University of the Pacific

Senior Thesis

Physics 199

Perfect Discretizations, Renormalization, and Polygonal Regions

Author:
Latimer D. Harris-Ward

Research Advisor:
Dr. Kieran Holland

December 4, 2020

Contents

1 Introduction

Differential operators play important roles in various disciplines and are the stars of ordinary and partial differential equations, which describe phenomena like drug absorption, force fields through electric or gravitational potential, marginal cost in the production of goods, and fluid flow, to name a few. As such, discovering the properties of differential operators and solving differential equations aid in developing a deeper understanding of these concepts and processes, many of which arise in everyday situations. For example, the Navier-Stokes equations describe all kinds of fluid flow and have applications in aerodynamics, blood flow, traffic flow, meteorology, and other disciplines [3]. However, many differential equations do not have explicit analytical formulas, much like certain forms of the Navier-Stokes equations, and not all closed form expressions are useful. This necessitates the use of numerical methods for analysis.

There are many different methods used to solve differential equations numerically. Common numerical schemes include Euler's method, Runge-Kutta Methods, finite differences, the finite element method, spectral methods, and more [7]. Each method has its baggage, which includes computation times and disk-space requirements. However, because numerical methods approximate smooth solutions, one of the more important features is the error of the approximation. Generally, the error of a numerical approximation is a measure of how much the approximation deviates from the smooth solution. Ideally, one would hope that as the approximation improves, the error decreases to zero, which means the numerical approximation converges to the smooth solution in the limit. Unfortunately, some methods converge and others diverge. Euler's method, for example, uses linear interpolation via first order Taylor polynomials to produce approximate solutions to ODE. We obtain the blueprint for this method by solving an initial-value problem of the form:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad t \geq t_0, \quad \mathbf{y}(t_0) = \mathbf{y}_0.$$

The solution is

$$\mathbf{y}(t) = \mathbf{y}(t_0) + \int_{t_0}^{t} \mathbf{f}(s, \mathbf{y}(s))ds \approx \mathbf{y}_0 + (t - t_0)\mathbf{f}(t_0, \mathbf{y}_0),$$

where Euler's method arises from using the left-endpoint rule to approximate the integral, i.e., evaluating the slope $\mathbf{f}(t, \mathbf{y}(t))$ at the left-endpoint t_0 and scaling by the width of the interval $t - t_0$. The iterative form of Euler's method is thus

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n), \quad n = 0, 1, \ldots,$$

where $h > 0$ is the time step and $\mathbf{y}_n$ is the numerical approximation of the exact solution $\mathbf{y}(t_n)$ [7]. Euler's method is a convergent[1] scheme and has an order of $p = 1$, which means $|\mathbf{y}_{n+1} - \mathbf{y}_n| = \mathcal{O}(h^{p+1}) = \mathcal{O}(h^2)$ for all $n = 0, 1, \ldots$ Conversely, the following Adams-Bashforth scheme, a multistep method of order $p = 6$,

$$\begin{aligned}&\mathbf{y}_{n+3} + \tfrac{27}{11}\mathbf{y}_{n+2} - \tfrac{27}{11}\mathbf{y}_{n+1} - \mathbf{y}_n \\ &= h\left[\tfrac{3}{11}\mathbf{f}(t_{n+3}, \mathbf{y}_{n+3}) + \tfrac{27}{11}\mathbf{f}(t_{n+2}, \mathbf{y}_{n+2}) + \tfrac{27}{11}\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) + \tfrac{3}{11}\mathbf{f}(t_n, \mathbf{y}_n)\right],\end{aligned}$$

is not a convergent method, despite its robust appearance [7]. If a method does not converge, we are to avoid it at all costs.

In order to numerically solve differential equations, whose solutions are continuous, we need to employ a discrete domain on which we construct our numerical approximations. If there is no grid, there is no way for us to approximate smooth solutions! Thus, the grid is another necessary component for numerical simulations. Every numerical method used to solve differential equations requires a grid of some kind, whether it is one-dimensional or multi-dimensional. Grids are formed by taking a finite region and separating it into smaller sub-regions that are usually rectangular or triangular in shape. This process is called discretization. As the grid becomes finer, where we can fit an increasing amount of

[1] See [7] for the proof.

sub-regions as they become smaller, the numerical approximation usually improves, but not in all cases. This is one of the motivations for this paper.

In particular, the subject of this paper is the Laplacian and its "perfect" counterpart. The Laplacian appears in many of the partial differential equations mentioned above and describes heat conduction, wave propagation, and transportation. Specifically, we compare the standard Laplacian with the perfect Laplacian in the context of the Poisson equation with trivial boundary conditions, that is,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad u\Big|_{\partial\Omega} = 0. \tag{1.1}$$

To understand what equation (1.1) is saying, we first examine the left-hand side, which is the Laplacian of $u(x,y)$. We treat the Laplacian with care in Section 2, but in short, the Laplacian of u is a scalar-valued function that knows about the average value of u at every point (x,y). In the context of equation (1.1), u is *the unknown* function and f is *the Laplacian* of u, and the task is to reconstruct the unknown function u, as with any differential equation, ensuring it obeys the boundary conditions. As described before, analytic solutions are sometimes difficult, and in some cases impossible, to obtain. Interestingly, numerics can even be used to construct analytic solutions as the limiting behavior is analyzed! So, we rely on the numerics in hopes that we can discover an analytic solution, and if not, we look to visualizations of the solution u to understand its behavior.

Although there are many ways to solve the Poisson equation numerically, we employ the use of finite differences. And from these finite difference formulas, we can develop iterative schemes used to construct the numerical solution. The most common iterative scheme used to solve the Poisson problem, which we will be utilizing, is the Jacobi method. We firstly choose the Poisson equation because it is does not involve time as an additional dimension. This is advantageous because we don't have to store matrices for every time step. Further, we choose the Jacobi method because it is convergent [10] and it will be useful in comparing the standard discrete Laplacian to the perfect Laplacian. The perfect Laplacian is a type of discrete Laplacian that does not have discretization errors [6], which is why it is at the heart

of this paper. This can be invaluable in numerically solving differential equations because this in theory packs up the baggage of discretization error we met earlier and ships it away.

To summarize the motivation of this paper, we care about the perfect Laplacian because it eliminates discretization error. Thus, we can expect exact solutions using the perfect Laplacian, in theory. So, there is one goal with three components: compare and contrast the standard and perfect Laplace operators in order to quantify the differences in i) storage and memory, ii) discretization error, and iii) the order of each solver. In Section 2, we briefly delve into the history of Quantum Field Theory, which is responsible for *renormalization*. Then, we supply a formal treatment of the standard Laplacian, its finite discretization, and the importance of the Poisson equation, after which we return to renormalization and present an informal derivation of the perfect Laplacian[2] and examine its properties. Afterwards, in Section 3, we discuss at length the methods utilized to accomplish the goal, including the systems used, pseudo-code, the algorithm, and more. Lastly, in Section 4, we provide an in depth analysis and discussion of the results, ending with concluding thoughts in Section 5.

2 Background

It is important to examine the foundations of Quantum Field Theory to understand what renormalization is, why we need it, and how it gives rise to the perfect Laplacian. Afterwards, we give a thorough treatment of the standard Laplacian, discuss the Poisson equation, and present an informal derivation of the perfect Laplacian.

2.1 The History of Quantum Field Theory: To Infinity and Beyond

After the rapid rise of Quantum Mechanics (QM) in the early twentieth century, physicists sought to extend the new reality of quantization to more than just energy states and atomic spectra. In 1926, physicists Max Born and Werner Heisenberg, along with mathematician Pascual Jordan, published a paper titled "On quantum mechanics," where they

[2] A formal treatment of the derivation of the perfect Laplacian is avoided because it is beyond the scope of this paper. For more a formal treatment, please see [4, 5, 6].

explored the idea of expressing field quantities in terms of matrices, much like position and momentum [8], dealing with the electromagnetic field in empty space [15]. Two years later, theoretical physicist Paul Dirac published his paper "The quantum theory of the emission and absorption of radiation," where he first used the term "Quantum Electrodynamics" (QED), the first developed Quantum Field Theory (QFT). In his paper, Dirac prescribed a process whereby one could transfer the quantum mechanical theme of discreteness to electromagnetic fields, attempting to and successfully describing the process of the spontaneous emission of electromagnetic radiation, where photons are created from electronic transitions in atoms. Dirac's result would ultimately describe photons as elements of the quantized electromagnetic field, where photons are freely created or destroyed [8, 15]. This had important implications because the theory of QM could not adequately account for such phenomena due to the relativistic behavior of photons, and thus Quantum Field Theory was born. Ultimately, Wolfgang Pauli's and Heisenberg's joint work in 1929, in addition to Dirac's, would lay the foundations for QFT that would inspire a small-scale scientific revolution [8].

There were additional fascinating papers in response to Dirac's treatment of the electric field. According to Weinberg (1977), P. Jordan and Eugene Wigner in 1928, and then Heisenberg's and Pauli's joint papers in 1929 and 1930 demonstrated an astonishing result for QFT. They demonstrated that material particles, like protons and electrons, could also be understood as the quanta of various fields in the same way Dirac described photons as quanta of the electromagnetic field. So, there would be proton fields and electron fields, meaning that material particles could also be freely created and destroyed [15]. Other important developments in QFT were Enrico Fermi's description of beta-decay (β-decay); the emergence of anti-particles as a natural consequence of the novel field theoretic approach, discovered by Oppenheimer, Wendell Furry, Pauli, and Victor Weisskopf, while surprisingly opposed by Dirac; and Hideki Yukawa's field theoretic approach to describing nuclear forces between protons and neutrons as facilitated by the exchange of scalar particles called mesons. Yukawa further wrote on how the interaction between charged particles could be expressed as the exchange of *virtual photons*—photons that don't have the fundamental energy of a real

photon $E = h\nu$—as opposed to classical electromagnetic fields that interact at a distance [12, 15]. These virtual photons would eventually become the culprits in the emergence of the infinite.

According to M. Kuhlmann (2020), the best way to define QFT is as an extension of QM for the analysis of systems with many particles, with many degrees of freedom, and fields. And with many degrees of freedom, one may encounter dangerous infinities. This is exactly what Oppenheimer faced. In one of his 1930 papers, Oppenheimer was trying to observe an energy shift in atomic orbitals due to an atomic electron's interaction with the electromagnetic field. Just as the exchange of virtual photons between two electrons produces energy, so would a single electron's emission and reabsorption of a virtual photon produce a self-energy [15]. This self-energy was infinite because the electron and virtual photon could share the momentum of the original electron in an infinite number of ways. The self-energy of an electron was a perturbative sum over all the ways in which the momentum could be shared between the two particles, which led to a divergent, infinite sum. Other infinities like these arose and many ideas surfaced as potential solutions to this problem (like negative probabilities and interactions at a distance), but the answer to containing these infinities was not nearly as sophisticated as previously thought [8, 15].

To understand the solution to the infinity problem, we briefly revisit Special Relativity. The energy of an electron is $E = \gamma m_e c^2$, and at rest, $\gamma = 1$. Oppenheimer found that the energy contribution of the an electron's emission and reabsorption of a virtual photon was infinite while the electron was both in motion *and* at rest. This would mean that the known mass of the electron was a combination of it's "bare" mass and its infinite "self-mass," a similar case for its "bare" electric charge and infinite "self-charge," implying the "bare" mass and "bare" charge being infinite as well. Although counter-intuitive, this would not be problematic because of the possibility of the bare mass infinity cancelling out the self-mass infinity. This process of cancelling infinities by absorbing them into a redefinition of a finite number of physical parameters is called *renormalization* [12, 15].

Renormalization became such an important development that different QFTs were iden-

tified based on whether they were renormalizable or not. The theories that would admit a finite number of renormalizable quantities, with corrections for quantities like mass and charge, are called renormalizable theories. There are not very many renormalizable theories, but the simplest theory of photons, electrons, and positrons (QED) is renormalizable [15]. Eventually, extensions of QFT to unify three of the fundamental forces led to the standard model—the brainchild of Sheldon Glashow, Abdus Salam, and Steven Weinberg—which is the foundation of modern particle physics [9]. The standard model successfully unified the electromagnetic, weak, and strong forces[3], or interactions, and current efforts seek to incorporate gravitational force. Because gravitation is a consequence *of* space-time as opposed to taking place *in* space-time, like the other three forces, the quantization of gravity remains a task so great that not even renormalization can tame it. Before informally discussing renormalization and the "derivation" of the perfect Laplacian, we formally scrutinize and build an intuition for the standard Laplacian.

2.2 The Standard Laplacian

The Laplacian, denoted as $\nabla \cdot \nabla$, ∇^2, or Δ, is a differential operator defined by taking the divergence of the gradient of a scalar-valued function. Formally, suppose we have a mapping $f\colon \Omega \to \mathbb{R}$ where $\Omega \subset \mathbb{R}^n$ and $f \in C^k(\Omega)$, that is, f is k-times differentiable. The Laplacian is a mapping $\Delta : C^k(\Omega) \to C^{k-2}(\Omega)$, where the resultant function is $(k-2)$-times differentiable [13]. This means f must be at least twice differentiable. Generally, the Laplacian in Euclidean space[4] is given by

$$\Delta f = \nabla^T(\nabla f) = \begin{bmatrix} \frac{\partial}{\partial x_1} & \cdots & \frac{\partial}{\partial x_n} \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \cdots + \frac{\partial^2 f}{\partial x_n^2}.$$

[3]The electromagnetic force deals with interactions between charged particles, the weak force is associated with radioactive decay, and the strong force is responsible for holding the atomic nucleus together.

