the higher order smoothers. Regardless, with this third algorithm, the number of iterations are still increasing as the smoothing order increases.

The visualizations of the magnitude of the coefficients of Y on a logarithmic scale are examined to see if there is any similar behavior to that of the Fourier case with KSS methods. These visualizations are shown in Figures 4.11, 4.12, and 4.13 for the grid sizes $m = 8, 16, 24$ and order $p = 1, 2, 3$, respectively. The way that the basis of Chebyshev polynomials has been normalized in this third approach seems to have the opposite effect of creating small higher frequency components. The figures show that the higher frequency components have the largest magnitude. Also, as the order increases, the coefficients are increasing for lower-degree Chebyshev polynomials. This does not allow room to neglect any frequencies in order to improve the method.

Table 4.6: *Numerical Results ('KSS Solve 3') - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 3*'.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	12	5.63E-02	14	3.42E-02	17	2.86E-02
$12^2=144$	37	3.40E-02	47	6.91E-03	63	1.60E-03
$16^2=256$	60	2.78E-02	61	3.81E-03	86	3.67E-04
$20^2=400$	60	2.91E-02	77	2.76E-03	90	2.36E-04
$24^2=576$	128	1.97E-02	116	1.39E-03	138	1.23E-04
Rate of Conv.	0.96		2.92		4.96	

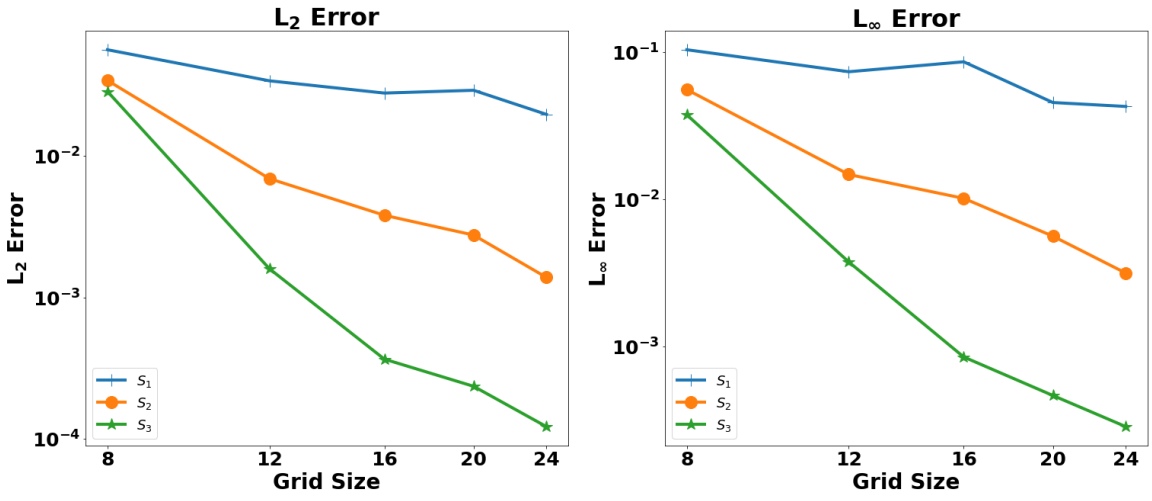
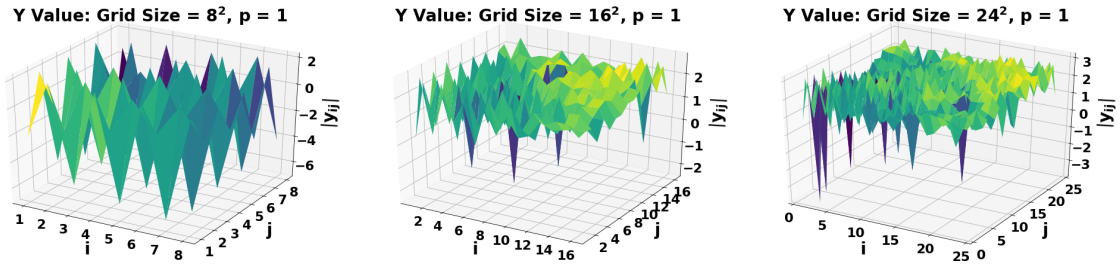


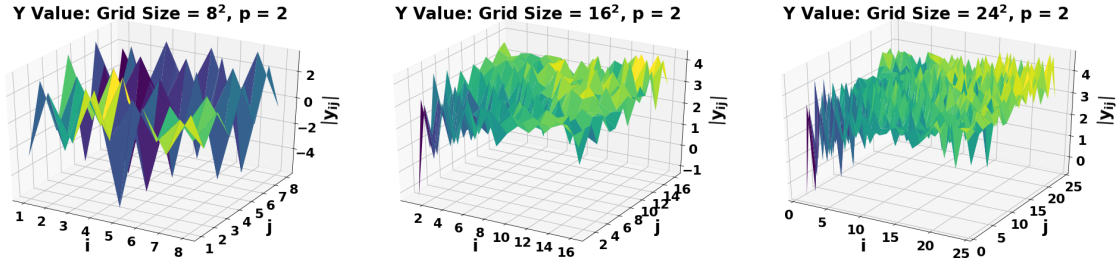
Figure 4.10: Error Plots ('*KSS Solve 3*') - Example 1 - Dirichlet Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.1) computed using '*KSS Solve 3*'. The index of S represents the value of p.



(a) (b) (c)

Figure 4.11: Y Value: '*KSS Solve 3*' with $p = 1$

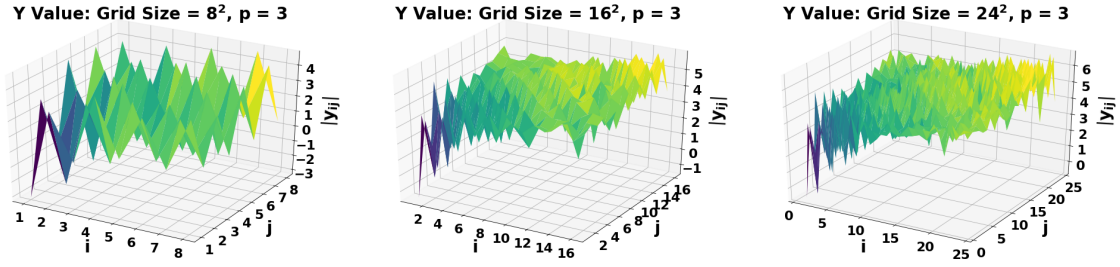
Figures 4.11a, 4.11b, and 4.11c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.7) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.12: Y Value: '*KSS Solve 3*' with $p = 2$

Figures 4.12a, 4.12b, and 4.12c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.7) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.13: Y Value: '*KSS Solve 3*' with $p = 3$

Figures 4.13a, 4.13b, and 4.13c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T)^T(CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.7) is used while approximating Equation (4.1).

4.1.4 '*KSS Solve 4*'

The following numerical results are computed using the final KSS approach, '*KSS Solve 4*' from Section 3.3, where $\mathbf{y} = (C\mathfrak{C}^T M)^T (CS^{-1}C^T)^{-1}\mathbf{b}$ in (3.9) is the bilinear form being approximated with KSS. The computed solution for this approach comes after post-processing where $\mathbf{u}_m = \mathfrak{C}^{-1}\mathbf{y}$. The results come from solving Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. The error plots are shown in Figure 4.14 where the trends and rate of convergence are somewhat similar to that of '*KSS Solve 3*' in Figure 4.10. Looking at the numerical results in Table 4.7, the drastic change in results can be seen when compared to the previous algorithm. The number of iterations considerably decreased while still sustaining the same accuracy. For a small comparison, on the largest grid size with the highest order smoother, the number of iterations for '*KSS Solve 3*' is 138, whereas the number of iterations for '*KSS Solve 4*' is a much smaller 18.

From looking at the numerical results, it can already be seen that this algorithm has some of the same benefits as in the Fourier case for KSS since the number of iterations are decreasing as the smoothing order increases. Recall that the basis of Chebyshev polynomials is normalized differently than in '*KSS Solve 3*'. In this algorithm, the dampening of frequency space has been included inside the bilinear form.

The magnitude of the coefficients of Y on a logarithmic scale are viewed for grid sizes $m = 8, 16, 24$ and order $p = 1, 2, 3$, respectively. The Figures 4.15, 4.16, and 4.17 show optimistic results with similarities to that of the Fourier KSS case. The figures show that as the degree of the Chebyshev polynomials increases, the magnitude of the coefficients are decreasing. Also, as the order p increases, the drop-off in magnitude is more pronounce, or the coefficients decrease faster as p increases. These results leave the question considering if this final approach, '*KSS Solve 4*', can be improved to be more efficient. It may be possible to only compute the lower frequency components and neglect the higher frequency components similar to that in the Fourier cases of KSS methods.

Table 4.7: *Numerical Results ('KSS Solve 4') - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 4*'.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	8	5.64E-02	8	3.41E-02	9	2.86E-02
$12^2=144$	32	3.36E-02	30	6.99E-03	26	1.64E-03
$16^2=256$	42	2.84E-01	37	3.89E-03	22	4.60E-04
$20^2=400$	28	2.93E-02	22	3.13E-03	17	9.39E-04
$24^2=576$	46	2.15E-02	29	2.28E-03	18	5.88E-04
Rate of Conv.	0.88		2.46		3.53	

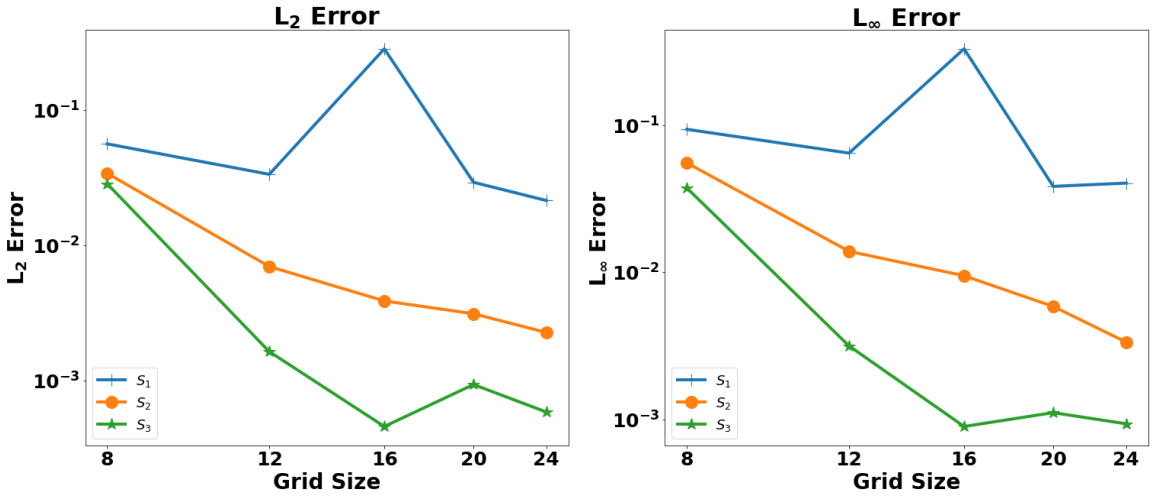
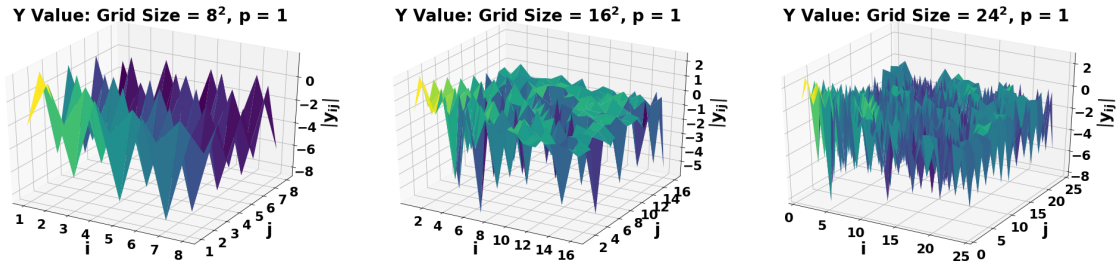


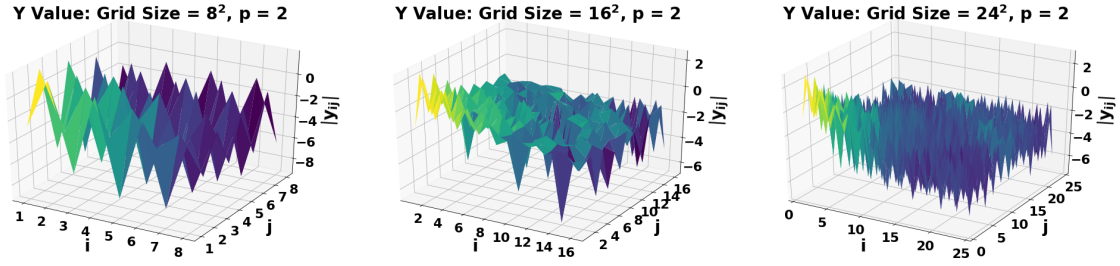
Figure 4.14: Error Plots ('*KSS Solve 4*') - Example 1 - Dirichlet Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.1) computed using '*KSS Solve 4*'. The index of S represents the value of p.