[4]We will assume hereafter that we are working in $\mathbb{R}^n$.

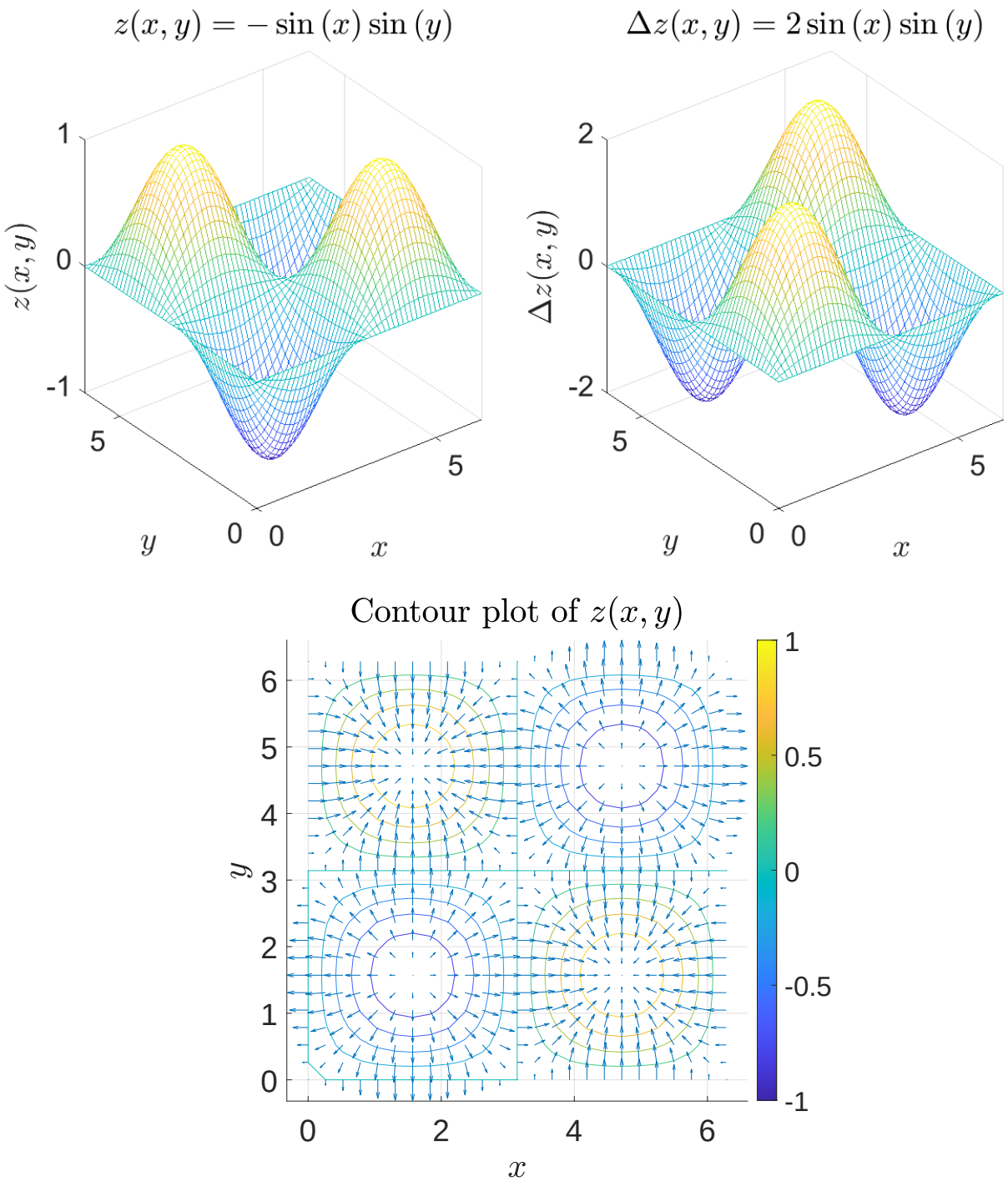


Figure 1: The top figure plots the function $z(x,y) = 2\sin(x)\sin(y)$ and its Laplacian $\Delta z(x,y) = -\sin(x)\sin(y)$. The corresponding contour plot of z is a clear visualization of how the Laplacian is the divergence of the gradient of z.

Algebraically, the Laplacian $\Delta f(p)$ of a function f at a point p measures the average value of f over small spheres around p [1]. For an intuitive understanding, we return to the definition at the beginning of the paragraph. The gradient of f is a vector field describing the direction in which f undergoes the greatest rate of increase[5] [11]. The divergence measures how much a vector field spreads away from a point. Thus, composing the latter with the former, the Laplacian at a point is a local measure of how much the direction of greatest increase of f spreads away from a point, or rather, how much f increases away from a point. We can see this behavior in Figure 1 above, which contains a plot of the function $z(x, y) = -\sin(x)\sin(y)$, its Laplacian $\Delta z = 2\sin(x)\sin(y)$, and its contour plot and gradient. We see the gradient naturally diverges, or spreads, most at local minima because there is an omnidirectional increase in z away from the minimum, giving a positive divergence. Conversely, the gradient diverges least (converges), or concentrates, at local maxima, where there is an omnidirectional decrease in z away from the maximum, giving a negative divergence. Hence, the Laplacian is also a generalization of the measurement of concavity.

To approximate the Laplacian, we utilize finite differences beginning with the one dimensional case. For a map $f\colon \mathbb{R} \to \mathbb{R}$ where $x \mapsto f(x)$, the Laplacian is

$$\Delta f = \frac{d^2 f}{dx^2},$$

which is the second derivative of f with respect to x. Further, we assume f is thrice differentiable. Because the second derivative measures the rate of change of the first derivative, we firstly use finite differences to approximate the first derivative. We can use Taylor expansions[6] to approximate f evaluated at $x + \Delta x$ and $x - \Delta x$:

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{\Delta x^2}{2} f''(x) + \frac{\Delta x^3}{6} f^{(3)}(\xi_+), \quad x < \xi_+ < x + \Delta x, \tag{2.1}$$

$$f(x - \Delta x) = f(x) - f'(x)\Delta x + \frac{\Delta x^2}{2} f''(x) - \frac{\Delta x^3}{6} f^{(3)}(\xi_-), \quad x - \Delta x < \xi_- < x, \tag{2.2}$$

[5]For completeness, when evaluated at a point, the magnitude of the gradient is equal to the rate of increase in that direction.

[6]We can also use Lagrange interpolation.

where the last term in each equation represents the error of the approximation. There are two options that yield the first derivative: solve for $f'(x)$ in equations (2.1) and (2.2) independently, or subtract equation (2.2) from equation (2.1). By doing both, we obtain the following expressions for the approximation of the first derivative:

$$\begin{aligned}
&\textbf{Forward Difference Approximation:} && f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} + \mathcal{O}(\Delta x),\\
&\textbf{Backward Difference Approximation:} && f'(x) \approx \frac{f(x) - f(x-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x),\\
&\textbf{Central Difference Approximation:} && f'(x) \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2).
\end{aligned}$$

If we add equations (2.1) and (2.2) and solve for $f''(x)$, we obtain the finite difference approximation of the second derivative:

$$f''(x) \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{2.3}$$

Extending equation (2.3) to $\mathbb{R}^2$, we approximate the Laplacian as

$$\Delta f \approx \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2} + \frac{f(x, y+\Delta y) - 2f(x,y) + f(x, y-\Delta y)}{\Delta y^2}, \tag{2.4}$$

where

$$\frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2} \approx \frac{\partial^2 f}{\partial x^2}, \tag{2.5}$$

$$\frac{f(x, y+\Delta y) - 2f(x,y) + f(x, y-\Delta y)}{\Delta y^2} \approx \frac{\partial^2 f}{\partial y^2}. \tag{2.6}$$

2.3 The 5-Point Difference Scheme for the Poisson Equation

In order to practically apply equation (2.4) numerically, we need to introduce a grid. Here, we adopt the notation used by A. Iserles in his eighth chapter [7]. The Poisson equation in its fullness demands that we specify the domain of interest and the boundary conditions. Thus, for the purposes of this paper, we are working with the Poisson equation on a square

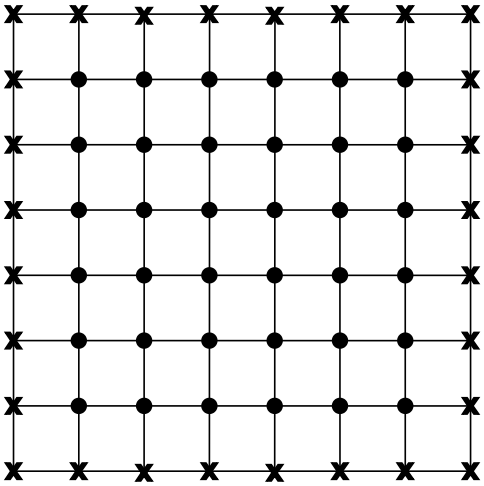


Figure 2: This is what the standard discretization of the square looks like. The interior points are solid circles and the boundary points are marked by crosses.

domain with homogeneous Dirichlet boundary conditions,

$$\begin{aligned} \Delta u &= f, \quad (x,y) \in \Omega, \\ u(x,y) &= 0, \quad (x,y) \in \partial\Omega, \end{aligned} \tag{2.7}$$

where

$$\begin{aligned} \Omega &= \big\{(x,y) \in \mathbb{R}^2 \colon 0 \le x, y \le L\big\}, \\ \partial\Omega &= \big\{(x,y) \in \mathbb{R}^2 \colon x = 0, L,\ y = 0, L\big\}, \end{aligned}$$

and f is a sufficiently smooth function. For the grid, we define $x_k = x_0 + k\Delta x$ and $y_l = y_0 + l\Delta x$, where the starting grid point is $(x_0, y_0) = (0,0)$, the step-size is $\Delta x = L/(N+1)$ and $k, l = 1, 2, \ldots, N$. We can observe what a discretized square looks like in Figure 2. Further, we define $u_{k,l}$ to be the approximation to the solution $u(x_0 + k\Delta x, y_0 + l\Delta x)$ and $f_{k,l}$ to be the right-hand side sampled on the grid $f(x_0 + k\Delta x, y_0 + l\Delta x)$. Now, we utilize equation (2.4) as our scaffolding to write the discrete form of the equation (2.7):

$$\frac{u_{k+1,l} + u_{k-1,l} + u_{k,l+1} + u_{k,l-1} - 4u_{k,l}}{\Delta x^2} = f_{k,l}, \tag{2.8}$$

which we can equivalently express as

$$u_{k+1,l} + u_{k-1,l} + u_{k,l+1} + u_{k,l-1} - 4u_{k,l} = (\Delta x)^2 f_{k,l}. \tag{2.9}$$

Equation (2.8) is what is known as the *five-point* finite difference scheme for the Poisson equation and it approximates the Laplacian Δu at the (k, l)th grid point. We can conveniently express this as a *computational stencil*, shown in Figure 3, which is a pictorial representation of the finite difference scheme [7],

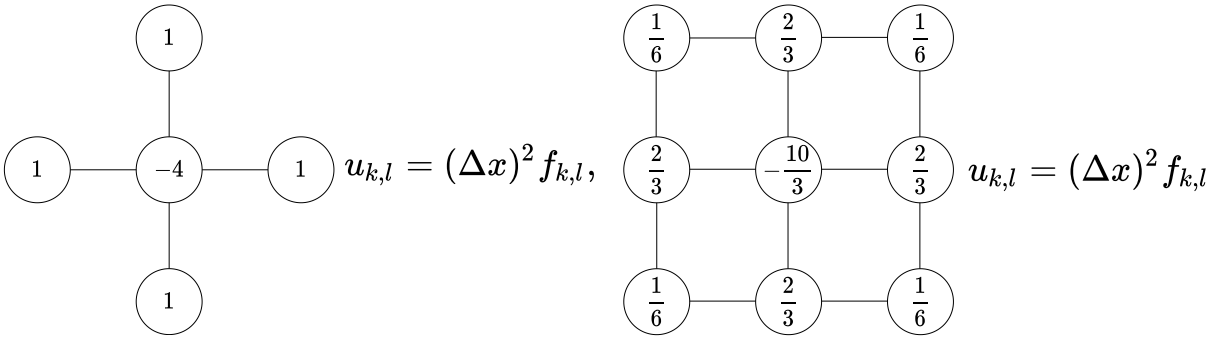


Figure 3: The left is a diagrammatic representation of equation (2.9) and the right the nine-point formula.

hence the designation as the five-point stencil. We also include the *nine-point formula* finite difference scheme by its side. Lastly, we have the *modified nine-point scheme*. Because we will be utilizing modified nine-point method, we have included both stencils, with the modified stencil in Figure 4.

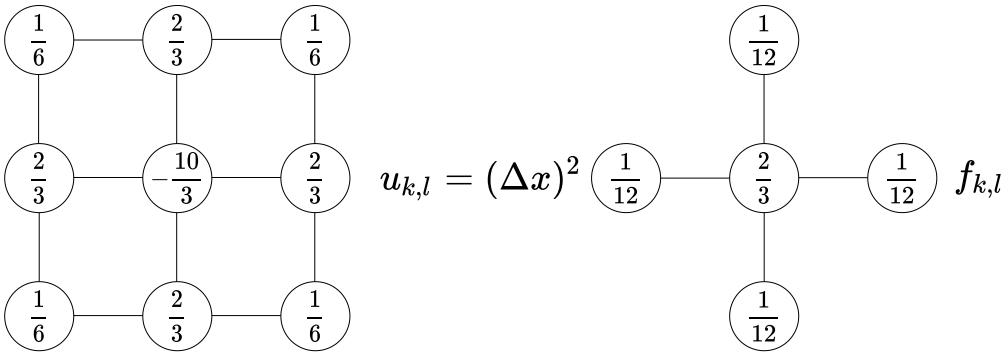


Figure 4: This is a diagrammatic representation of the modified nine-point formula.

Concerning equation (2.7), we are interested in finding the solution u. We can accomplish this numerically using equation (2.9) by isolating $u_{k,l}$, which is the difference between the average of its nearest neighbors and the corresponding right-hand side scaled by a quarter of the square of the step-size. By iterating over each grid point multiple times, we steadily improve the approximation until the algorithm converges. This is called the *Jacobi method* of iteration. However, for now, we examine the importance of the Poisson equation.

2.4 The Poisson Equation

The Poisson equation is one of the most fundamental PDE in classical mechanics and classical electrodynamics, but also has applications in hydrodynamics, diffusion, and any other system describing equilibrium or steady-state behavior. Here, we only consider the Poisson equation that appears in mechanics and electrodynamics. Suppose we have a vector field $f\colon \mathbb{R}^3 \to \mathbb{R}^3$ that is *conservative*. Mathematically, this has three meanings:

1. *the line integral over the field* f between two points P_1 and P_2 has the same value regardless of the path taken from P_1 to P_2,
2. the field f is *the gradient of a scalar-valued function* φ, that is, $f = \nabla\varphi$,
3. *the curl of the field is identically zero*, or $\nabla \times f = 0$,

where each statement is equivalent[7]. Physically, we can interpret statements 1-3 as the following:

1. *the work done in a field* f between any two points P_1 and P_2 has the same value regardless of the path taken from P_1 to P_2,
2. the field f is *the gradient of a scalar-valued potential* φ, that is, $f = \nabla\varphi$, which typically describes the potential energy of a system as a function of position,
3. *the field is not rotational in any way*, or $\nabla \times f = 0$.

[7]We do not present the proof in this paper. To see the relationship between the three statements, see [11].

One more important statement not included above is: *a conservative vector field satisfies the Poisson equation.*

The conservative vector fields we encounter in mechanics and electrodynamics are the gravitational field $\mathbf{g}$ and the electric field $\mathbf{E}$[8]. The reason why the Poisson equation is important stems from the physicists fundamental desire to discern and determine the path of a particle for all times. If we have a potential field $\mathbf{F}$ with data suggesting that its divergence produces a density distribution ρ, that is $\nabla \cdot \mathbf{F} = \rho$, but we do not know the the components of this field, we can write $\nabla \cdot \nabla\varphi = \rho$, which is precisely the Poisson equation. We can then proceed to solve the Poisson equation for the potential function. Once we have the potential function, we can potentially reconstruct the field, determine the force, and (hopefully) deduce the equations of motion governing a particle traversing through the field.

We can also arrive at the Poisson equation using electrodynamics. Maxwell's equations, given by

$$\textbf{Gauss' Law:} \qquad \nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \tag{2.10}$$

$$\textbf{Gauss' Law for Magnetism:} \qquad \nabla \cdot \mathbf{B} = 0, \tag{2.11}$$

$$\textbf{Faraday's Law:} \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{2.12}$$

$$\textbf{Ampère's Law:} \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}, \tag{2.13}$$

govern all of classical electrodynamics. To obtain Poisson's equation, we are particularly interested in Gauss' Law. Gauss' Law states that the divergence of the electric field is proportional to the charge density producing the electric field. By making the substitution $\mathbf{E} = -\nabla V$, Gauss' Law becomes the Poisson equation. A similar analysis will lead to an analogous result using the gravitational field and gravitational potential energy. As a quick aside, the negative sign in the expression arises so as to describe that objects will naturally flow away from a position of high potential energy to a position of low potential energy.

[8]It is common to represent vectors using bold-face text. We will continue to use this notation hereafter.

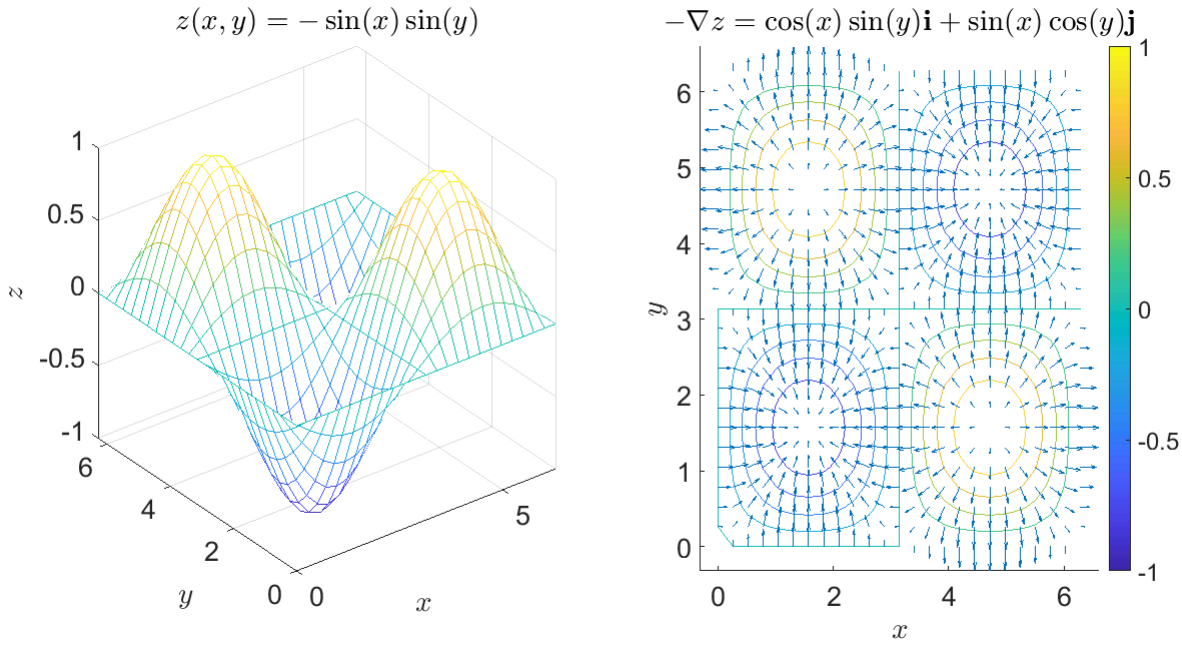


Figure 5: This is a graph of a potential energy function and its field.

Since the gradient points in the direction of greatest increase, adding a negative sign encodes that a particle flows in the direction of greatest decrease of potential energy. We can visualize this phenomenon in Figure 5 above. From a mechanics point of view, we can imagine the figure as containing hills and valleys. A ball will naturally roll down a hill into a valley. Using electrodynamics, from the reference frame of a positive test charge, the peaks represent positive charges and the troughs negative charges. Thus, the test charge will repel away from the positive charges and will attract towards the negative charges. At long last, we return to renormalization and the perfect Laplace operator now that we have established a firm foundation of the standard Laplacian, its finite difference equivalent, and the importance of the Poisson equation.

2.5 Renormalization and the RG Transformation

As a quick refresher, the process of renormalization is used to define a QFT where we can absorb infinities into a redefinition of one or more physical parameters. This is a prerequisite to the *renormalization group transformation (RG)*, which is required to derive

the perfect Laplace operator. Even still, we require one more tool to begin understanding the renormalization process and the RG transformation: *regularization*.

To understand what regularization is, we briefly turn to perturbation theory and present the process of renormalization. Perturbation theory defines a systematic approach to approximate solutions to problems as power series based on the solution to a simpler, related problem. This power series is called a perturbation expansion. Suppose we have a theory where desire to express a function $F(x)$, representing a physical quantity, as a perturbation expansion in terms of the free parameter g_0 [2]. Then the function has the form

$$F(x) = g_0 + g_0^2 F_1(x) + g_0^3 F_2(x) + \cdots , \tag{2.14}$$

where the terms $F_i(x)$ represent divergent quantities, with $F_1(x) = \alpha \int_0^\infty dt/(t+x)$, as an example. The integral sums over virtual states and $\alpha(t+x)^{-1}$ represents the probability amplitude for each of those states [2]. This makes expansion of F ill-defined due to the infinite upper bound of the F_i's. Since we have only one free parameter g_0, called the *coupling constant*, in the theory, we need only to measure the physical quantity $F(x)$ at one point $x = \mu$ in order to fully describe the theory. The purpose of this step is to fix the value of g_0 to reproduce the experimental value $F(\mu) = g_R$, where g_R is called the *renormalized coupling constant* [2]. Now, since F is ill-defined, we seek to express F in terms of g_R. However, to proceed with the renormalization, we need the regularization procedure as our first step. This consists of defining F in terms of a limit Λ, called the *regulator*, such that

1. the $F_i(x)$ are well-defined before the limit is taken, and

2. the original expansion is recovered when taking the limit after renormalization [2].

The regulator acts as a cutoff. We will describe what this cutoff represents when we discuss the RG transformation. The adoption of Λ constitutes the introduction of a new set of functions F_Λ and $F_{i,\Lambda}$ based on Λ so that

$$F_\Lambda(x) = F_\Lambda(x, g_0, \Lambda) = g_0 + g_0^2 F_{1,\Lambda}(x) + g_0^3 F_{2,\Lambda}(x) + \cdots , \tag{2.15}$$

is finite for finite Λ and $F_{1,\Lambda}(x) = \alpha \int_0^\Lambda dt/(t+x)$. Once regularization is complete, we can then rewrite equation (2.15) in terms of g_R and μ. Then, by taking the limit $\lim_{\Lambda\to\infty} F_\Lambda(x, g_R, \mu)$ at fixed g_R and μ, assuming it exists, we theoretically recover $F(x)$ [2]. This concludes the process of renormalization.

Physicists often perform regularization on the space-time continuum, that is, on the continuous domain of three spatial dimensions plus the time dimension. However, for the derivation of the perfect Laplacian, we resort to regularization on the grid, or the lattice. Because regularization is a limiting process, we can naturally extend it to the grid, since with grids we tend to decrease the step-size for better approximations. Lattice regularization is also useful because by definition it is a non-perturbative process and it is convenient for constrained variables [4], which meets the criteria for solving a boundary value problem on the grid.

With renormalization and regularization under our belts, we now discuss the RG transformation. In our example above, we reparameterized F in terms of g_0 and $F(\mu)$, regularized to avoid infinities, and took the limit of the regularized functions to recover F. However, although we have outlined the renormalization process, there are still issues that arise. Renormalization is recursive and higher order divergences are cancelled out by lower order terms expanded about powers of g_R after regularization [2]. But, what if there is a divergence at a lower order term that invalidates the perturbation expansion, due to a large difference between Λ and μ? We ask this question because during renormalization, terms like $g_0 \log(\Lambda/\mu)$ arise, and due to a property called *universality*, it is impossible to know if Λ is very large or truly infinite [2]. Universality describes that in order to understand what happens at lower order terms, we do not need to know about what happens at higher order terms. So, how do we continue? The renormalization group transformation provides our answer. In the words of Delamotte (2004), "instead of using perturbation theory to make a big jump between two very distinct scales, say Λ and μ, we should use it to perform a series" of smaller steps between scales of similar size, like μ_1 and μ_2. This action of reparameterizing from one scale to another of similar size is the essence of the RG transformation. By performing the RG trans-

formation, we can eliminate the scale Λ with our initial g_0 and adopt a scale better suited for the physics in which we are interested [2, 4]. So, to demystify and unify renormalization, regularization, and the RG transformation, we provide a physical interpretation.