(a) (b) (c)

Figure 4.15: Y Value: '*KSS Solve 4*' with $p = 1$

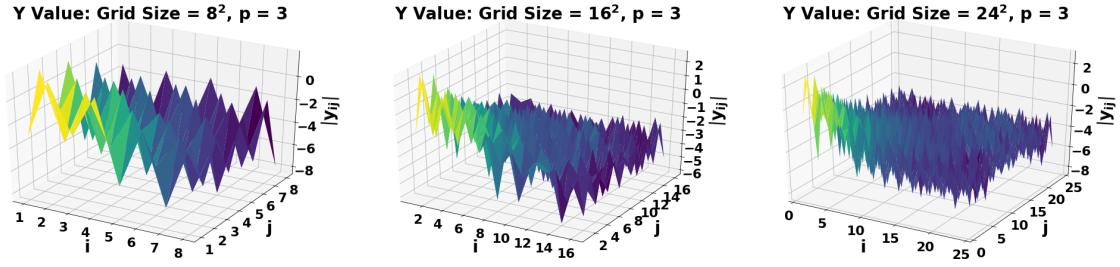
Figures 4.15a, 4.15b, and 4.15c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T M)^T (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.9) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.16: Y Value: '*KSS Solve 4*' with $p = 2$

Figures 4.16a, 4.16b, and 4.16c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T M)^T (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.9) is used while approximating Equation (4.1).



(a) (b) (c)

Figure 4.17: Y Value: '*KSS Solve 4*' with $p = 3$

Figures 4.17a, 4.17b, and 4.17c visualize the coefficients of Y on a logarithmic scale for each grid size $8^2, 16^2, 24^2$, respectively, where i and j correspond to the Chebyshev polynomial degrees in x or y. Here, $\mathbf{y} = (C\mathfrak{C}^T M)^T (CS^{-1}C^T)^{-1}\mathbf{b}$ from (3.9) is used while approximating Equation (4.1).

4.1.5 PCG

These results are computed using the previous PCG method and used to compare with the component-wise KSS approaches. The results come from solving Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. The numerical results are shown in Table 4.8 with the error plots shown in Figure 4.18. By comparing the last algorithm, '*KSS Solve 4*', with the results from PCG, it can be seen that the KSS method has a very close rate of convergence with PCG but with a significantly less number of iterations. Overall, the component-wise approach of the '*KSS Solve 4*' algorithm has maintained the same accuracy as that of PCG.

From the four algorithms provided in this section, '*KSS Solve 4*' performed the best and has the highest chance of optimization in the future. For the rest of the examples in the numerical results, the last algorithm, '*KSS Solve 4*', will be used in comparison with PCG. The simplified term of KSS may also be used when referring to this last algorithm.

Table 4.8: *Numerical Results (PCG) - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using PCG.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	34	5.63E-02	17	3.42E-02	19	2.86E-02
$12^2=144$	75	3.21E-02	36	6.91E-03	28	1.59E-03
$16^2=256$	123	3.01E-02	39	3.81E-03	30	3.65E-04
$20^2=400$	188	3.85E-02	41	2.76E-03	31	2.36E-04
$24^2=576$	269	1.51E-01	69	1.37E-03	49	1.21E-04
Rate of Conv.	-0.9		2.93		4.97	

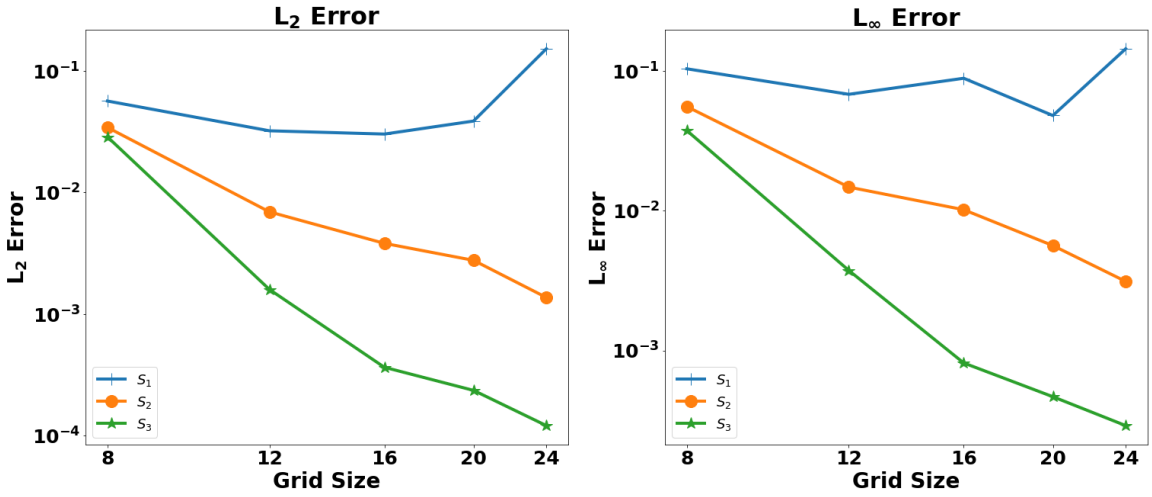


Figure 4.18: Error Plots (PCG) - Example 1 - Dirichlet Problem on Ω_1

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.1) computed using PCG. The index of S represents the value of p.

4.2 Example 1 - Dirichlet Problem on Ω_2

For an example on a more complex domain, we consider the star-shaped domain Ω_2 shown discretized in Figure 4.1b. Recall that this domain is also referred to as a flower-shape with five petals of the form

$$\Omega_2 = \{(r,\theta) | r < 0.75(1+0.2\cos(5\theta))\}.$$

Using this more complex domain, we have the following example of a Dirichlet BVP,

$$\begin{cases} -\Delta u = -6x+6y & \text{in } \Omega_2 \\ u = x^3 - y^3 & \text{on } \Gamma = \partial\Omega_2. \end{cases} \tag{4.2}$$

The numerical results in this section are computed using '*KSS Solve 4*' and PCG. It is a little difficult to compare with this example on the disc-shaped domain (4.1) since the number of grid points used has increased and the smoothing order $p = 4$ has been included. For the grid sizes and smoothing orders that are in common, the error plots in Figures 4.19 and 4.20 for KSS and PCG, respectively, have the same trends as the error plots for Ω_1, with the disc-shaped domain having a slightly smaller error. The numerical results for KSS are shown in Table 4.9 and the numerical results for PCG are shown in Table 4.10. Although both methods still perform well overall for this star-shaped domain example, the number of iterations needed for this domain has, to some extent, increased. When comparing between KSS and PCG, '*KSS Solve 4*' has a higher convergence rate for the lower order smoother, while having close to the same convergence rate for $p = 2$. For the higher order smoothers, PCG achieves a smaller error but with a significantly larger amount of iterations needed. '*KSS Solve 4*', on the other hand, produces accurate results for any order smoother with a very low number of iterations.

Table 4.9: *Numerical Results ('KSS Solve 4') - Example 1 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.2) computed using '*KSS Solve 4*'.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	14	8.92E-02	13	4.30E-02	13	3.22E-02	14	2.72E-02
$16^2=256$	33	4.67E-02	32	1.01E-02	30	3.35E-03	24	2.37E-03
$24^2=576$	57	5.66E-02	48	7.55E-03	37	1.71E-03	28	9.58E-04
$32^2=1024$	69	3.73E-02	51	2.40E-03	35	2.63E-03	27	7.19E-04
$40^2=1600$	74	4.34E-02	49	1.84E-03	32	2.75E-03	25	8.59E-04
Rate of Conv.	0.45		1.96		1.53		2.15	

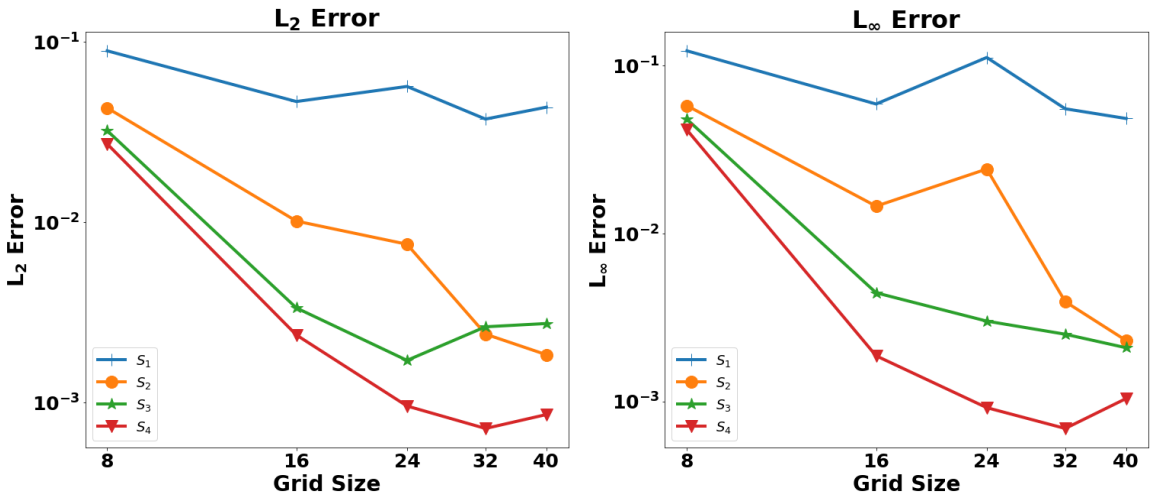


Figure 4.19: Error Plots ('*KSS Solve 4*') - Example 1 - Dirichlet Problem on Ω_2
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.2) computed using '*KSS Solve 4*'. The index of S represents the value of p.

Table 4.10: *Numerical Results (PCG) - Example 1 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.2) computed using PCG.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	25	9.12E-02	21	4.30E-02	22	3.22E-02	25	2.72E-02
$16^2=256$	72	7.47E-02	36	9.94E-03	47	2.81E-03	72	7.39E-04
$24^2=576$	159	8.08E-02	53	7.35E-03	83	1.09E-03	159	1.67E-04
$32^2=1024$	264	6.50E-01	51	1.88E-03	100	2.07E-04	235	2.72E-05
$40^2=1600$	402	9.77E-01	61	1.06E-03	124	9.63E-05	319	9.26E-06
Rate of Conv.	-1.47		2.3		3.61		4.96	

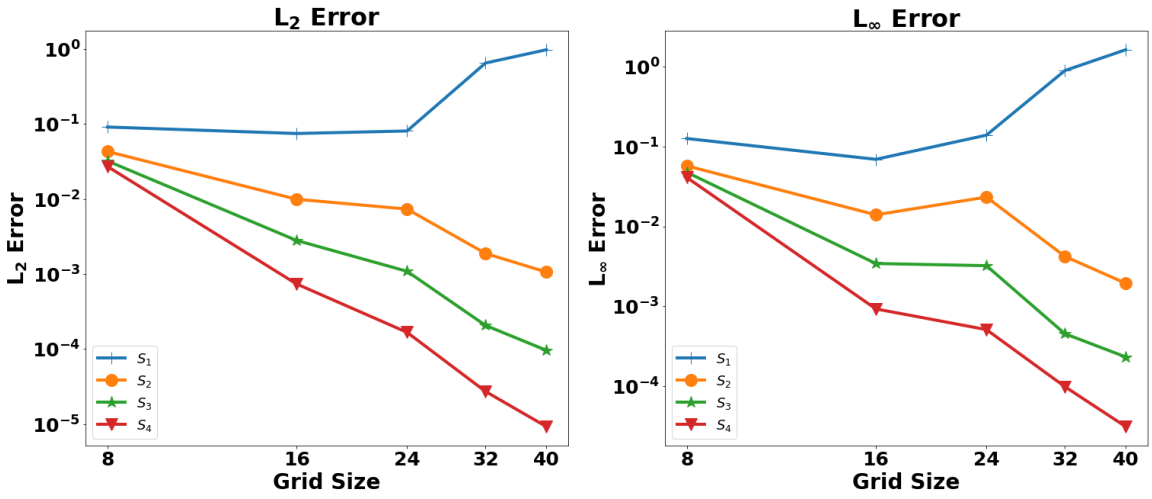


Figure 4.20: Error Plots (PCG) - Example 1 - Dirichlet Problem on Ω_2
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.2) computed using '*KSS Solve 4*'. The index of S represents the value of p.