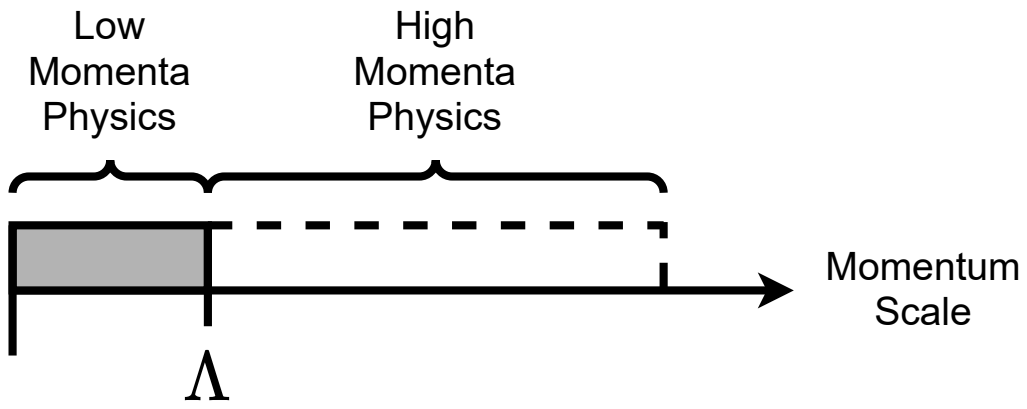


Figure 6: A diagram of the momentum scales and the regulator Λ, which separates the physics we are interested in from the high momenta physics.

Let us revisit Oppenheimer's 1930 problem: the infinite self-energy of the electron-virtual-photon system due to their sharing the original electron's momentum an infinite number of ways. Since there is an infinite range of momenta, we can prescribe a cutoff Λ, the regulator, which separates the low momenta, in which our theory is valid, from the high momenta, where new physics lives [2]. We can see this diagrammatically in Figure 6. At this point, we can begin renormalizing our theory. However, we do not know how the physics behaves for our system at Λ, so it would be better for the results of our theory to be independent of Λ while accounting for the effects of the higher momenta. So, we perform the RG transformation to remove the degrees of freedom at high momenta while accounting for their effect on the low momentum variables [2, 4]. Having provided a physical interpretation of the renormalization process, we now move on to the perfect Laplacian.

2.6 The Perfect Laplacian

Because the derivation of the perfect Laplacian requires extensive knowledge of renormalization, regularization, and the RG transformation on the lattice (and then some), we

attempt to present a clear, yet informal derivation of the perfect Laplacian without sacrificing important details. To begin, our domain is the square $\Omega = (0, L)^2$, and we start with what is called the continuum action for a free scalar field in two dimensions:

$$S = \int_\Omega \frac{1}{2} \sum_{\mu=1}^{2} \partial_\mu \phi(\mathbf{x}) \partial_\mu \phi(\mathbf{x}) \; d\Omega. \tag{2.16}$$

Here, $\phi(\mathbf{x})$ defines a scalar-valued field as a function of $\mathbf{x} \in \mathbb{R}^2$ and ∂_μ represents the partial derivative with respect to x when $\mu = 1$ and with respect to y when $\mu = 2$. Further, $d\Omega = dxdy$. To understand how we arrive at equation (2.16), we look to the Lagrangian formulation of classical mechanics in terms of fields. In classical mechanics, the integrand of the action would consist of the Lagrangian as a function of generalized coordinates for position and velocity, as functions of time, defined in the usual way: $\mathcal{L}(t, q(t), \dot{q}(t)) = T - V$. The quantities T and V denote the kinetic and potential energy, respectively. However, for our purposes, our Lagrangian is of the form $\mathcal{L}(\mathbf{x}, \phi(\mathbf{x}), \partial_\mu \phi(\mathbf{x}))$. The kinetic and potential energy terms are

$$\begin{aligned} T =& \frac{1}{2} \int_\Omega \dot{\phi} \; d\Omega, \\ V =& \frac{1}{2} \int_\Omega (\nabla \phi)^2 \; d\Omega + \frac{1}{2} \int_\Omega m^2 \phi^2 \; d\Omega, \end{aligned}$$

respectively. Because we are working with the free scalar field, we do not have a kinetic energy term, and the mass has a value $m = 0$, removing the potential energy term. We only keep the first term in V, called the gradient energy [14]. Our ultimate goal is to minimize the action S so that we can derive the equations of motion to uncover the history and predict the future of a particle[9], where $\mathcal{L}$ satisfies the Euler-Lagrange equation:

$$\frac{\partial \mathcal{L}}{\partial \phi} - \partial_\mu \left(\frac{\partial \mathcal{L}}{\partial (\partial_\mu \phi)} \right) = 0. \tag{2.17}$$

[9]In this case, we do not mean particle in the classical sense, i.e. a point particle. Rather, because we are dealing with quantum fields, by particle we mean wave.

The simplest Lagrangian one can construct is

$$\mathcal{L} = \frac{1}{2}\partial_\mu\phi\partial_\mu\phi - \frac{1}{2}m^2\phi^2. \tag{2.18}$$

Substituting this Lagrangian into equation (2.17) gives us the Klein-Gordon equation for a free particle:

$$\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\phi - \Delta\phi + \frac{m^2c^2}{\hbar^2}\phi = 0.$$

If we set $m = 0$ in equation (2.18), we obtain the Lagrangian of equation (2.16), and the Klein-Gordon equation reduces to the wave equation for a classical system.

So, where does the Laplacian come from? We utilize integration by parts. For multidimensional functions over a domain Ω, the formula for integration by parts is

$$\int_\Omega \nabla u \cdot \nabla u \; d\Omega = \int_\Gamma u(\nabla u \cdot \mathbf{n}) \; d\Gamma - \int_\Omega u\nabla^2 u \; d\Omega.$$

Here, $\Gamma = \partial\Omega$ is the boundary of Ω. We can recast equation (2.16) as $\int_\Omega \frac{1}{2}\nabla\phi \cdot \nabla\phi \; d\Omega$ and apply integration by parts to obtain

$$\frac{1}{2}\int_\Omega \nabla\phi \cdot \nabla\phi \; d\Omega = \frac{1}{2}\int_\Gamma \phi(\nabla\phi \cdot \mathbf{n}) \; d\Gamma - \frac{1}{2}\int_\Omega \phi\nabla^2\phi \; d\Omega. \tag{2.19}$$

The first integral on the right-hand side is equal to zero. This results from the normal to the surface bounded by Γ being $\mathbf{n} = \mathbf{k}$, which is orthogonal to $\mathbf{i}$ and $\mathbf{j}$, hence a zero inner product. Thus, we are left with

$$S = -\frac{1}{2}\int_\Omega \phi\nabla^2\phi \; d\Omega, \tag{2.20}$$

for the action. By replacing the continuous Laplacian with the discrete Laplacian, derived in Subsection 2.3, and substituting the resulting Lagrangian into the Euler-Lagrange equation, we obtain equation (2.8) with a right-hand side $f_{k,l} = 0$ for every k, l, known as the discrete Laplace equation. We already know from Subsection 2.2 that the error of the discrete Laplacian is $\mathcal{O}(\Delta x^2)$. However, we can construct the perfect Laplacian $\rho(r)$, which eradicates

discretization errors. We can accomplish this utilizing the discrete form of equation (2.16), given by

$$S = \frac{1}{2}\sum_{n,r} \rho(r)\phi_n\phi_{n+r}, \tag{2.21}$$

where $\rho(r) = \rho(r_1, r_2)$ is the perfect Laplacian, $n = (n_1, n_2) = (x_k, y_l)$ are the points on our grid, and $r = (r_1, r_2)$ represents the distance from the central point, where $r \in \mathbb{N}_0^2$ [4]. As a brief example, for the standard Laplacian $\rho_S(r)$, we have

$$\rho_S(0,0) = 4 \quad \text{and} \quad \rho_S(-1,0) = \rho_S(1,0) = \rho_S(0,-1) = \rho_S(0,1) = -1.$$

To obtain the perfect Laplacian we need to perform an RG transformation associated with a fixed point action. This means that under successive RG transformations, the action, and hence, the values of $\rho(r)$ do not change. However, because this operation is beyond the scope of this paper, see [4, 5, 6] for a comprehensive derivation. After performing the RG transformation and solving the associated fixed point equation, the perfect Laplacian in momentum space is

$$\frac{1}{\tilde{\rho}(q)} = \sum_{l=-\infty}^{+\infty} \frac{1}{(q+2\pi l)^2} \prod_{i=0}^{1} \frac{\sin^2(\frac{q_i}{2})}{(\frac{q_i}{2}+\pi l_i)^2} + \frac{1}{3\kappa}, \tag{2.22}$$

where $l = (l_0, l_1)$, $(q+2\pi l)^2 = (q_0+2\pi l_0)^2 + (q_1+2\pi l_1)$, and κ is called the *blocking number*. This number relays how we group our grid points based on the RG transformation. Further, the sum represents a double sum over both l_0 and l_1. By Fourier transforming equation (2.22), we obtain the perfect Laplacian

$$\rho(r_1, r_2) = \int_{-\pi}^{\pi} \frac{d^2q}{(2\pi)^2} e^{iqr}\tilde{\rho}(q) = \frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} e^{i(r_1q_1+r_2q_2)}\tilde{\rho}(q)\, dq_1 dq_2. \tag{2.23}$$

In addition to the equation above, we include two additional equations: used to determine the perfect Laplacian near a wall and a corner, respectively,

$$\textbf{Near A Wall:} \quad \rho(r, n) = \rho(r) - \rho(r_1 + 2n_1 + 1, r_2), \tag{2.24}$$

$$\textbf{Near A Corner:} \quad \rho(r,n) = \rho(r) - \rho(r_1 + 2n_1 + 1, r_2) - \rho(r_1, r_2 + 2n_2 + 1) + \rho(r_1 + 2n_1 + 1, r_2 + 2n_2 + 1), \tag{2.25}$$

where n_1 is the number of units between the center value, $\rho(0,0)$, and the positive $x-$axis and n_2 is the number of units between the center value and the positive $y-$axis.

There are some important properties to note about the perfect Laplacian. The values returned by the perfect Laplacian are called couplings because they relay the relationship between a point and its neighbors, hence the grid points are coupled with one another. The properties of $\rho(r)$ depend on the blocking number, and $\kappa \approx 2$ is the optimal value [4, 5, 6]. With $\kappa = 2$, the couplings $\rho(r)$ decrease exponentially like $\sim \exp(-3.44|r|)$. With this κ, $\rho(r)$ is dominated by the nine surrounding couplings, as in Figure 3. Another important consideration is the truncation. Because there are infinitely many couplings, the perfect Laplacian must be truncated for practical use. It is important that as we refine our grid, our truncated perfect Laplacian approach the continuum result. We can achieve this because we expect the discretization to preserve the physical properties of the continuous Laplacian [6] due to the RG transformation. For the continuous system, the energy of an excitation with momentum k_1 is $E(k_1) = |k_1|$ for $k_1 \in (-\infty, \infty)$. We find this is indeed also true for the perfect Laplacian, based on the infinite summation over l [5, 6]. Thus we must satisfy the condition

$$\sum_{r_1,r_2} \rho(r_1, r_2) = 0. \tag{2.26}$$

Further, the perfect Laplacian must approach the the continuous Laplacian for small values, so $\lim_{q\to 0} \rho(q) = q^2$ [5, 6]. For this to be true, we must satisfy

$$\sum_{r_1,r_2} \rho(r_1, r_2) \left(r_1^2 + r_2^2\right) = -4. \tag{2.27}$$

By satisfying the conditions in equations (2.26) and (2.27), we *normalize* the perfect Laplacian. After normalization, we may successfully implement the practical use of the perfect Laplacian. Now, we discuss the methods.

3 Methods

Before outlining the methods used, let us briefly revisit the motivation for this paper after the lengthy opening. Differential equations are important because they serve as models with which we can develop an understanding of the world around us, from the atomic to the astronomical. However, because many differential equations lack analytical solutions, we utilize numerical approaches. Not all numerical methods are equal—some are better and others are worse—and so, we choose methods best suited for the problem at hand. In this paper, we compare and contrast discrete standard and perfect Laplace operators to assess their efficacy in solving the Poisson equation, exploring the storage and memory, discretization error, and the order of the operators. Now, we discuss our methods.

3.1 The Poisson Systems

	System	**Solution**
1	$\Delta u = 2\sin(x)\sin(y), \quad (x,y) \in [0,2\pi]^2$	$u(x,y) = -\sin(x)\sin(y)$
2	$\Delta u = 16\left(r^2 - \frac{1}{2}\right)e^{-2r^2}$ $r^2 = (x-\pi)^2 + (y-\pi)^2, \quad (x,y) \in [0,2\pi]^2$	$u(x,y) = e^{-2r^2}$
3	$\Delta u = 2y(2\pi - y)(\sin(2x) + x\cos(2x)) - 2x\sin^2(x)$ $(x,y) \in [0,2\pi]^2$	$u(x,y) = xy(2\pi - y)\sin^2(x)$
4	$\Delta u = \begin{cases} 2\sin(x)\sin(y), & (x,y) \in R, \\ 0, & (x,y) \notin R, \end{cases} \quad (x,y) \in [0,2\pi]^2$	$u(x,y) = \begin{cases} -\sin(x)\sin(y), & (x,y) \in R, \\ 0, & (x,y) \notin R, \end{cases}$

Table 1: These are the systems used for analysis. They all obey homogeneous Dirichlet boundary conditions.