4.3 Example 2 - Dirichlet Problem on Ω_1

We now consider a different Dirichlet BVP on the disc-shaped domain Ω_1,

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega_1 \\ u = x^2 - y^2 & \text{on } \Gamma = \partial\Omega_1. \end{cases} \tag{4.3}$$

The numerical results in this section are computed using '*KSS Solve 4*' and PCG. The error plots are shown in Figures 4.21 and 4.22 for KSS and PCG, respectively. The numerical results for KSS are shown in Table 4.11 with the results shown for PCG in Table 4.12. Looking at the tables for this example, '*KSS Solve 4*' produced better results for the lower order smoother for both convergence rate and number of iterations. When comparing the higher order smoothers, the error for KSS does not decrease as quickly as PCG, however, KSS computes an accurate solution with notably less iterations overall. Similarly in other numerical methods, there are trade-offs such as rate of convergence versus a low number of iterations.

Table 4.11: *Numerical Results ('KSS Solve 4') - Example 2 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.3) computed using '*KSS Solve 4*'.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2 = 64$	6	5.88E-02	5	8.81E-03	4	1.92E-03	4	5.49E-04
$16^2 = 256$	21	3.00E-02	11	2.01E-03	8	6.91E-04	7	2.79E-05
$24^2 = 576$	26	2.64E-02	8	4.52E-03	6	5.98E-04	5	9.99E-04
$32^2 = 1024$	23	1.78E-02	6	3.57E-03	4	6.27E-04	4	1.01E-03
$40^2 = 1600$	29	1.80E-02	8	1.69E-03	5	1.42E-04	4	5.33E-05
Rate of Conv.	0.74		1.02		1.62		1.45	

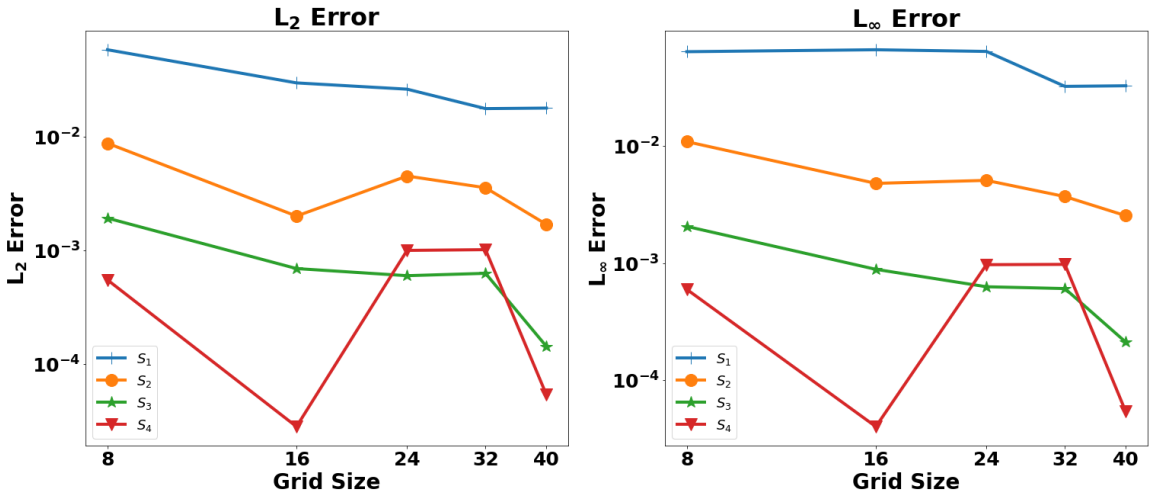


Figure 4.21: Error Plots ('*KSS Solve 4*') - Example 2 - Dirichlet Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.3) computed using '*KSS Solve 4*'. The index of S represents the value of p.

Table 4.12: *Numerical Results (PCG) - Example 2 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.3) computed using PCG.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	34	5.88E-02	17	8.81E-03	15	1.92E-03	17	5.46E-04
$16^2=256$	123	2.88E-02	38	1.58E-03	29	9.71E-05	43	2.84E-06
$24^2=576$	269	4.01E-02	68	5.01E-04	46	2.17E-05	65	7.68E-07
$32^2=1024$	446	8.97E-02	62	3.17E-04	47	8.41E-06	68	2.22E-07
$40^2=1600$	699	1.27E-01	78	2.10E-04	55	3.41E-06	83	4.69E-08
Rate of Conv.	-0.48		2.32		3.94		5.82	

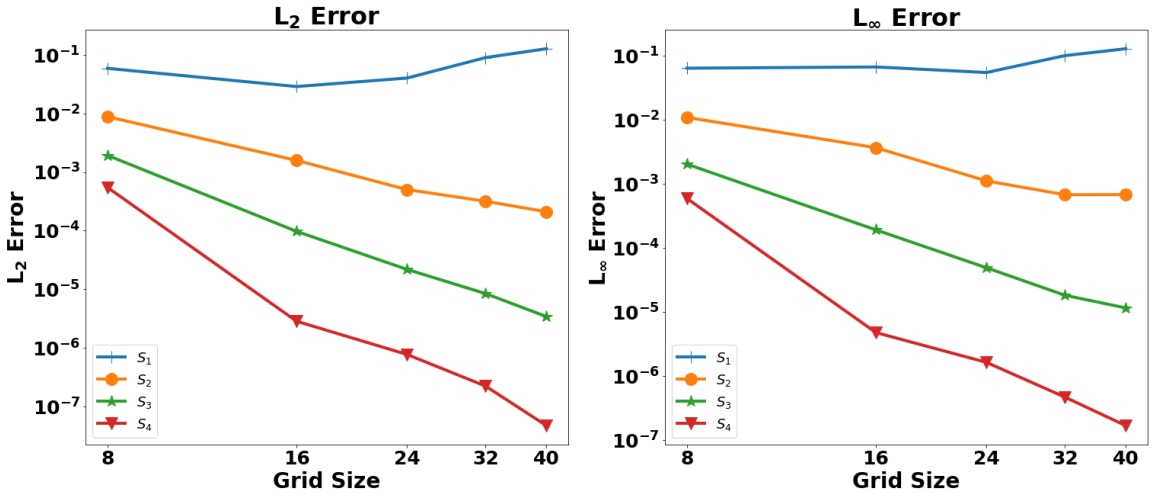


Figure 4.22: Error Plots (PCG) - Example 2 - Dirichlet Problem on Ω_1

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.3) computed using PCG. The index of S represents the value of p.

4.4 Example 2 - Dirichlet Problem on Ω_2

Similar to the previous example, we have the same Dirichlet BVP now considered on the star-shaped domain Ω_2,

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega_2 \\ u = x^2 - y^2 & \text{on } \Gamma = \partial\Omega_2. \end{cases} \tag{4.4}$$

The numerical results in this section are computed using '*KSS Solve 4*' and PCG. The error plots are shown in Figures 4.23 and 4.24 for KSS and PCG, respectively. Looking at the numerical results in Table 4.13 for KSS and Table 4.14 for PCG, it can be seen that the overall rate of convergence is better for PCG. However, '*KSS Solve 4*' still solves the example retaining a small error between 10^{-2} and 10^{-4} with far fewer iterations than PCG. Looking at the last column $p = 4$ in Table 4.13 for KSS, the rate of convergence has decreased compared to the lower order smoothers. This drop in convergence rate brought the idea of looking into the tolerance level that is used for checking convergence in the Block Lanczos algorithm. In Chapter 5.1, we will further investigate the effects of lowering the tolerance level. A simple comparison can also be made with this example on the disc-shaped domain (4.3). For the more complicated star-shaped domain, the total number of iterations increases substantially for PCG while they only increase by a handful for KSS.

Table 4.13: *Numerical Results ('KSS Solve 4') - Example 2 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.4) computed using '*KSS Solve 4*'.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2 = 64$	10	2.33E-02	8	1.05E-02	8	2.45E-03	8	4.97E-04
$16^2 = 256$	19	4.68E-02	14	3.97E-03	11	1.24E-03	10	1.24E-04
$24^2 = 576$	29	4.61E-02	17	4.65E-03	12	4.24E-04	9	1.56E-04
$32^2 = 1024$	33	3.32E-02	17	1.92E-03	12	5.24E-04	9	7.17E-04
$40^2 = 1600$	32	2.79E-02	17	1.85E-03	11	3.53E-04	8	3.02E-04
Rate of Conv.	-0.11		1.08		1.2		0.31	

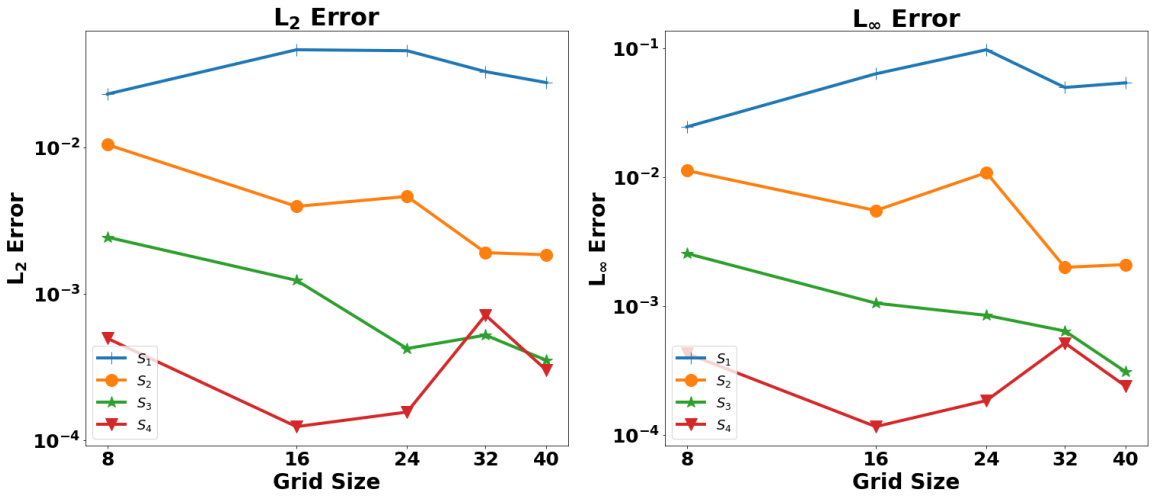


Figure 4.23: Error Plots (‘*KSS Solve 4*’) - Example 2 - Dirichlet Problem on Ω_2
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.4) computed using ‘*KSS Solve 4*’. The index of S represents the value of p.

Table 4.14: *Numerical Results (PCG) - Example 2 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.4) computed using PCG.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	25	8.72E-02	21	1.05E-02	21	2.45E-03	23	4.91E-04
$16^2=256$	72	6.03E-02	35	4.32E-03	46	3.61E-04	72	3.45E-05
$24^2=576$	159	6.00E-02	51	3.89E-03	83	2.45E-04	159	9.42E-06
$32^2=1024$	264	5.43E-02	55	9.45E-04	96	2.99E-05	233	1.02E-06
$40^2=1600$	402	5.32E-02	65	6.68E-04	124	1.86E-05	320	5.57E-07
Rate of Conv.	0.31		1.71		3.03		4.21	

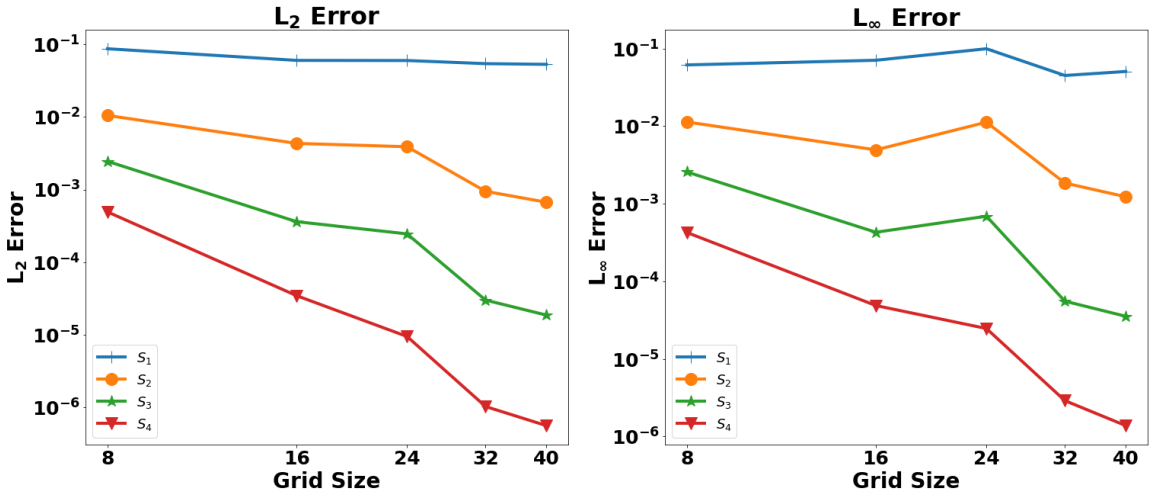


Figure 4.24: Error Plots (PCG) - Example 2 - Dirichlet Problem on Ω_2

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.4) computed using PCG. The index of S represents the value of p.

4.5 Example 2 - Neumann Problem on Ω_1

We now introduce an example with different boundary conditions. For the results in this section we consider a BVP with Neumann boundary conditions on the disc-shaped domain Ω_1,

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega_1 \\ \frac{\partial u}{\partial \nu} = 2(x^2 - y^2) & \text{on } \Gamma = \partial\Omega_1. \end{cases} \tag{4.5}$$

The exact solution is given by $x^2 - y^2$. The numerical results in this section are computed using '*KSS Solve 4*' and PCG. Looking at the error plots in Figure 4.26 and the numerical results in Table 4.16 for PCG, it can be seen that this method performs poorly for this boundary condition. Although KSS performs somewhat better, as seen in the error plots in Figure 4.25 and the numerical results in Table 4.15, implementation of SEEM on a Chebyshev grid does not work well for this example with Neumann boundary conditions. In [1], proper results can be found for this problem using SEEM with a Fourier series. Future work includes trying to improve the numerical results for this example by subtracting $u(0,0)$ to ensure a unique solution.

Table 4.15: *Numerical Results ('KSS Solve 4') - Example 2 - Neumann Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.5) computed using '*KSS Solve 4*'.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2 = 64$	10	2.90E-01	8	4.32E-02	7	9.75E-02	6	1.07E-01
$16^2 = 256$	41	7.40E-01	21	5.02E-02	15	1.02E-01	9	1.08E-01
$24^2 = 576$	71	8.07E-01	24	4.30E-02	12	1.07E-01	6	1.08E-01
$32^2 = 1024$	54	8.95E-01	17	4.35E-02	7	1.08E-01	4	1.08E-01
$40^2 = 1600$	99	8.13E-01	18	8.37E-02	8	1.08E-01	4	1.08E-01
Rate of Conv.	-0.64		-0.41		-0.06		-0.01	

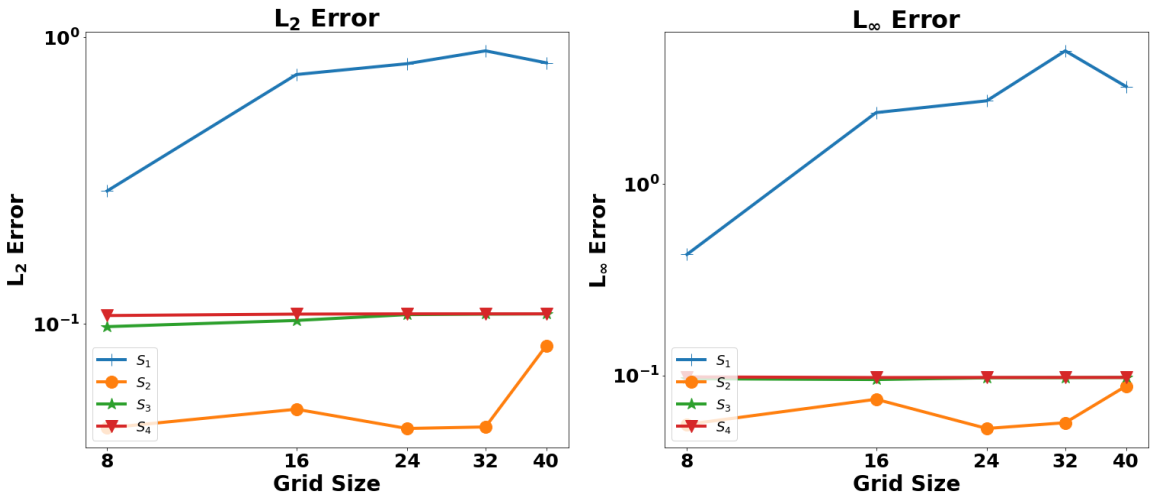


Figure 4.25: Error Plots ('*KSS Solve 4*') - Example 2 - Neumann Problem on Ω_1
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.5) computed using '*KSS Solve 4*'. The index of S represents the value of p.

Table 4.16: *Numerical Results (PCG) - Example 2 - Neumann Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.5) computed using PCG.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	34	4.46E+00	34	1.14E+01	34	1.27E+01	34	1.28E+01
$16^2=256$	123	2.89E+00	123	1.23E+01	123	1.31E+01	123	1.32E+01
$24^2=576$	269	1.86E+00	269	1.23E+01	269	1.29E+01	269	1.29E+01
$32^2=1024$	446	1.00E+00	446	1.19E+01	446	1.28E+01	446	1.28E+01
$40^2=1600$	699	1.15E+00	699	1.25E+01	699	1.27E+01	699	1.27E+01
Rate of Conv.	0.84		-0.06		-0.0		0.0	

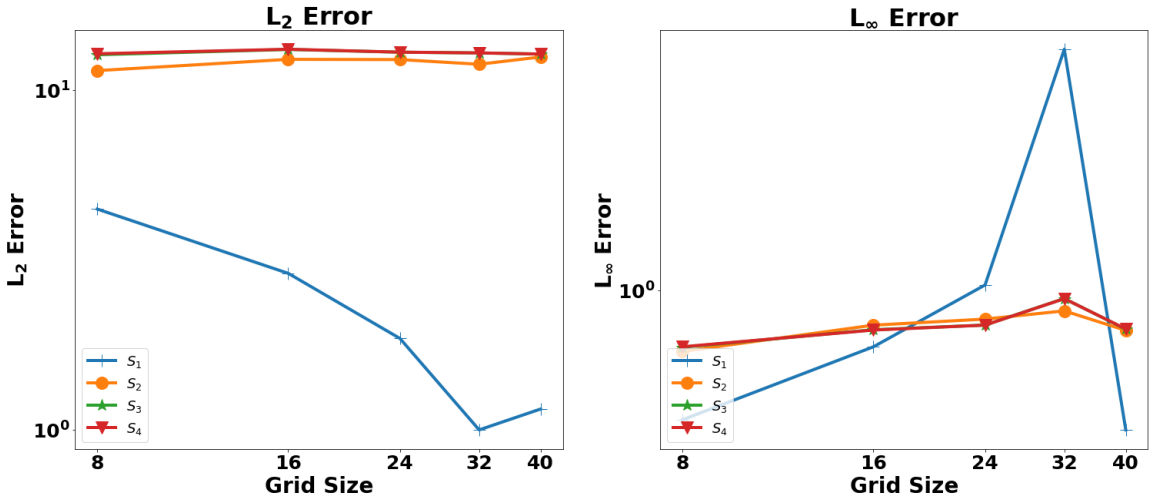


Figure 4.26: Error Plots (PCG) - Example 2 - Neumann Problem on Ω_1

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.5) computed using PCG. The index of S represents the value of p.

4.6 Example 2 - Robin Problem on Ω_2

We now look at an example with a combination of previous boundary conditions. In this section, we consider the BVP with Robin boundary conditions on the star-shaped domain Ω_2,

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega_2 \\ u + \frac{\partial u}{\partial \nu} = \left(x^2 - y^2 + \frac{\partial (x^2 - y^2)}{\partial \nu} \right) \Big|_{\Gamma} & \text{on } \Gamma = \partial \Omega_2. \end{cases} \tag{4.6}$$

The exact solution is given by $x^2 - y^2$. The numerical results in this section are computed using '*KSS Solve 4*' and PCG. Looking at the error plots for KSS in Figure 4.27, it can be seen that the error decreases as the smoothing order increases. The error is decreasing to less than 10^{-4} on the larger grid sizes for smoothing order $p = 4$. By simply comparing with the error plots for PCG in Figure 4.28, KSS achieves a smaller error and seems to have performed better throughout for this problem. The numerical results are shown in Tables 4.17 and 4.18 for KSS and PCG, respectively. For the lower order smoothers of $p = 1$ and $p = 2$, KSS and PCG have the same convergence rates. KSS, however, has considerably less iterations than PCG. For PCG, the higher order smoothers are not converging, even with the large number of iterations. KSS converges for the higher order smoothers, $p = 3$ and $p = 4$, while decreasing the error and the number of iterations as the smoothing order increases. Overall, '*KSS Solve 4*' performed better than PCG for this example with Robin boundary conditions.

Table 4.17: *Numerical Results ('KSS Solve 4') - Example 2 - Robin Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.6) computed using '*KSS Solve 4*'.

	$p = 1$		$p = 2$		$p = 3$		$p = 4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2 = 64$	12	3.99E-01	9	1.23E-01	8	2.46E-02	9	4.25E-03
$16^2 = 256$	25	6.85E-01	16	1.33E-01	15	7.44E-03	11	5.12E-04
$24^2 = 576$	41	6.61E-01	19	5.44E-02	14	3.39E-04	11	6.12E-05
$32^2 = 1024$	52	8.26E-01	19	4.22E-02	13	6.35E-04	8	5.06E-05
$40^2 = 1600$	64	7.94E-01	19	2.39E-02	12	5.59E-04	7	1.23E-04
Rate of Conv.	-0.43		1.02		2.35		2.2	

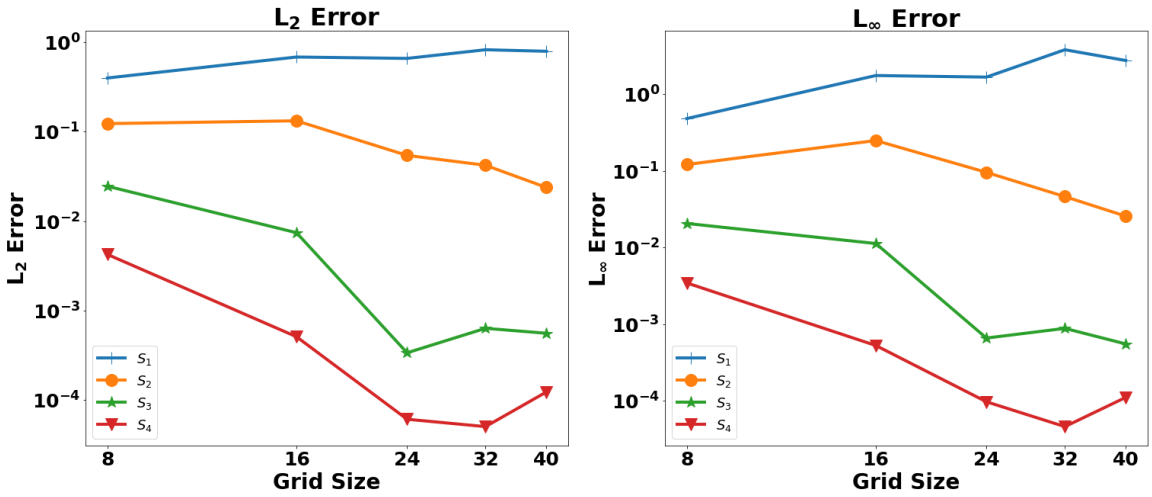


Figure 4.27: Error Plots (‘*KSS Solve 4*’) - Example 2 - Robin Problem on Ω_2
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.6) computed using ‘*KSS Solve 4*’. The index of S represents the value of p.

Table 4.18: *Numerical Results (PCG) - Example 2 - Robin Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.6) computed using PCG.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	25	3.99E-01	25	1.23E-01	25	2.47E-02	25	4.10E-03
$16^2=256$	72	6.85E-01	72	1.33E-01	72	1.15E-02	72	1.53E-02
$24^2=576$	159	6.61E-01	159	5.45E-02	159	1.54E-02	159	4.43E-03
$32^2=1024$	264	8.27E-01	264	4.29E-02	264	1.38E-02	264	4.65E-02
$40^2=1600$	402	7.94E-01	402	2.35E-02	402	7.21E-02	402	5.73E-03
Rate of Conv.	-0.43		1.03		-0.67		-0.21	

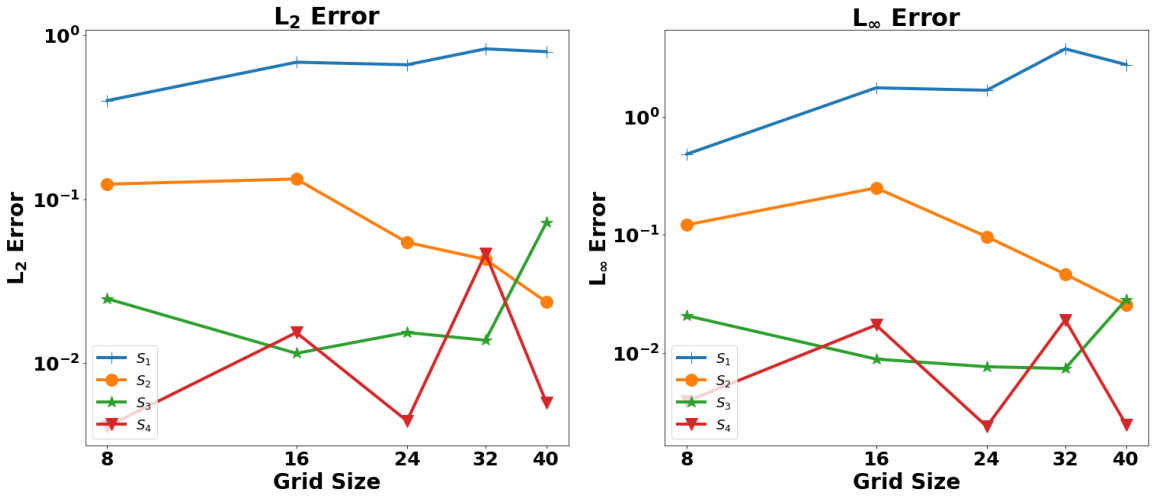


Figure 4.28: Error Plots (PCG) - Example 2 - Robin Problem on Ω_2

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution for Equation (4.6) computed using PCG. The index of S represents the value of p.