For the analysis, we compare the standard 5 and modified 9-point discrete Laplacians with four versions of the perfect Laplacian, all of which constitute their own Poisson solver. Four different Poisson systems were derived for this purpose and they are listed in Table 1 above. These systems were derived to provide variation in testing the efficacy of all Laplacians

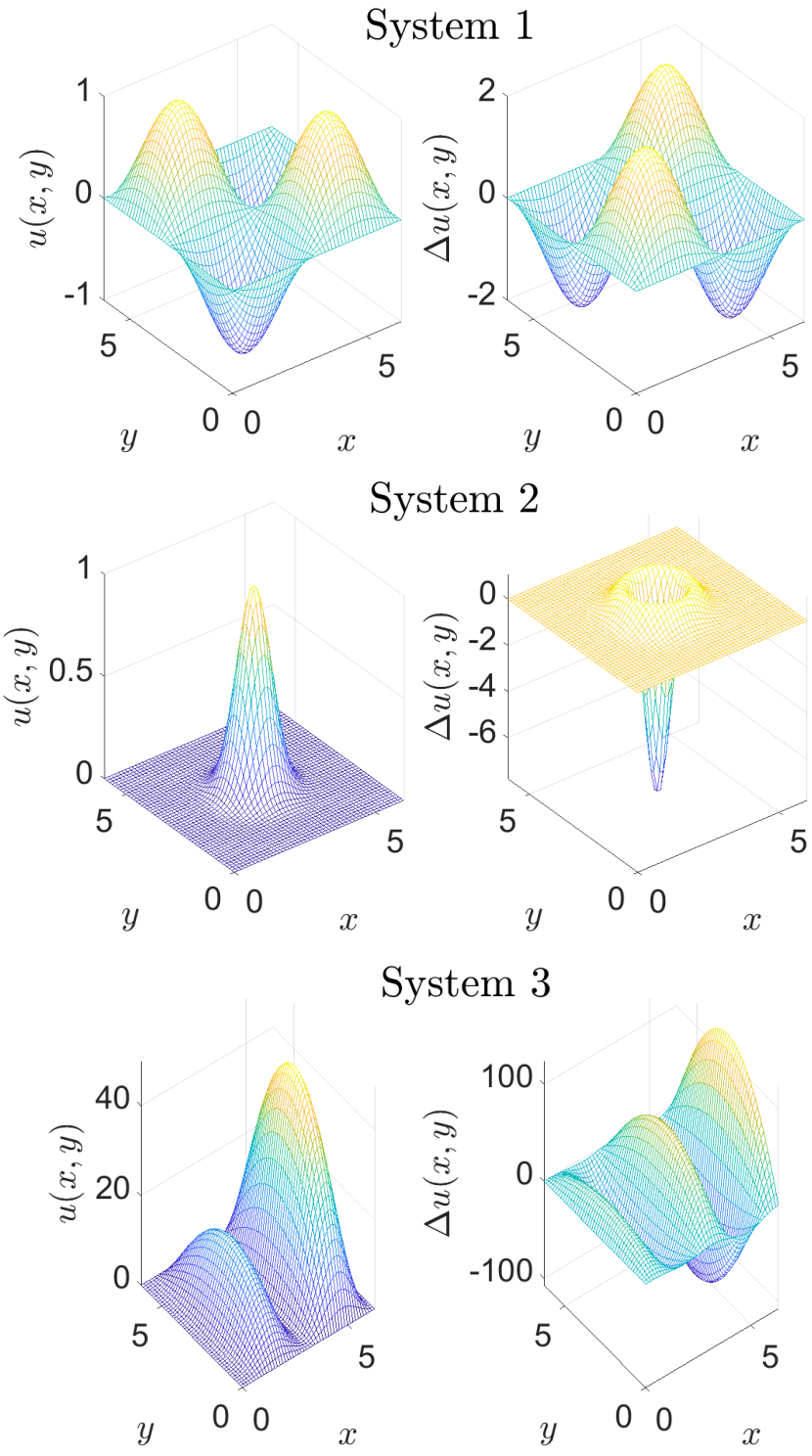


Figure 7: Visualization of all of the systems listed in Table 1. The solutions are on the left and the Laplacian on the right.

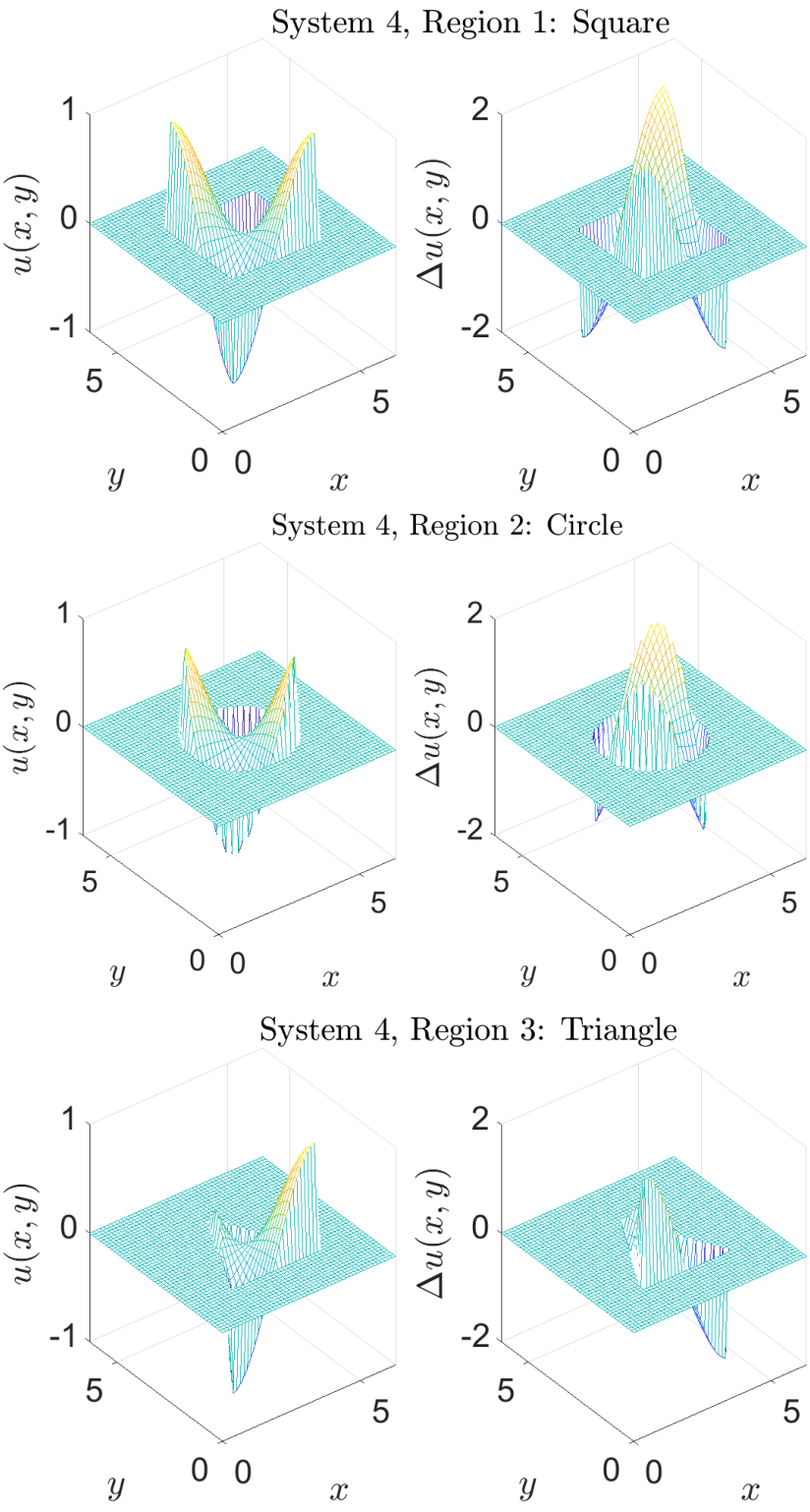


Figure 8: System 4 with the different internal regions. The solutions are on the left and the Laplacians are on the right.

in different environments: System 1 is 2π-periodic, System 2 is spherically symmetric, System 3 is neither 2π-periodic nor spherically symmetric, and System 4 is piecewise. The plots of the first three systems together are in Figure 7 on page 24, and the plots for System 4 with the three different regions—rectangular, circular, and triangular—are in Figure 8 on page 25. However, the most important characteristic is that all four systems obey homogeneous Dirichlet boundary conditions due to how the perfect Laplacian is defined. Now we discuss how to code the Poisson solvers. The complete files are on the Git Repository here.

3.2 Coding the Poisson Solvers

The Poisson solvers were written in MATLAB version R2020b. Most functions presented here accept variable input arguments, so inputs after the semi-colon denote these inputs. For consistency, we present the Poisson solvers in the order in which they appear in the algorithm. But before discussing the solvers, we discuss the implementation of the perfect Laplacian. The perfect Laplacian (see equation (2.23)) was written as the `couplings` function, which returns the couplings and takes variable inputs. This was accomplished by first writing equation (2.22) as a double `for`-loop in a local function within the main function. This local function returns $1/\tilde{\rho}(q)$ and takes r_1, r_2, and N as arguments, where $-N$ and $+N$ are the lower and upper limits for the double `for`-loop (and hence the double summation), respectively. Then, the parent function contains equations (2.23)-(2.25), with r_1, r_2, and N as fixed inputs and n_1 and n_2 as the variable inputs.

Couplings Function: Return Couplings of the Perfect Laplacian

```
function Return Couplings, Inputs: r1, r2, N; n1, n2
Equation(2.23) % Inputs: r1, r2, N
Equation(2.24) % Inputs: r1, r2, N; n1
Equation(2.25) % Inputs: r1, r2, N; n1, n2
function Return 1/rho, Inputs: q1, q2, N % local/internal function
Double Sum as Double for Loop % Inputs: q1, q2, N
end
end
```

The first solver is the `myPoisson1` function and has options for both standard 5 and modified 9-point stencils (see Figures 3 and 4), in addition to one of three unique perfect Laplace operators, derived independently, which utilizes the modified 9-point stencil as a scaffold. This is called the Perfect Modified (PM) 9-point operator, and the standard 5 and 9-point operators are referred to as the STD 5-point operator and the M 9-point operator, respectively. These Laplacians use the Jacobi method, as mentioned in Section 2, to solve the Poisson equation. Explicitly, the Jacobi method is provided below in both the traditional and vectorized forms, found in Leveque (2007). In the code below, $u_{i,j}$ represents the (i,j)th approximation to the solution of the Poisson equation, h is the step-size, $f_{i,j}$ is the (i,j)th

Jacobi Method

```
% iterating over the points
for 1:maxiter
    for i = 2:N-1
        for j = 2:N-1
            u(i,j) = 0.25*(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)-h^2*f(i,j));
        end
    end
end

% vectorized form
I = 2:N-1;
J = 2:N-1;
for 1:maxiter
    u(I,J) = 0.25*(u(I+1,J)+u(I-1,J)+u(I,J+1)+u(I,J-1)-h^2*f(I,J));
end
```

value of the laplacian of $u_{i,j}$, and the capital and lowercase i's and j's represent the interior points. The first and last rows and columns of u contain the homogeneous Dirichlet boundary conditions. The pseudo-code for `myPoisson1` is below. To derive the couplings of the PM 9-point operator, the `couplings` function must be used. First, an empty 3×3 array was

myPoisson1: Solve Laplace/Poisson Equation with any Boundary Conditions

```
function Return Solution to Laplace or Poisson Equation, Inputs: RHS, ...
    Step-Size; Boundary Conditions (BC), Iterations
5-Point Stencil Operator % Inputs: RHS, Step-Size; BC, Iterations
M 9-Point Stencil Operator % Inputs: RHS, Step-Size; BC, Iterations
PM 9-Point Stencil Operator % Inputs: RHS, Step-Size; BC, Iterations
end
```

constructed, after which a double `for`-loop was used to determine the non-normalized couplings. Because the eight nearest values are required, in addition to the central value, both loop indices range from -1 to 1 and iterate over r_1 and r_2 with $N = 25$ for the double summation. To normalize the couplings, equations (2.26) and (2.27) were used. Equation (2.26) must be used first to determine the central value. By setting the original central value to zero and then adding together all elements of the array, we obtain a prerequisite to new the central value. Then, to satisfy equation (2.26), the new central value will then be the negative of this prerequisite number. Once completed, the array is updated with the new central value. Next, equation (2.27) is used. In applying this second condition, we obtain a new value, and we scale the updated array by -4 divided by the new value for complete normalization. The `myPoisson1` function returns the solution to either the Laplace or Poisson equations on a square with any boundary conditions. The fixed inputs are the right-hand side of the system and the step size, whereas the variable inputs are the boundary conditions and the number of iterations for the Jacobi method.

The second Poisson solver is the `myPoisson3` function, which utilizes the second of the three unique perfect Laplace operators, called the Single Normalized 25-point perfect Laplace

myPoisson3: Solve Poisson Equation with the SN Operator

```
function Return Solution to Poisson Equation, Inputs: RHS, Step-Size, ...
    Iterations
SN 25-Point Operator % Inputs: RHS, Step-Size, Iterations
end
```

operator (SN 25-point operator). This utilizes ρ_1 on page 8 of [6]. This operator was derived using the `couplings` function described above in the same manner as the PM 9-point operator. However, the loop indices range from -2 to 2. The `myPoisson3` function returns the solution to only the Poisson equation with trivial boundary conditions and accepts the right-hand side of the system, the step-size, and the number of iterations for the Jacobi method as inputs.