4.7 Random Example 3

The following numerical results are computed using random functions. These random examples are created using the following approach.
For each grid size:

- Choose order of decay ($d = 2, 4, 6$)
- Create random Discrete Cosine Transform (DCT) coefficients
- Apply the smoothing operator d number of times
- Apply the Inverse Discrete Cosine Transform (IDCT)

The order of decay determines the smoothness of the function with $d = 6$ being the most smooth function. The random functions are created for the grid sizes of $m = 16, 24, 32, 40$ and each smoothing order $p = 1, 2, 3, 4$. It is important to note that a random function for a certain decay is created for each grid size.

The first random example is created. The functions in Random Example 3 were created using a decay of 2; therefore, these are the least smooth functions of the random examples. The L^2 error plot for '*KSS Solve 4*' in Figure 4.29 shows convergence for each smoothing order in this example with the error around 10^{-1}. Similar results are shown on the L^2 error plot in Figure 4.30 for the higher order smoothers of PCG. In PCG, the error for $p = 1$ tends to grow for the larger grid sizes ($m > 24$). The numerical results are shown in Tables 4.19 and 4.20 for KSS and PCG, respectively, where the convergence rate is around 0.86 for each smoothing order of KSS and for the higher order smoothers for PCG. It is important to note that the rate of convergence in the random examples differs from previous examples since a random function of a certain decay is created for each grid size. For $p = 2, 3, 4$, KSS and PCG achieve approximately the same accuracy and also skew towards the same number of iterations. For this random example 3, KSS performs better than PCG for the lower order smoother $p = 1$ with a convergence rate of 0.86 compared to 0.01 for PCG, while also maintaining a lower number of iterations.

Table 4.19: *Numerical Results ('KSS Solve 4') - Random Example 3.* The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using '*KSS Solve 4*' for the random example 3.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2 = 256$	43	5.71E-01	29	5.51E-01	27	5.53E-01	31	5.65E-01
$24^2 = 576$	83	3.12E-01	52	3.14E-01	51	3.15E-01	61	3.17E-01
$32^2 = 1024$	95	3.02E-01	50	2.79E-01	62	2.84E-01	84	2.87E-01
$40^2 = 1600$	106	2.61E-01	37	2.52E-01	63	2.53E-01	102	2.55E-01
Rate of Conv.	0.86		0.85		0.85		0.87	

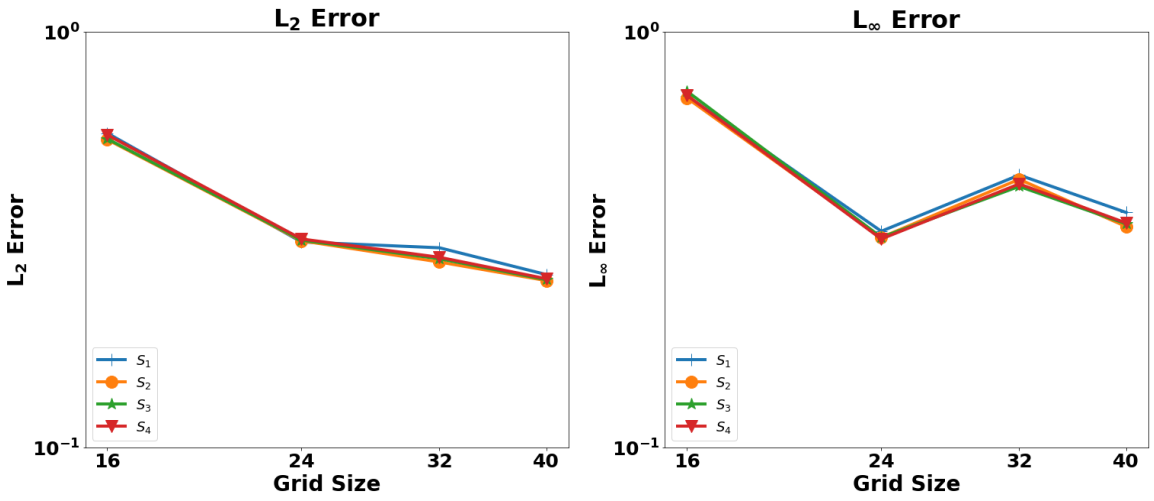


Figure 4.29: Error Plots ('*KSS Solve 4*') - Random Example 3
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using '*KSS Solve 4*' for the random example 3. The index of S represents the value of p.

Table 4.20: *Numerical Results (PCG) - Random Example 3*. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using PCG for random example 3.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2 = 256$	114	5.70E-01	34	5.51E-01	29	5.53E-01	45	5.65E-01
$24^2 = 576$	255	3.13E-01	55	3.14E-01	44	3.15E-01	65	3.17E-01
$32^2 = 1024$	427	4.33E-01	57	2.87E-01	47	2.85E-01	75	2.87E-01
$40^2 = 1600$	676	5.64E-01	64	2.53E-01	54	2.53E-01	92	2.55E-01
Rate of Conv.	0.01		0.85		0.85		0.87	

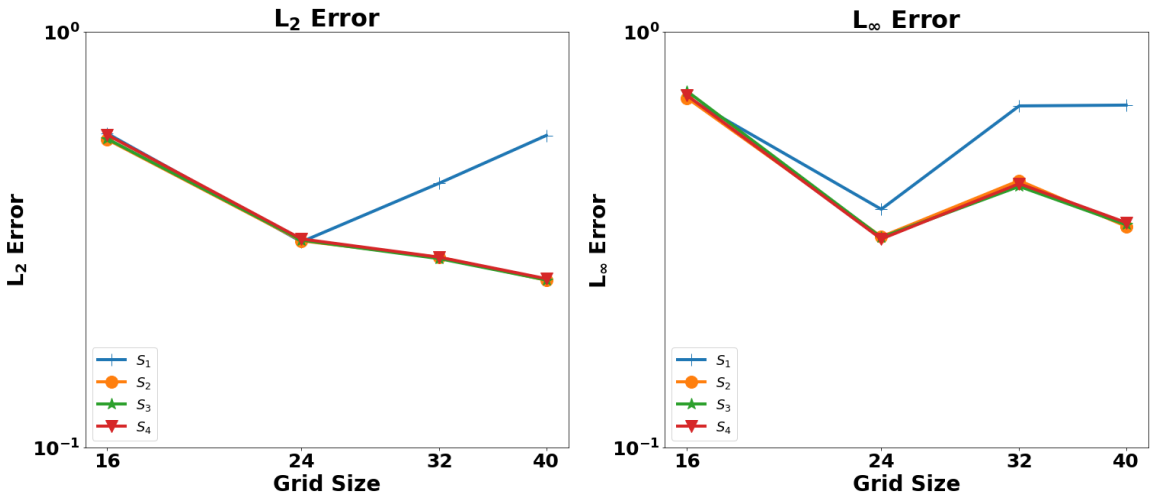


Figure 4.30: Error Plots (PCG) - Random Example 3
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using PCG for the random example 3. The index of S represents the value of p.

4.8 Random Example 4

We consider the next random example. Random Example 4 was created using a decay of 4, creating a smoother function than the previous example. The error plots for KSS and PCG are shown in Figures 4.31 and 4.32, respectively. Looking at the L^2 error plots, it can be seen that for the higher order smoothers, KSS and PCG have very similar graphs and convergence rates with the error around 10^{-2}. Comparably, to the first example, the error for the lower order smoother is smaller for KSS. The numerical results are shown in Table 4.21 for KSS and Table 4.22 for PCG. For the higher order smoothers, $p = 2, 3, 4$, KSS and PCG maintain the same accuracy with a convergence rate around 2.72. KSS, however, has approximately half the number of iterations as PCG. As for the lower order smoother $p = 1$, KSS has a better convergence rate of 2.2 compared to 1.2 for PCG and tremendously less iterations.

Table 4.21: *Numerical Results ('KSS Solve 4') - Random Example 4*. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using '*KSS Solve 4*' for the random example 4.

	$p = 1$		$p = 2$		$p = 3$		$p = 4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2 = 256$	32	2.21E-01	19	2.41E-01	16	2.39E-01	20	2.38E-01
$24^2 = 576$	57	4.56E-02	34	2.89E-02	22	2.89E-02	29	2.90E-02
$32^2 = 1024$	65	2.84E-02	31	1.18E-02	21	1.22E-02	31	1.18E-02
$40^2 = 1600$	61	2.95E-02	27	1.98E-02	22	1.99E-02	31	1.98E-02
Rate of Conv.	2.2		2.73		2.71		2.71	

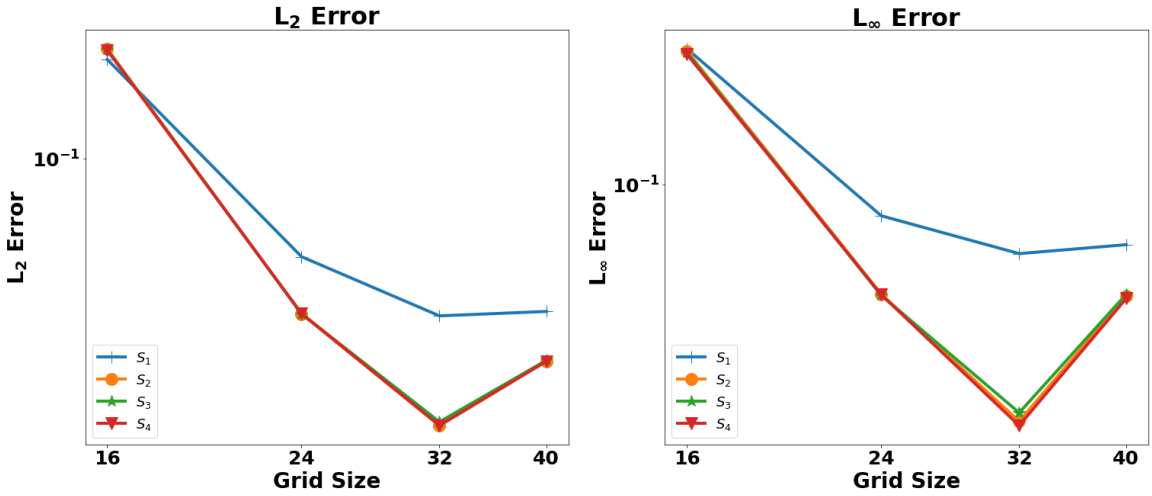


Figure 4.31: Error Plots ('*KSS Solve 4*') - Random Example 4
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using '*KSS Solve 4*' for the random example 4. The index of S represents the value of p.

Table 4.22: *Numerical Results (PCG) - Random Example 4*. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using PCG for the random example 4.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2=256$	114	2.22E-01	34	2.39E-01	28	2.39E-01	42	2.38E-01
$24^2=576$	255	4.47E-02	59	2.89E-02	41	2.89E-02	62	2.89E-02
$32^2=1024$	427	1.14E-01	58	1.18E-02	45	1.18E-02	70	1.18E-02
$40^2=1600$	676	7.37E-02	67	1.98E-02	52	1.98E-02	85	1.98E-02
Rate of Conv.	1.2		2.72		2.72		2.71	

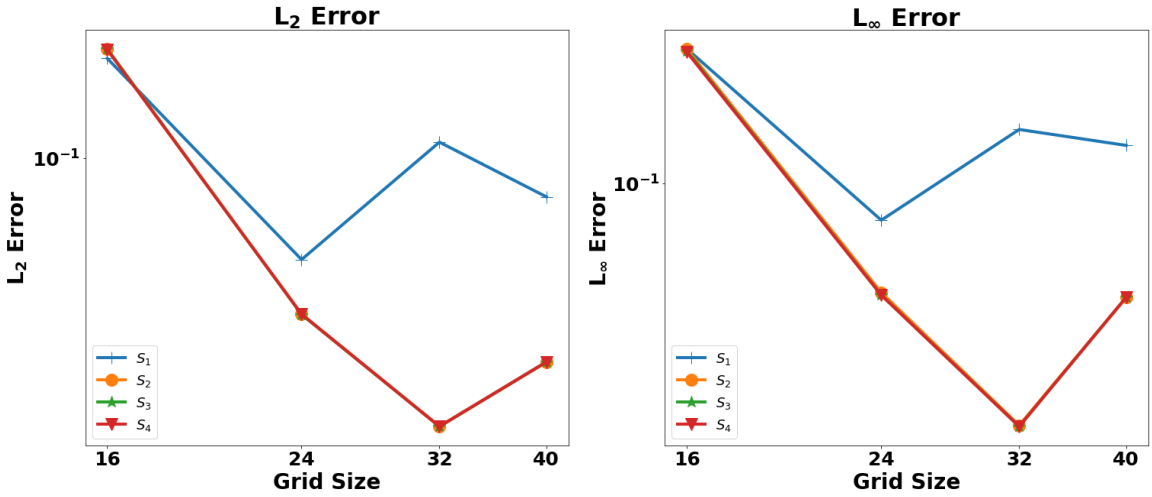


Figure 4.32: Error Plots (PCG) - Random Example 4

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using PCG for the random example 4. The index of S represents the value of p.

4.9 Random Example 5

Lastly, we consider the final example for the numerical results. Random Example 5 was created using a decay of 6. This example is the smoothest of the random functions. The error plots for KSS and PCG are shown in Figures 4.33 and 4.34, respectively. Comparably, with the previous random examples, the trends of the error plots for KSS and PCG are very similar for the higher order smoothers with some error values in this example getting smaller than 10^{-3}. The numerical results are shown in Table 4.23 for KSS and Table 4.24 for PCG. For the higher order smoothers, $p = 2, 3, 4$, KSS maintains the same accuracy as PCG with convergence rates around approximately 6.75. Furthermore, the number of iterations for KSS is significantly lower than the number of iterations for PCG, where the number of iterations needed for KSS decrease as the smoothing order p increases. Again, KSS performs better for the lower order smoother $p = 1$ with a much higher convergence rate of 4.32 compared to the convergence rate of 2.5 for PCG. KSS also achieves this higher rate with far fewer iterations.

Table 4.23: *Numerical Results ('KSS Solve 4') - Random Example 5*. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using '*KSS Solve 4*' for the random example 5.

	$p = 1$		$p = 2$		$p = 3$		$p = 4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2 = 256$	35	1.81E+00	18	1.93E+00	12	1.93E+00	10	1.93E+00
$24^2 = 576$	65	3.15E-02	20	8.01E-04	16	4.46E-04	10	4.31E-04
$32^2 = 1024$	75	2.84E-02	17	4.92E-04	12	3.64E-04	9	3.99E-04
$40^2 = 1600$	36	3.46E-02	18	4.12E-03	12	4.38E-03	8	3.72E-03
Rate of Conv.	4.32		6.71		6.64		6.82	

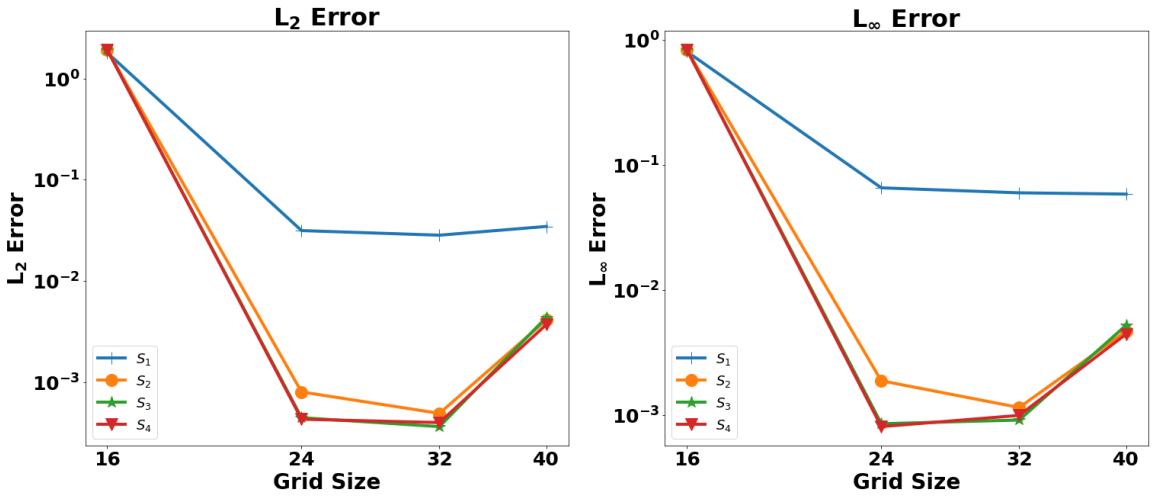


Figure 4.33: Error Plots (‘*KSS Solve 4*’) - Random Example 5
This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using ‘*KSS Solve 4*’ for the random example 5. The index of S represents the value of p.

Table 4.24: *Numerical Results (PCG) - Random Example 5*. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution computed using PCG for the random example 5.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$16^2=256$	114	1.81E+00	35	1.93E+00	26	1.93E+00	37	1.93E+00
$24^2=576$	255	3.06E-02	57	5.28E-04	37	4.24E-04	51	4.24E-04
$32^2=1024$	427	2.90E-02	58	3.85E-04	39	3.67E-04	58	3.67E-04
$40^2=1600$	676	1.83E-01	66	3.90E-03	49	3.85E-03	68	3.85E-03
Rate of Conv.	2.5		6.77		6.79		6.79	

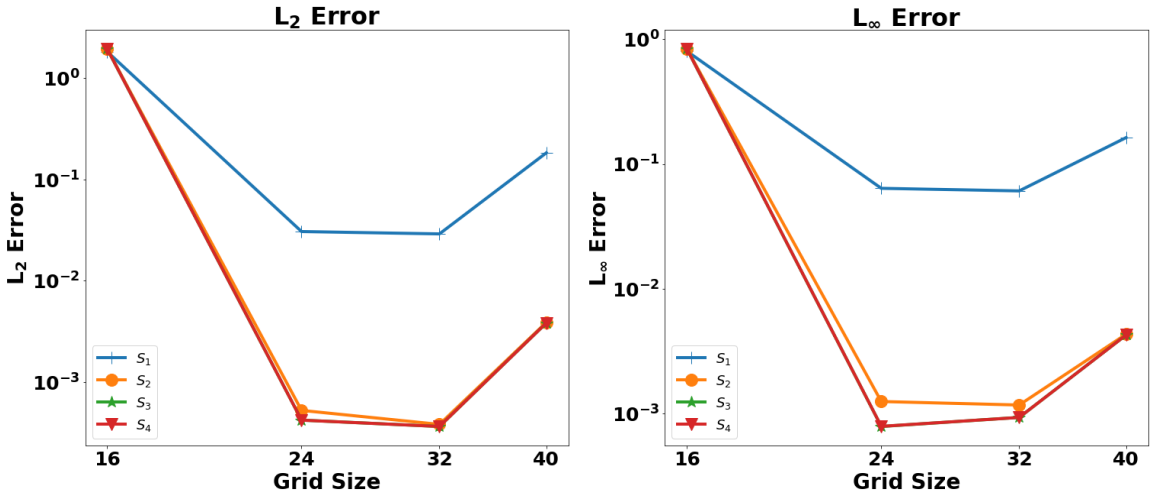


Figure 4.34: Error Plots (PCG) - Random Example 5

This figure contains the error for different order smoothers using the L^2 norm and the L^∞ norm of the approximate solution computed using PCG for the random example 5. The index of S represents the value of p.

Chapter 5

DISCUSSIONS AND CONCLUSION

In this chapter, we will discuss further techniques for optimization and provide a conclusion for the component-wise algorithm. We will view some results that show a brief insight of a couple optimization techniques such as changing the tolerance and eigenvalue behavior. We will also compare the results from the block lanczos algorithm to the non-block lanczos algorithm. We will finish with the conclusion of the component-wise approach.

5.1 Changing Tolerance

In this section, we will look at one possible technique for improving the results. We considered how changing the tolerance level would affect the results. For all numerical results in the previous chapter, a tolerance level of $1e-5$ was used to check convergence in the Block Lanczos algorithm. The tolerance level was lowered to $1e-8$ to see what difference it would make in the numerical results. The following results, shown in Tables 5.1 and 5.2, were computed using the first example (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1.

Looking at the column $p = 3$ in Table 5.1 where the tolerance level is $1e-5$, the rate of convergence is 3.53. By comparing this same column $(p = 3)$ from Table 5.2 where the tolerance level is $1e-8$, it can be seen that the rate of convergence has raised to 4.96. While lowering the tolerance made a decent increase in the rate of convergence, it also increased the number of iterations for each grid size. Again, looking at the last columns $(p = 3)$ for Tables 5.1 and 5.2, the number of iterations has increased drastically. For the tolerance level of $1e-5$ in Table 5.1, the highest number of iterations for a grid size is 26, whereas for the tolerance level of $1e-8$ in Table 5.2, the highest number of iterations for a grid size is 107. For this case, the increase in the rate of convergence may not be worth the increase in number of iterations. This leads to the discussion that there may be a different tolerance level with a better trade-off between improving the rate of convergence while keeping the number of iterations low.

Table 5.1: *Changing Tolerance (*$1e-5$*) - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 4*' with a tolerance level of $1e-5$.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	8	5.64E-02	8	3.41E-02	9	2.86E-02
$12^2=144$	32	3.36E-02	30	6.99E-03	26	1.64E-03
$16^2=256$	42	2.84E-01	37	3.89E-03	22	4.60E-04
$20^2=400$	28	2.93E-02	22	3.13E-03	17	9.39E-04
$24^2=576$	46	2.15E-02	29	2.28E-03	18	5.88E-04
Rate of Conv.	0.88		2.46		3.53	

Table 5.2: *Changing Tolerance (*$1e-8$*) - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 4*' with a tolerance level of $1e-8$.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	16	5.63E-02	15	3.42E-02	16	2.86E-02
$12^2=144$	38	3.42E-02	45	6.91E-03	46	1.60E-03
$16^2=256$	61	2.79E-02	60	3.81E-03	62	3.67E-04
$20^2=400$	74	3.02E-02	64	2.77E-03	59	2.35E-04
$24^2=576$	130	2.12E-02	104	1.37E-03	107	1.23E-04
Rate of Conv.	0.89		2.92		4.96	

We now consider a second example of how changing the tolerance level affects the results. Again, the tolerance level for checking convergence is lowered from $1e-5$ to $1e-8$. The following results, shown in Tables 5.3 and 5.4, were computed using the second example (4.4) with Dirichlet boundary conditions on the star-shaped domain Ω_2.

Looking at the last column $p = 4$ of Table 5.3 where the tolerance level is $1e-5$, the rate of convergence is quite low at 0.31. By comparing this result with the last column $p = 4$ of Table 5.4 where the tolerance level is $1e-8$, it can be seen that the rate of convergence has notably increased to 3.36. Of course, with lowering the tolerance level, the number of iterations has increased. Looking at the same column $(p = 4)$ in Tables 5.3 and 5.4, the highest number of iterations for a grid size in the first table is 10, whereas the highest number of iterations for a grid size in the latter table is 50. Although there is an increase in iterations with the smaller tolerance level, the number of iterations still remain low when compared with PCG (Table 4.14). Overall, for this example with the increase in iterations, the higher rate of convergence may be a better payoff. More results would need to be ran for different types of examples to conclude the best choice of a tolerance level for checking convergence.