The last of the three unique perfect Laplace operators is the Parameterized Non-normalized 25-point Laplace operator (PNN 25-point operator) in the `myPoisson2` function. We discuss this operator in conjunction with the operator used in the `perfPoisson` function, denoted as the Parameterized Normalized 25-point Laplace operator (PN 25-point operator), discussed in Hauswirth (2000). Each operator consists of six separate Laplace operators that differ

myPoisson2: Solve Poisson Equation with the PNN Operator

```
1 function Return Solution to Poisson Equation, Inputs: RHS, Step-Size, ...
      Iterations
2 PNN 25-Point Operator % Inputs: RHS, Step-Size, Iterations
3 end
```

perfPoisson: Solve Poisson Equation with the PN Operator

```
1 function Return Solution to Poisson Equation, Inputs: RHS, Step-Size, ...
      Iterations
2 PN 25-Point Operator % Inputs: RHS, Step-Size, Iterations
3 end
```

depending on the grid location, i.e., near a wall, near a corner, or far from both. First, the `couplings` function was used in the same manner as the SN 25-point Laplace operator. However, instead of deriving just ρ_1, where only r_1, r_2, and N were specified, the values n_1 and n_2 were also specified for the PNN operator. We list the values for n_1 and n_2 below in Table 2. The first difference between the PNN and PN operators is the obvious distinction of normalization: the PNN operator is not normalized and the PN operator is normalized.

n_1	n_2	PNN Operator
1	-	$\hat{\rho}_2$
0	-	$\hat{\rho}_3$
1	1	$\hat{\rho}_4$
0	1	$\hat{\rho}_5$
0	0	$\hat{\rho}_6$

Table 2: The values of n_1 and n_2 to be specified for the PNN operator.

The second difference is the normalization process of the PN operator does not consist of deriving each operator individually, as with the PNN operator, and then normalizing each one. In Hauswirth (2000), only ρ_1 was derived and normalized, and the last five operators were built using ρ_1 and equations (2.26) and (2.27). Once the operators were formed, the non-normalized operators were housed in the `myPoisson2` function and the normalized operators were housed in the `perfPoisson` function. Both return the solution to the Poisson equation with trivial boundary conditions and accept the right-hand side of the system, the step-size, and the number of iterations for the Jacobi method as inputs.

There is an important note regarding the `myPoisson2` function with respect to deriving the operators. Due to the symmetric nature of equations (2.26) and (2.27), the operators need some adjustment before use. As an example, we examine the equivalent of the ρ_4 PN operator for the PNN operator, $\hat{\rho}_4$. This operator is obtained by using a double `for`-loop iterating over the `couplings` function with the inputs of $(r_1, r_2) = (i, j)$ and $(n_1, n_2) = (1, 1)$ with $N = 25$. The result is the first set of values below in Figure 9. We can see that the first row and the first column are negative reflections of the second row and second column, respectively. So, how do we appropriately adjust $\hat{\rho}_4$? We know there is one space between the central value and the positive $x-$axis ($n_1 = 1$) and there is one space between the central value and the positive $y-$axis ($n_2 = 1$). However, due to MATLAB's output, the positive $x-$axis corresponds to the first row and the positive $y-$axis corresponds to the first column. Thus, the central value is one unit from the top row instead of the bottom row, which is unexpected due to the notation of the first quadrant of the coordinate plane. Thus, in the resulting $\hat{\rho}_4$ operator for the PNN operator, the first row and column contain zeros because

they represent the boundary of the grid. The last modification to apply so that $\hat{\rho}_4$ looks like ρ_4 in Hauswirth (2007) is to flip the matrix about the vertical axis (from left to right). The result is the second set of values below in Figure 9.

```
hat_rho4 =

   -0.1873    0.1873    0.6160    0.1897    0.0023
    0.1873   -0.1873   -0.6160   -0.1897   -0.0023
    0.6160   -0.6160    3.2405   -0.6180   -0.0021
    0.1897   -0.1897   -0.6180   -0.1903   -0.0007
    0.0023   -0.0023   -0.0021   -0.0007    0.0016


hat_rho4 =

         0         0         0         0         0
   -0.0023   -0.1897   -0.6160   -0.1873         0
   -0.0021   -0.6180    3.2405   -0.6160         0
   -0.0007   -0.1903   -0.6180   -0.1897         0
    0.0016   -0.0007   -0.0021   -0.0023         0
```

Figure 9: The unadjusted $\hat{\rho}_4$ operator is on the top and the adjusted operator is on the bottom.

3.3 The Regions of System 4

3.3.1 The Tetragonal Region

Before delving into the algorithms, we derive the expressions necessary to define the regions used in the computation of System 4. The first region is a convex tetragonal region, provided in Figure 10 below. Since System 4 is zero outside the region and equal to System

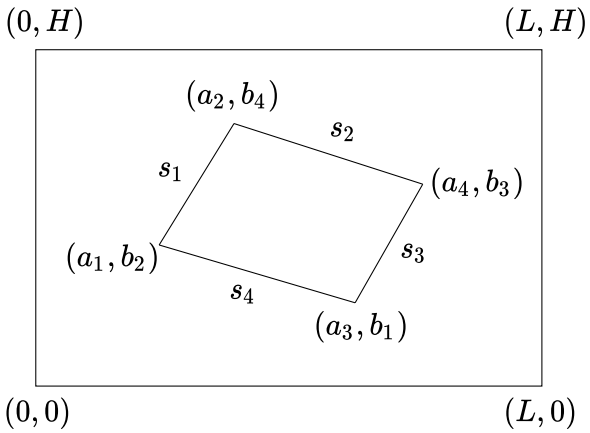


Figure 10: Diagram of a tetragonal region.

1 inside the region, we are interested in the interior x and y values, that is,

$$R_1 = \{(x, y) \in \mathbb{R}^2 \colon s_3, s_4 \leq (x, y) \leq s_1, s_2\}.$$

We express the region as a collection of inequalities, expressing the sides s_i as equations of lines using the Point-Slope formula. We have

$$\begin{aligned}
s_1 &: \quad y \leq m_1(x - a_1) + b_2, \quad m_1 = \frac{b_4 - b_2}{a_2 - a_1}, \\
s_2 &: \quad y \leq m_2(x - a_2) + b_4, \quad m_2 = \frac{b_3 - b_4}{a_4 - a_2}, \\
s_3 &: \quad y \geq m_3(x - a_4) + b_3, \quad m_3 = \frac{b_3 - b_1}{a_4 - a_3}, \\
s_4 &: \quad y \geq m_4(x - a_3) + b_1, \quad m_4 = \frac{b_1 - b_2}{a_3 - a_1}.
\end{aligned}$$

tetragon: Return Desired Tetragonal Region

```
function Return Tetragonal Region, Inputs: Grid; Default ...
    Square/Diamond, Vertices
Tetragon Inequalities % Inputs: Grid; Default Square/Diamond, Vertices
end
```

In MATLAB, the above system of inequalities was written explicitly in the function file `tetragon`, which takes the grid as a fixed input and the vertices of the desired tetragon as the variable input. It is important to note that $a_1 \leq a_2, a_3 \leq a_4$ and $b_1 \leq b_2, b_3 \leq b_4$ must be satisfied to preserve the shape of the region. Further, in the absence of vertices, the default region is either a square or a diamond, depending on the specified option.

3.3.2 The Elliptical Region

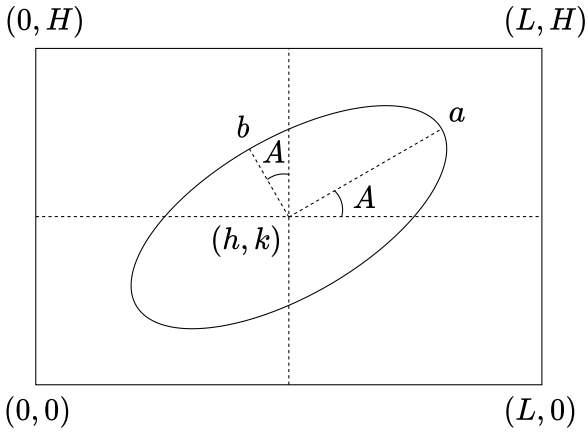


Figure 11: Diagram of an elliptical region.

The next region is the elliptical region, R_2, shown in Figure 11 above. We know that the general expression for an ellipse is

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1. \tag{3.1}$$

To generalize this expression, we add the possibility of rotation by an angle A measured from the positive $x-$axis. It is not immediately clear how to accomplish this, but to begin,

we can make the following equality:

$$x - h = a\cos\theta$$
$$y - k = b\sin\theta.$$

If the ellipse is centered at the origin and $\theta = 0$, then we find $x = a$. Likewise, we find $y = b$ when $\theta = \pi/2$. We can thus express the above as a vector, namely

$$\mathbf{v} = \begin{bmatrix} x-h \\ y-k \end{bmatrix}.$$

Now, operate on $\mathbf{v}$ with the rotation matrix:

$$\begin{bmatrix} \cos(A) & -\sin(A) \\ \sin(A) & \cos(A) \end{bmatrix} \begin{bmatrix} x-h \\ y-k \end{bmatrix} = \begin{bmatrix} (x-h)\cos(A) - (y-k)\sin(A) \\ (x-h)\sin(A) + (y-k)\cos(A) \end{bmatrix} = \tilde{\mathbf{v}}.$$

Because rotation preserves lengths, we can rewrite

$$(x-h)\cos(A) - (y-k)\sin(A) = a\cos\theta \implies \cos^2\theta = \frac{\left((x-h)\cos(A) - (y-k)\sin(A)\right)^2}{a^2}$$
$$(x-h)\sin(A) + (y-k)\cos(A) = b\sin\theta \implies \sin^2\theta = \frac{\left((x-h)\sin(A) + (y-k)\cos(A)\right)^2}{b^2}.$$

By utilizing the Pythagorean identity, we obtain the general formula for an ellipse

$$\frac{\left((x-h)\cos(A) - (y-k)\sin(A)\right)^2}{a^2} + \frac{\left((x-h)\sin(A) + (y-k)\cos(A)\right)^2}{b^2} = 1.$$

Finally, replacing "=" with "$\leq$," we obtain the region R_2. This region was written into the `ellipse` function. This function takes the grid as a fixed input and the center, major and minor axes, and angle from the positive $x-$axis as variable inputs. In like manner as the `tetragon` function, in the absence of variable inputs, the default output is a circle centered

at the grid. Further we restrict the center so that it lies within the grid.

ellipse: Return Desired Elliptical Region

```
function Return Elliptical Region, Inputs: Grid; Default Circle, ...
    Center, Axes, Angle
Ellipse Inequality % Inputs: Grid; Default Circle, Center, Axes, Angle
end
```

3.3.3 The Triangular Region

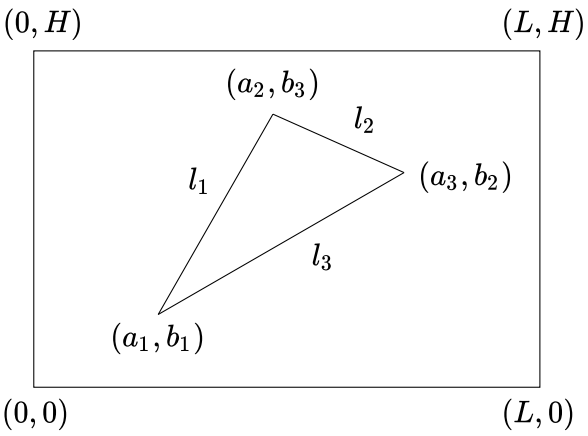


Figure 12: Diagram of a triangular region.

The derivation of the triangular region R_3, shown in Figure 12, is identical to that of the tetragonal region. The region is defined by the inequalities provided below.

$$
\begin{aligned}
l_1: &\quad y \le m_1(x - a_1) + b_1, \quad m_1 = \frac{b_3 - b_1}{a_2 - a_1}, \\
l_2: &\quad y \le m_2(x - a_2) + b_3, \quad m_2 = \frac{b_2 - b_3}{a_3 - a_2}, \\
l_3: &\quad y \ge m_3(x - a_3) + b_2, \quad m_3 = \frac{b_2 - b_1}{a_3 - a_1},
\end{aligned}
$$

The `triangle` function takes the grid as a fixed input and the vertices of the triangle are the variable inputs. In the absence of the vertices, the function returns an equilateral triangle at the center of the grid. In the same manner as the `tetragon` function, the conditions

$a_1 \leq a_2 \leq a_3$ and $b_1, b_2 \leq b_3$ must be satisfied to preserve the triangular shape of the region.

triangle: Return Desired Triangular Region

```
function Return Triangular Region, Inputs: Grid; Default Triangle, Vertices
Triangle Inequalities % Input: Grid; Default Triangle, Vertices
end
```

3.4 The Algorithm

Two algorithms were implemented. Algorithm 1 solves Systems 1-3, and Algorithm 2 solves System 4. They both plot the solutions and calculate the error of each solver. We look to the file size of each Poisson solver for assessing the storage and memory.