Table 5.3: *Changing Tolerance (*$1e-5$*) - Example 2 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.4) computed using '*KSS Solve 4*' with a tolerance level of $1e-5$.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2 = 64$	10	2.33E-02	8	1.05E-02	8	2.45E-03	8	4.97E-04
$16^2 = 256$	19	4.68E-02	14	3.97E-03	11	1.24E-03	10	1.24E-04
$24^2 = 576$	29	4.61E-02	17	4.65E-03	12	4.24E-04	9	1.56E-04
$32^2 = 1024$	33	3.32E-02	17	1.92E-03	12	5.24E-04	9	7.17E-04
$40^2 = 1600$	32	2.79E-02	17	1.85E-03	11	3.53E-04	8	3.02E-04
Rate of Conv.	-0.11		1.08		1.2		0.31	

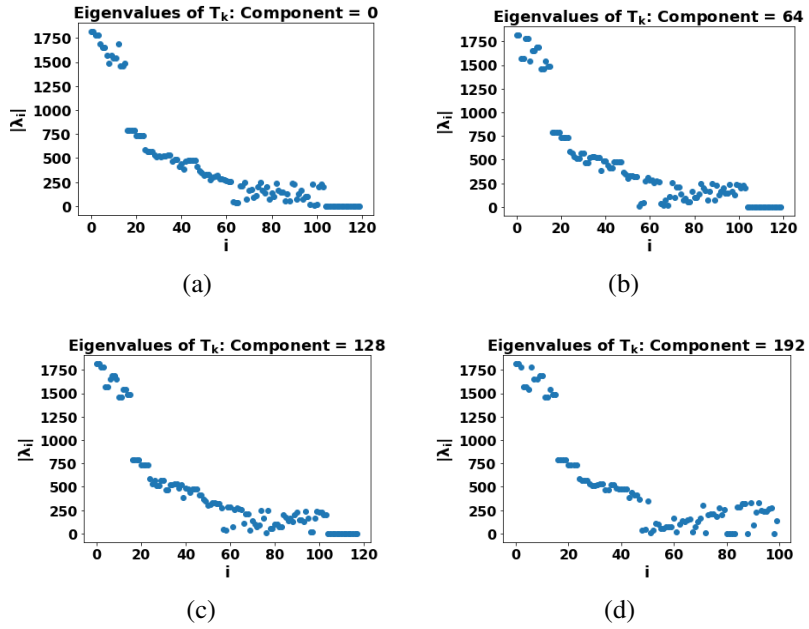
Table 5.4: *Changing Tolerance* ($1e-8$) *- Example 2 - Dirichlet Problem on* Ω_2. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.4) computed using '*KSS Solve 4*' with a tolerance level of $1e-8$.

	$p=1$		$p=2$		$p=3$		$p=4$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	14	2.36E-02	16	1.05E-02	17	2.45E-03	14	4.91E-04
$16^2=256$	35	4.73E-02	31	4.32E-03	35	3.61E-04	31	3.48E-05
$24^2=576$	74	4.51E-02	50	3.89E-03	60	2.44E-04	45	9.50E-06
$32^2=1024$	112	1.99E-02	61	9.53E-04	72	2.97E-05	50	1.52E-06
$40^2=1600$	134	2.44E-02	67	6.68E-04	77	1.94E-05	48	2.21E-06
Rate of Conv.	-0.02		1.71		3.01		3.36	

5.2 Eigenvalue Behavior

In this section, we discuss another approach of possible optimization. In the following figures, the magnitude of the eigenvalues of the block tridiagonal matrix T_k, from the Block Lanczos algorithm, are studied. The idea is to find a trend of the eigenvalues between the orders of the smoothing operator. If there is a trend, the eigenvalues can possibly be modeled by a polynomial without the need to compute each eigenvalue explicitly. Similar research was done with the Fourier KSS method [17]. The numerical results in Figures 5.1, 5.2, 5.3, and 5.4 were computed using Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. These figures are computed on a grid size of 16^2 for each order $p=1,2,3,4$. Only a few components are chosen to give an insight into the behavior of the eigenvalues of the block tridiagonal matrix T_K.

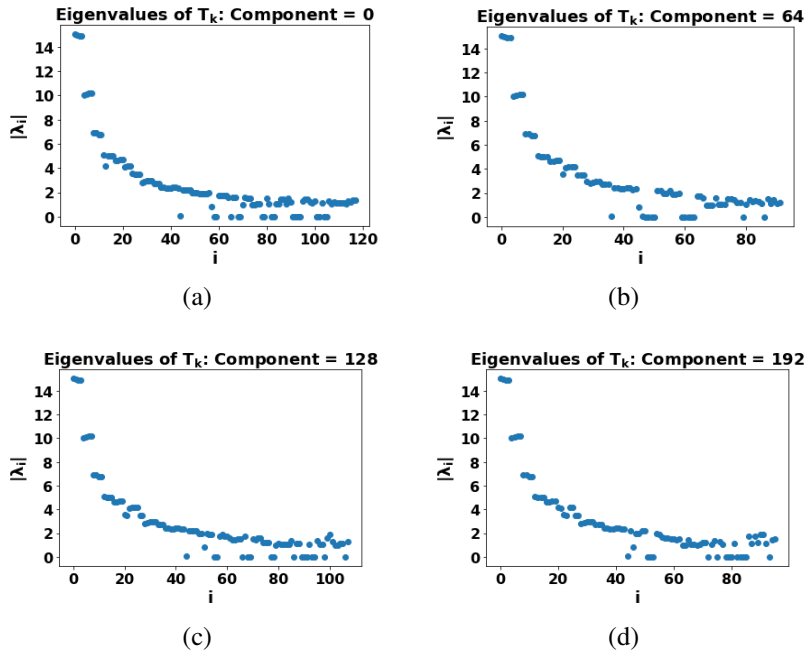
Looking at the Figures 5.1, 5.2, 5.3, and 5.4, it can be seen that as the order p increases, the shape of the graph becomes more prominent. Similar results were seen when looking at the graphs for the star-shaped domain Ω_2. One will notice that the higher order smoothers and higher-degree components have only a handful of iterations. This may be a result of the coefficients of the higher-degree Chebyshev polynomials having a smaller magnitude as seen when visualizing the coefficients of Y in Figures 4.15, 4.16, and 4.17. Although more examples would need to be tested to confirm this hypothesis, the trend shown in this example gives a good prediction that the eigenvalues could be modeled with a polynomial.



(a) (b)

(c) (d)

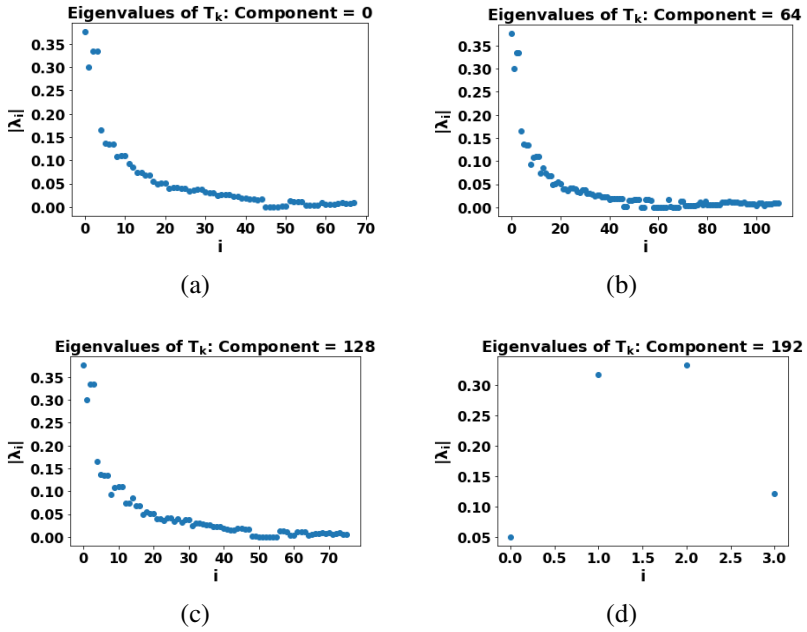
Figure 5.1: Eigenvalues - Example 1 on Ω_1 ($p = 1$)
These figures contain the magnitude of the eigenvalues $|\lambda_i|$ for each index number i. The results are from computing Example 1 (4.1) on the grid size of 16^2 with $p = 1$.



(a) (b)

(c) (d)

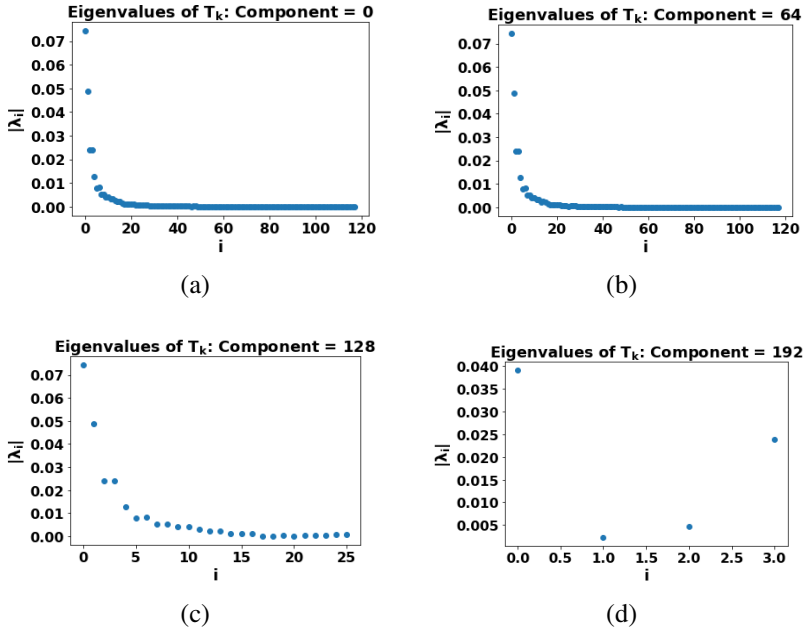
Figure 5.2: Eigenvalues - Example 1 on Ω_1 ($p = 2$)
These figures contain the magnitude of the eigenvalues $|\lambda_i|$ for each index number i. The results are from computing Example 1 (4.1) on the grid size of 16^2 with $p = 2$.



(a) (b)

(c) (d)

Figure 5.3: Eigenvalues - Example 1 on Ω_1 ($p = 3$)
These figures contain the magnitude of the eigenvalues $|\lambda_i|$ for each index number i. The results are from computing Example 1 (4.1) on the grid size of 16^2 with $p = 3$.



(a) (b)

(c) (d)

Figure 5.4: Eigenvalues - Example 1 on Ω_1 ($p = 4$)
These figures contain the magnitude of the eigenvalues $|\lambda_i|$ for each index number i. The results are from computing Example 1 (4.1) on the grid size of 16^2 with $p = 4$.

5.3 Block vs. Non-Block Lanczos

In this section, we show a little motivation for the choice of computing the Block Lanczos algorithm rather than the Non-Block Lanczos algorithm. We will look at the numerical results of one example using both approaches. First, the bilinear form, $\mathbf{u}^T f(A)\mathbf{v}$, for Non-Block Lanczos is computed by

$$\mathbf{u}^T f(A)\mathbf{v} = \frac{1}{4}\left[(\mathbf{u}+\mathbf{v})^T f(A)(\mathbf{u}+\mathbf{v}) - (\mathbf{u}-\mathbf{v})^T f(A)(\mathbf{u}-\mathbf{v})\right].$$

The numerical results to follow were computed using Example 1 (4.1) with Dirichlet boundary conditions on the disc-shaped domain Ω_1. The results from the Block Lanczos algorithm are shown in Table 5.5, whereas the results from the Non-Block Lanczos algorithm are shown in Table 5.6. By comparing the two tables, it can be seen that the Non-Block Lanczos does not provide an appropriate error for the solution; even after a high number of iterations, this approach does not converge. The reason behind this behavior is that the Non-Block Lanczos tends to a different selection of quadrature nodes than that of the Block Lanczos, generally leading to less accuracy [9].

Table 5.5: *Block Lanczos - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 4*' with the Block Lanczos algorithm.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	8	5.64E-02	8	3.41E-02	9	2.86E-02
$12^2=144$	32	3.36E-02	30	6.99E-03	26	1.64E-03
$16^2=256$	42	2.84E-01	37	3.89E-03	22	4.60E-04
$20^2=400$	28	2.93E-02	22	3.13E-03	17	9.39E-04
$24^2=576$	46	2.15E-02	29	2.28E-03	18	5.88E-04
Rate of Conv.	0.88		2.46		3.53	

Table 5.6: *Non-Block Lanczos - Example 1 - Dirichlet Problem on* Ω_1. The results in this table show the average number of iterations and error using the L^2 norm for each grid size and order p, along with the rate of convergence for each order p for the approximate solution of Equation (4.1) computed using '*KSS Solve 4*' with the Non-Block Lanczos algorithm.

	$p=1$		$p=2$		$p=3$	
Grid Size	**# iter**	**error**	**# iter**	**error**	**# iter**	**error**
$8^2=64$	68	5.63E-02	68	3.42E-02	68	2.86E+00
$12^2=144$	150	1.16E+01	150	3.82E+00	150	6.15E+00
$16^2=256$	245	8.84E-02	246	4.20E-02	246	4.18E+01
$20^2=400$	351	2.13E+00	376	4.98E-02	376	1.28E+01
$24^2=576$	520	1.79E+00	533	2.86E-01	538	6.16E+01
Rate of Conv.	-3.15		-1.93		-2.79	

5.4 Conclusion

The applicability of KSS methods has been expanded with the introduction of SEEM. KSS methods were previously only able to be applied on rectangular or box-shaped domains. Now, when combined with SEEM, they can be applied to general domains. Futhermore, when compared to the previous PCG solving method for SEEM, the component-wise KSS approach requires far fewer iterations while maintaining roughly the same accuracy. It is worth noting that SEEM has been applied to a three-dimensional case [2] and that the approach in '*KSS Solve 4*' of choosing the bilinear form does extend for a three-dimensional problem, as well. There are also certain benefits of KSS that can be used to gain advantage over the PCG method. PCG always has the requirement that each component must be computed. Since KSS is a component-wise approach, there are ways to optimize the algorithm and possibly only solve for certain components while approximating for the rest. This idea was discussed briefly in a previous section. There are different techniques of optimization that can improve our method in this way.

One idea comes from a discussion in the previous chapter where the coefficients of the bilinear forms were visualized. With those visualizations for '*KSS Solve 4*', it could be seen that the magnitude of the coefficients of the higher-degree Chebyshev polynomials decreased as the smoothing order p increased. Therefore, it could be possible to neglect these higher-degree Chebyshev polynomials and only compute the lower-degree Chebyshev polynomials,

thus improving the efficiency of this method. More tests would need to be ran and results compared to verify this idea; however, the results shown give a good indication that this same optimization technique used in the Fourier KSS method extends to the Chebyshev grid as well.

The second idea comes from a discussion in a previous section of approximating the eigenvalues of the block triadiagonal matrices, resulting from the Block Lanczos algorithm, with a polynomial. If this is possible, only a few components would need to be computed and a polynomial approximation could be used for the rest. With only a brief look into the eigenvalues, the results shown give an optimistic prediction of being able to apply this approach. Similarly in [17], this technique can be applied in addition with the optimization above of neglecting the higher degree components to further enhance the performance of KSS methods in a Chebyshev basis.

Of course, there are other components of this research that can be evaluated to improve this component-wise method. This includes determining if the tolerance level in the Block Lanczos algorithm is worth adjusting for overall rate of convergence in comparison with the number of iterations. For all numerical results, an average of the number of iterations for each component was used. Another idea is to see if the number of iterations varies based on the degree of the Chebyshev polynomial. It could be possible that the higher frequency components converge more slowly. We previously discussed that these higher frequency components have, so far, shown to be smaller. Therefore, computational expense could possibly be gained by neglecting these values.

In conclusion, this component-wise approach of KSS methods maintained the accuracy needed to compare with the previous PCG method while opening the door to several ways in which it can continue to be optimized. Additionally, incorporating SEEM with KSS extended the bounds of applying KSS methods in general.

Appendix A

Python Code

A.1 Block Lanczos Code

Python Code for Block Lanczos Algorithm:

```
def BLanczos(A,R):
    # QR Factorization:
    x, B = np.linalg.qr(R,mode='reduced')
    #  B_0  for Computing Solution
    B0 = B.copy()
    oldx = np.zeros(x.shape)
    j = 0
    tol = 1
    s=R.shape
    mx=s[0]
    X=x.copy()
    # Stopping Criterion
    # Max Iteration and Tolerance
    while j < mx and tol > 1e-5:
        V1 = A*x[:,0]
        V2 = A*x[:,1]
        V = np.zeros((len(V1),2))
        V[:,0] = V1
        V[:,1] = V2
        M = x.T.dot(V)
        R = V-x.dot(M)-oldx.dot(B.T)
        oldx = x.copy()
        # Save Previous  B
        oldB = B.copy()
        # QR Factorization:
        x, B = np.linalg.qr(R,mode='reduced')
        # Full Reorthogonalization
        x=x-X.dot(X.T.dot(x))
        x, B2 = np.linalg.qr(x,mode='reduced')
        X=np.append(X,x,axis=1)
        if j == 0:
            T = M
        else:
            # Store Previous  B
            # Create New Columns
            newcol = np.zeros((2*j,2))
            # Supdiagonal  =B^T
            newcol[-2:] = oldB.T
            # Create New Rows w/Diagonal
            newrow = np.zeros((2,2*j+2))
```

```
            # Subdiagonal   =B
            newrow[0:2,-4:-2] = oldB
            # Diagonal  =M
            newrow[0:2,-2:] = M
            T = np.append(np.append(T,newcol,axis=1),newrow,axis=0)
        # Column Vector
        e = np.zeros((2*j+2,2))
        e[0,0] = 1
        e[1,1] = 1
        # Solve T*FT=e Instead of Computing Inverse Explicitly
        FT=scipy.linalg.solve(T,e)
        FT_12=FT[0:2,0:2]
        # Alternate Procedure for  FT12
        if j==0:
            Z=np.linalg.inv(M)
            D=M.copy()
            D1=np.linalg.inv(D)
            C=np.eye(2)
        else:
            G=T[2*j:2*j+2,2*j-2:2*j]
            Om=T[2*j:2*j+2,2*j:2*j+2]
            L=G.dot(D1)
            D=Om-G.dot(L.T)
            D1=np.linalg.inv(D)
            C=L.dot(C)
            term=C.T.dot(D1.dot(C))
            Z=Z+term
        FT_12=Z.copy()
        y = B0.T.dot(FT_12.dot(B0))
        if j == 0:
            oldy = 0
        if j > 0:
            # Check Convergence of (1,2) Entry
            tol = abs(y[0,1]-oldy[0,1])
        oldy = y.copy()
        j = j+1
    return y,j
```

A.2 '*KSS Solve 1*' Code

Python Code for '*KSS Solve 1*':

```
    def kss_solve(self,interior,boundary):
            #  f :
            rhs1 = interior(self.gdata.x1[self.gdata.flag],...
                    self.gdata.x2[self.gdata.flag]);
            #  g :
            rhs2 = boundary(self.gdata.b[:,0],self.gdata.b[:,1]);
            #  b=v :
            rhs = np.hstack((rhs1,rhs2));
            # Grid Size  m^2 :
            m = self.gdata.m
```

```
            Y = np.zeros(m**2)
            J = np.zeros(m**2)
            for i in range(m**2):
                e = np.zeros(m**2)
                e[i] = 1
                #  u = C :
                u_vec = self.C(e)
                # *self.C  = C
                l = len(rhs)
                R0 = np.zeros((l,2))
                R0[:,0] = u_vec
                R0[:,1] = rhs
                # Block Lanczos Algorithm:
                y,J[i] = BLanczos(self.MM,R0)
                # Y  = C^T(CS^{-1}C^T)^{-1}b :
                Y[i] = y[0,1]
            Y = np.reshape(Y,(m,m))
            #  u_m = S^{-1}Y :
            u = self.ker(Y)
            # *self.ker = S^{-1}
            # Average Number of Iterations:
            it = np.mean(J)
            return u,it
```

A.3 '*KSS Solve 2*' Code

Python Code for '*KSS Solve 2*':

```
    def kss_solve2(self,interior,boundary):
            #  f :
            rhs1 = interior(self.gdata.x1[self.gdata.flag],...
                    self.gdata.x2[self.gdata.flag]);
            #  g :
            rhs2 = boundary(self.gdata.b[:,0],self.gdata.b[:,1]);
            #  b = v :
            rhs = np.hstack((rhs1,rhs2));
            l = len(rhs)
            Y = np.zeros(l)
            J = np.zeros(l)
            for i in range(l):
                e = np.zeros(l)
                e[i] = 1
                #  u= Standard Basis Vector:
                u_vec = e
                R0 = np.zeros((l,2))
                R0[:,0] = u_vec
                R0[:,1] = rhs
                # Block Lanczos Algorithm:
                y,J[i] = BLanczos(self.MM,R0)
                # Y  = (CS^{-1}C^T)^{-1}b :
                Y[i] = y[0,1]
            #  u_m = S^{-1}C^TY :
```

```
            u = self.ker(self.Ct(Y))
            # *self.ker = S^{-1}
            # *self.Ct = C^T
            # Average Number of Iterations:
            it = np.mean(J)
            return u,it
```

A.4 '*KSS Solve 3*' Code

Python Code for '*KSS Solve 3*':

```
    def kss_solve3(self,interior,boundary):
            #  f :
            rhs1 = interior(self.gdata.x1[self.gdata.flag],...
                    self.gdata.x2[self.gdata.flag]);
            #  g :
            rhs2 = boundary(self.gdata.b[:,0],self.gdata.b[:,1]);
            #  b = v :
            rhs = np.hstack((rhs1,rhs2));
            # Grid Size  m^2 :
            m = self.gdata.m
            Y = np.zeros(m**2)
            J = np.zeros(m**2)
            for i in range(m**2):
                e = np.zeros(m**2)
                e[i] = 1
                #  u = C C^T :
                u_vec = self.C(self.kerC(e))
                # *self.C  = C
                # *kerC  = C^T
                l = len(rhs)
                R0 = np.zeros((l,2))
                R0[:,0] = u_vec
                R0[:,1] = rhs
                # Block Lanczos Algorithm:
                y,J[i] = BLanczos(self.MM,R0)
                # Y  = (C C^T)^T (C S^{-1} C^T)^{-1} b :
                Y[i] = y[0,1]
            Y = np.reshape(Y,(m,m))
            #  u_m = C^{-1} M Y :
            u = self.kerCinvM(Y)
            # *self.kerCinvM = C^{-1} M
            # Average Number of Iterations:
            it = np.mean(J)
            return u,it
```

Python Code for $\mathfrak{C}^T$:

```
    def kerC(self,w,l=None):
        if l == None:
            l = self.l
        w = np.reshape(w,(self.gdata.m,self.gdata.m))
```

```
        w = np.real(dctnT(w))
        # *dctnT = Discrete Cosine Transform Transpose
        return w.flatten()
```

Python Code for $\mathfrak{C}^{-1}M$:

```
    def kerCinvM(self,w,l=None):
        if l == None:
            l = self.l
        w = np.reshape(w,(self.gdata.m,self.gdata.m))
        w = np.real(idctn(w*(1+self.gdata.fx**2 +...
                self.gdata.fy**2)**-l))/...
                (2*self.gdata.m)**2
        # *idctn = Inverse Discrete Cosine Transform
        return w.flatten()
```

A.5 '*KSS Solve 4*' Code

Python Code for '*KSS Solve 4*':

```
    def kss_solve4(self,interior,boundary):
            #  f :
            rhs1 = interior(self.gdata.x1[self.gdata.flag],...
                    self.gdata.x2[self.gdata.flag]);
            #  g  For Dirichlet BC :
            rhs2 = boundary(self.gdata.b[:,0],self.gdata.b[:,1]);
            #  g  For Neumann BC:
#            rhs2 = boundary(self.gdata.bn[:,0],self.gdata.bn[:,1]);
            #  g  For Robin BC:
#            rhs2 = boundary(self.gdata.g_grid)
            #  b=v :
            rhs = np.hstack((rhs1,rhs2));
            # Grid Size  m^2 :
            m = self.gdata.m
            Y = np.zeros(m**2)
            J = np.zeros(m**2)
            for i in range(m**2):
                e = np.zeros(m**2)
                e[i] = 1
                #  u=C*C^T*M :
                u_vec = self.C(self.kerCM(e))
                # *self.C  =C
                # *kerCM  =C^T*M
                l = len(rhs)
                R0 = np.zeros((l,2))
                R0[:,0] = u_vec
                R0[:,1] = rhs
                # Block Lanczos Algorithm:
                y,J[i] = BLanczos(self.MM,R0)
                # Y  =(C*C^T*M)^T(CS^-1C^T)^-1b :
                Y[i] = y[0,1]
            Y = np.reshape(Y,(m,m))
```

```
            #  u_m = C^{-1} Y :
            u = self.kerCinv(Y)
            # *self.kerCinv = C^{-1}
            # Average Number of Iterations:
            it = np.mean(J)
            return u,it
```

Python Code for $\mathfrak{C}^T M$:

```
    def kerCM(self,w,l=None):
        if l == None:
            l = self.l
        w = np.reshape(w,(self.gdata.m,self.gdata.m))
        w = np.real(dctnT(w*(1+self.gdata.fx**2 +...
                self.gdata.fy**2)**-l))
        # *dctnT = Discrete Cosine Transform Transpose
        return w.flatten()
```

Python Code for $\mathfrak{C}^{-1}$:

```
    def kerCinv(self,w,l=None):
        if l == None:
            l = self.l
        w = np.reshape(w,(self.gdata.m,self.gdata.m))
        w = np.real(idctn(w))/(2*self.gdata.m)**2
        # *idctn = Inverse Discrete Cosine Transform
        return w.flatten()
```

BIBLIOGRAPHY

[1] Agress, D., Guidotti, P.: A Novel Optimization Approach to Fictitious Domain Methods, *arXiv: Numerical Analysis* (2019)

[2] Agress, D., Guidotti, P., Yan, D.: The Smooth Selection Embedding Method with Chebyshev Polynomials, *arXiv: Numerical Analysis* (2019)

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

[4] Glowinski, R., Pan, TW., Periaux, J.: A fictitious domain method for Dirichlet problem and applications. *Computer Methods in Applied Mechanics and Engineering* **111**(3-4) (1994) 283-303.

[5] 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).

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

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

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

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

[17] Montiforte, V. A.: Automatic Construction of Scalable Time-Stepping Methods for Stiff PDES. *Masters Theses* (2018) 343.

[18] 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.

INDEX