The body of Algorithm 1 takes place in a `for`-loop where the loop iterates over the size of the grid. The loop index ranges from $j = 6, \ldots, 30$, which is used for the matrix index and implicitly defines the system size from the step-size $h = 2\pi/(j+1)$. This choice of indices was the best range to observe the relationship between the step-size and error. Before the loop, the arrays that store the matrix index, the errors of each Poisson solver, and the step-size for the grid were preallocated. The arrays for the matrix index and the step-size are both column vectors with 25 entries[10], and the array storing the errors has dimensions 25×6. This is because there are six different Poisson solvers each using a different Laplacian[11]. Then, inside the loop, with all necessary parameters in terms of j, the maximum number of iterations was capped at 1000 and the grid was defined in terms of the step-size. The purpose for choosing 1000 iterations was to allow each solver ample time to converge. Next, Systems 1-3 were coded as functions of the grid followed by the the Poisson solvers as functions of each system. Lastly, the error of each method was calculated by taking the infinity norm of the difference between the continuous solution and the solver solution. They were then stored in the error array for each Poisson solver. We include the psuedo-code below and

[10] These can also be row vectors with 25 columns.

[11] To be specific, the `myPoisson1` function houses three of the six Laplace operators. So, there are technically four Poisson solvers, one of which contains three different operators.

the code details are in the Git Repository here. There is an important note regarding the pseudo-code. Only System 1 is shown below. So, System 1 must be replaced with Systems 2 and 3 for each computation. This can be accomplished in MATLAB via commenting out the systems that are not used. Lastly, Algorithm 2 is identical to Algorithm 1. The distinction lies in replacing System 1 with System 4 confined to a region. In a likewise manner, only one region is used and the others are commented out.

For the analysis of the data, two figures were made. The first figure consisted of the errors plotted against the step-size, with the errors on the vertical axis and the step-sizes in descending order on the horizontal axis. For the second figure, the natural logarithm of both the errors and the step-sizes were computed. Then, for each solver, the log of the step-size and the log of the error was fit using a first order polynomial. Then the log of the error was plotted against the log of the step-size, with the former on the vertical axis and the latter on the horizontal axis. This concludes how the computations and analyses were conducted.

Algorithm 1: Solve Poisson Equation for Systems 1-3 with Varying Grid Size

```
Preallocate Matrix Index, Error, Step—Size
for j = 6:25
    MaxIter = 1000;
    Step—Size = 2*pi/(j+1);
    Store Step—Size;
    Store Matrix Index as j;
    Grid = x,y from 0 to 2*pi with Step—Size;
    System 1 as a Function of Grid;
    Poisson Solver 1 For System 1;...
    Poisson Solver 6 For System 1;
    Calculate Errors;
    Store Errors in Error Array;
end
Store Data Obtained From Loop In External File For Analysis
```

4 Analysis and Discussion

The analysis consists of analyzing and discussing the systems in the order in which they appeared, that is, System 1 to System 4. The overall analysis will begin with the memory and storage because it is the only section common to both. Then, we examine each system individually looking at the the discretization error, the order of the solver, and their plots.

4.1 Storage and Memory

Function File	Storage (Only Code)	Storage (Comments & Documentation)
`myPoisson1`	11 KB	14 KB
`myPoisson2`	12 KB	15 KB
`myPoisson3`	10 KB	13 KB
`perfPoisson`	12 KB	15 KB

Table 3: The file sizes of all the Poisson solvers.

We look only to the file sizes to analyze the storage and memory of each Poisson solver. We summarize the results in Table 3. To analyze the file sizes entirely, each file was duplicated where one copy was stored without comments and documentation and the other copy was stored with comments and documentation, for direct comparisons. For each file, the difference between the file with and without comments and documentation is 3 KB. The smallest file is the `myPoisson3` file and the largest files are the `myPoisson2` and `perfPoisson` functions, with the `myPoisson1` file being in between. This is naturally consistent, since the two largest files utilize six different operators, in addition to an implementation of the Jacobi method for each one. However, an important note is that the `myPoisson1` function contains three separate operators: the STD 5-point operator, the M 9-point operator, and the PM 9-point operator. This was only done for convenience. If separated into different function files, each would constitute about one third the file size presented in the table above. Thus, file size in this case does not really present a significant challenge. However, if the file size is too great, then the performance of the code could be affected.

4.2 System 1

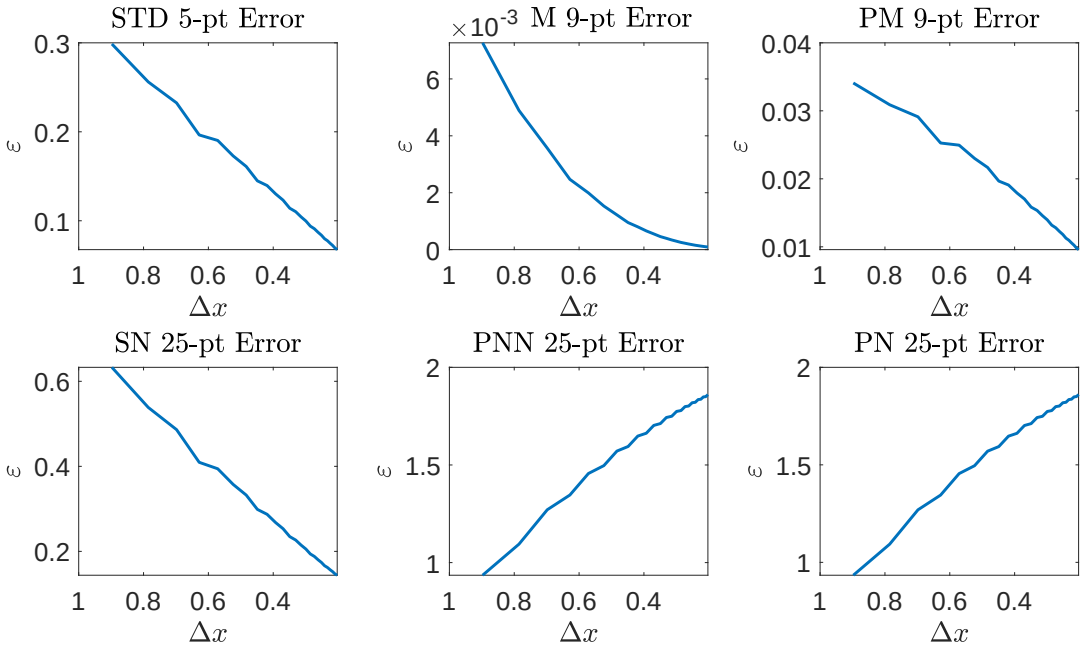


Figure 13: The errors of each solver as the step-size decreases for System 1.

Figure 13 above depicts the plots of the errors ε of each solver against the step-size Δx. In going from left-to-right and top-to-bottom, we find that the error decreases as step-size decreases for the first four panels, but increases for the last two panels. Additionally, the error for the M 9-pt operator decreases much faster than any of the other operators. We can observe this behavior in Figure 14 below, which plots the log of the error against the log of the step-size and displays the order of the operator. As a note, for all plots of the order, the black dots represent the logarithmic data of the errors against the step-size, and the blue line represents the line of best fit. To recall from Section 2, the order of a numerical method is a number p such that $|\mathbf{y}_{n+1} - \mathbf{y}_n| = \mathcal{O}(\Delta x^{p+1})$. Because we have the exact solutions, we may identically define the order using the absolute value of the difference between the approximation $\tilde{u}$ and the exact solution u for a particular grid size, $|\tilde{u}_{\Delta x} - u_{\Delta x}| = \mathcal{O}(\Delta x^{p+1})$. Note that both $\tilde{u}_N$ and u_N are matrices, and again, we utilize the infinity norm. In Figure 14, we can see that the order of the STD, PM, and SN operators all have an order of

approximately $p = 1$, which means that the error for these methods decreases as $\mathcal{O}(\Delta x^2)$. For further concreteness, when the step-size is halved, the error decreases by a factor of 4. In looking at the PNN and PN operators, they have a negative order, meaning that the error

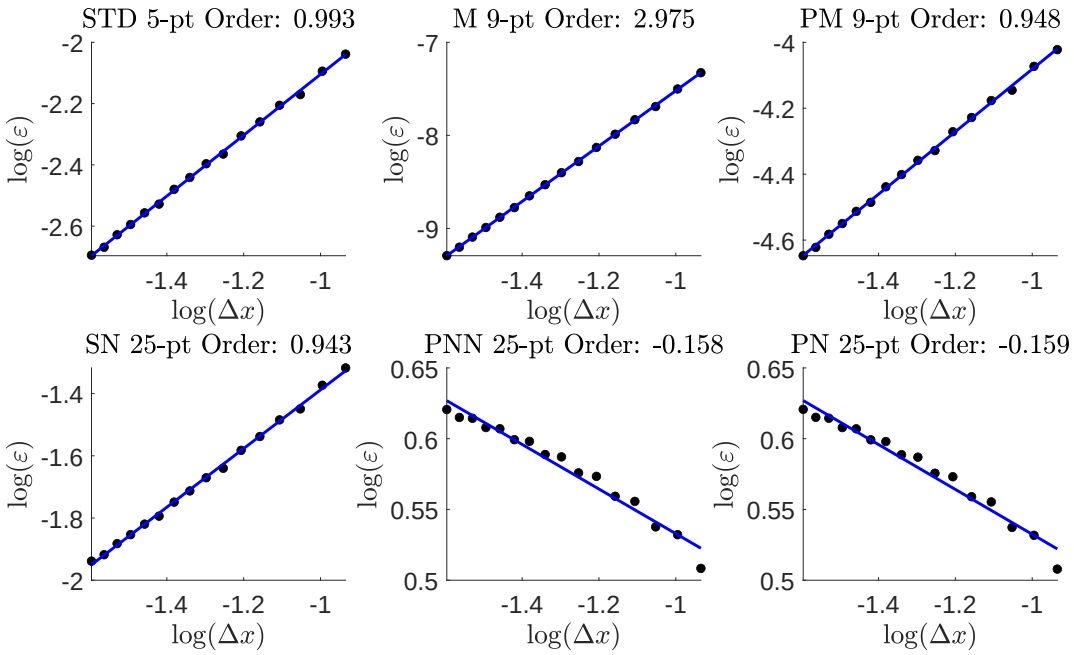
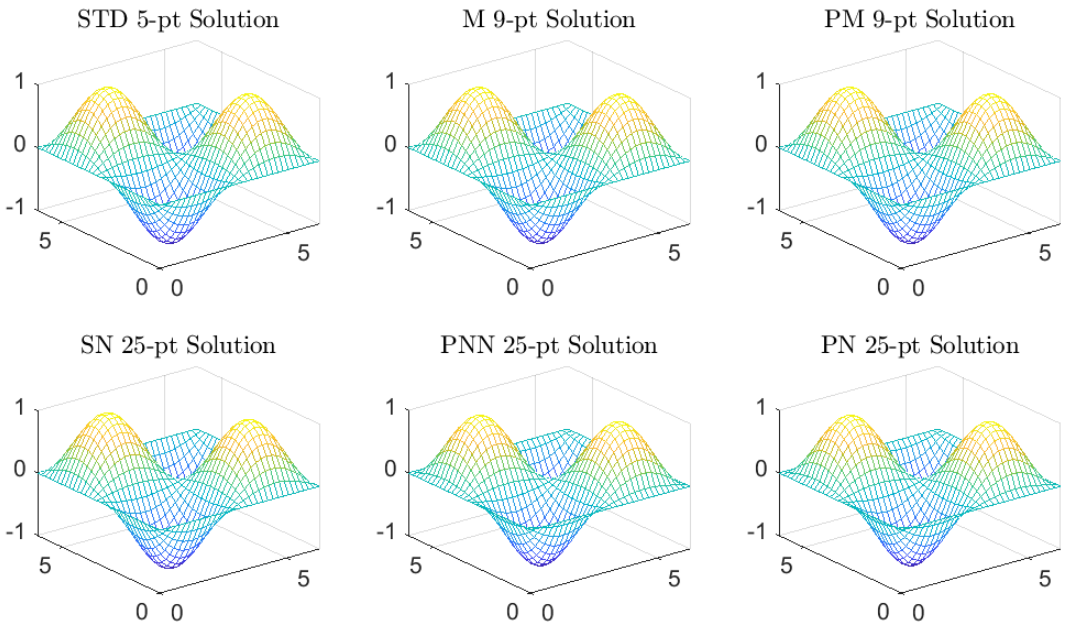
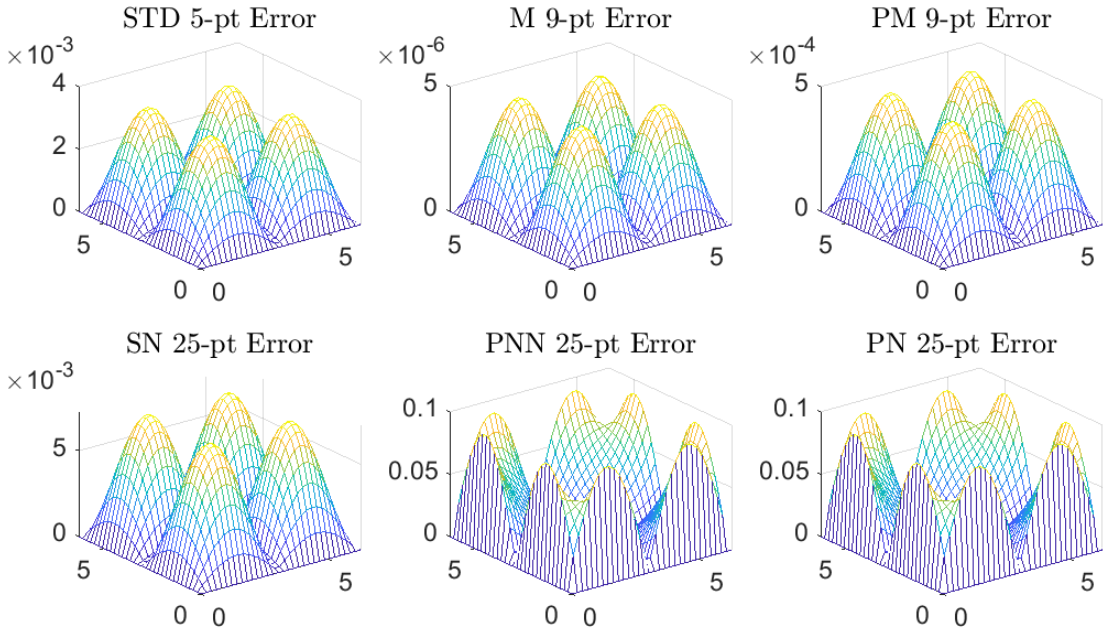


Figure 14: Plots of the orders of each operator for System 1.

decreases as $\mathcal{O}(\Delta x^{0.8})$, which is worse than all other operators. Lastly, the M operator has an order of 3, where the error decreases as $\mathcal{O}(\Delta x^4)$, which is consistent with numerical analyses of the modified 9-point method [7, 10]. Given the nature of the perfect Laplace operators (operators PM, SN, PNN, and PN), for System 1, they are worse than the standard 5 and modified 9-point operators. Further, for this system, the parameterized perfect Laplace operators perform worse than both the PM and SN operators, which both contain only one operator. For completion, we analyze the plots of the solutions of each solver and their errors for System 1, provided in Figure 15. These plots were produced using 1000 iterations for each operator and a matrix index equal to 30. For Figure 15a, we have the solution plots. In comparing these plots with that of the exact solution, found in Figure 7, they are essentially identical. However, when we examine the absolute value of the errors in Figure 15b, we can



(a) Plots of the solutions calculated by the operators.



(b) Plots of the errors of the operators.

Figure 15: These are the solution and error plots for each of the operators for System 1.

see how drastically different the operators are in solving System 1 in scale, which is consistent with Figure 13. Most notably, though, is that the largest error for the first four operators occurs at the peaks and troughs of the solution, whereas the largest errors for the PN and PNN operators occur at the boundaries.

4.3 System 2

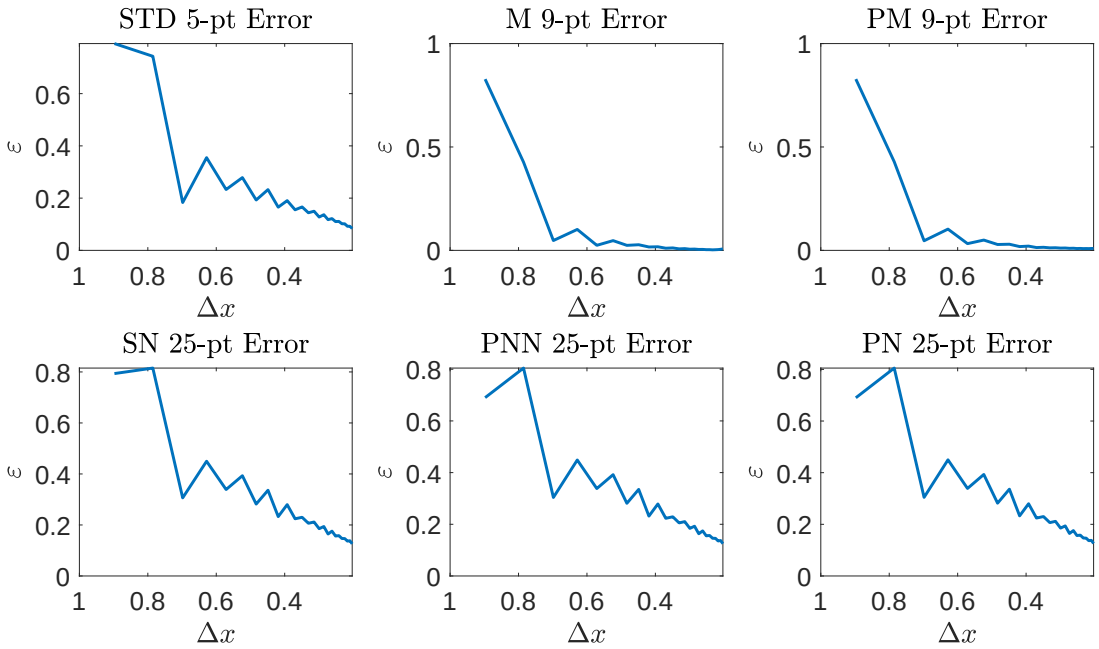


Figure 16: The errors of each solver as the step-size decreases for System 2.

For System 2, we again observe a decreases in error for the first four operators as the step size decreases, but in contrast to System 1, the PNN and PN operators also exhibit a decreasing error as the step-size decreases. This is shown in Figure 16 above. Further, they appear to decrease in a manner similar to the SN operator. What is interesting to note is the oscillatory behavior in the decrease of the errors. In System 1, the decreases and increases in the error was rather smooth. This could be a result in the difference between System 1 being $2\pi-$periodic and System 2 being spherically symmetric. In looking at the orders of each operator in Figure 17, we can indeed verify that the STD, SN, PNN, and PN operators

all have an order of about 1 with an error decay of $\mathcal{O}(\Delta x^2)$, although the STD operator is slightly better than the SN, PNN, and PN operators. Likewise, the SN operator is negligibly better than the PNN and PN operators, which are almost identical, much like in System 1. For the M operator, we do not observe anything drastically different than what we observed for System 1. The order of the M operator is 3 with an error decay of $\mathcal{O}(\Delta x^4)$. However, we do see a difference with the PM operator: the order of the method is about 2.5,

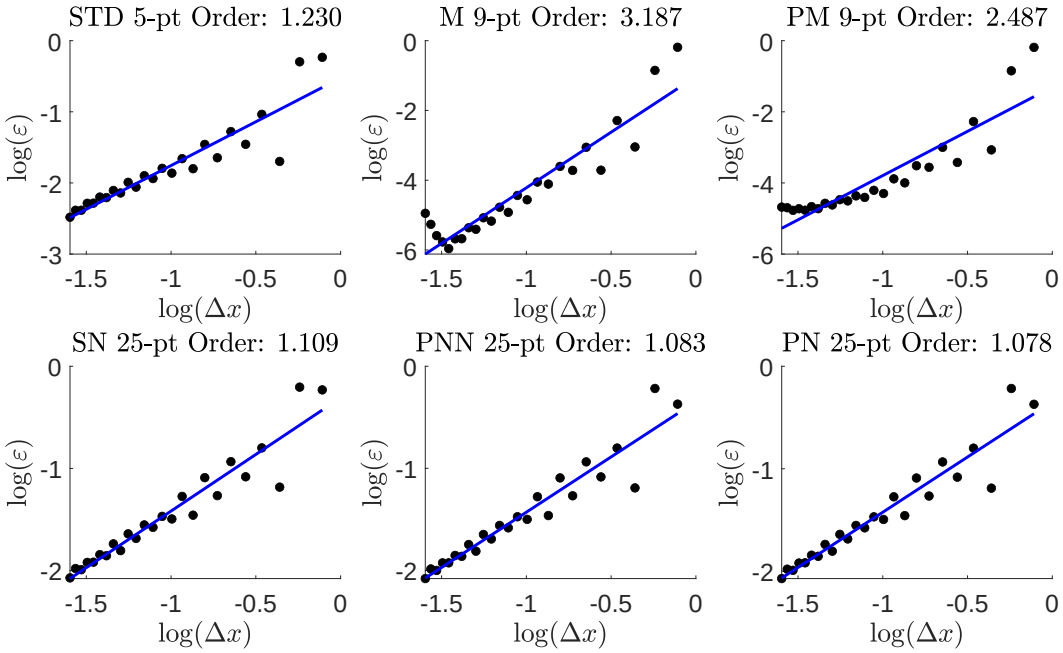
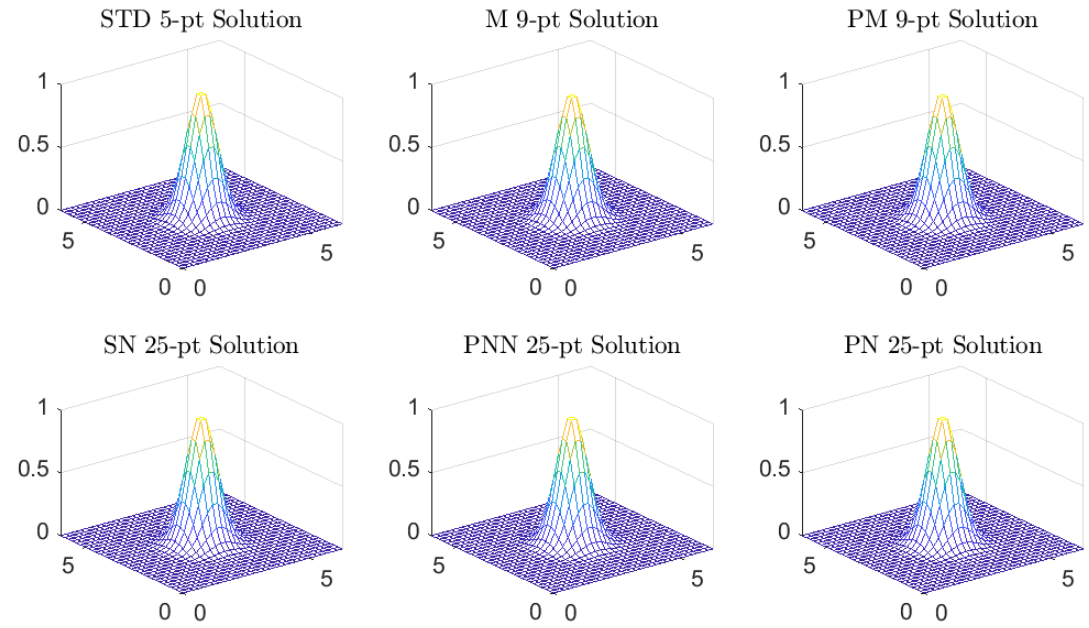
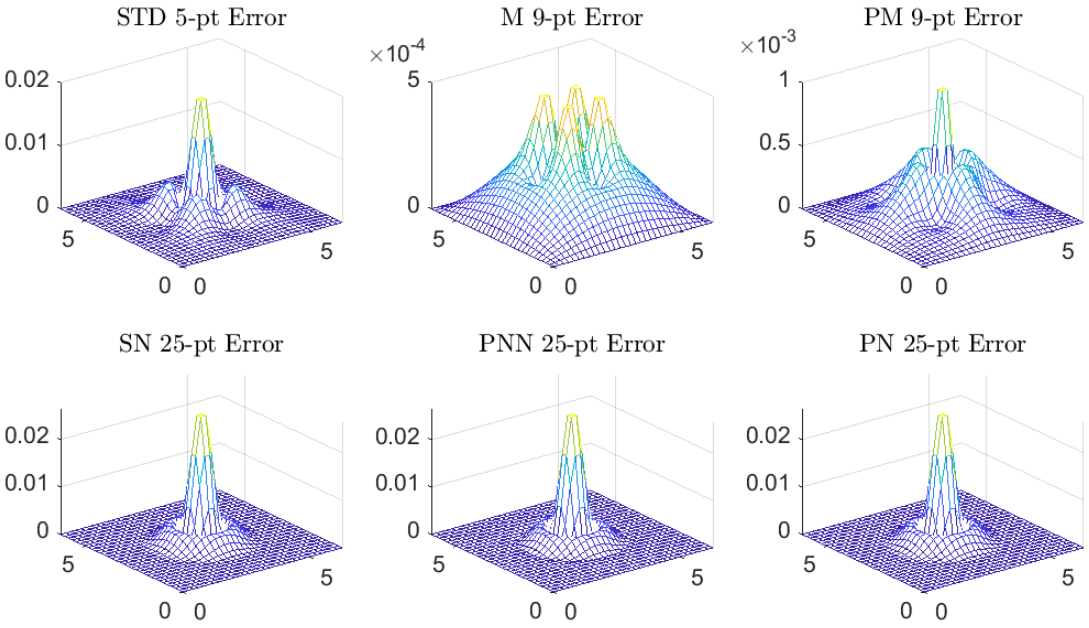


Figure 17: Plots of the orders of each operator for System 2.

hinting at an error decay of $\mathcal{O}(\Delta x^{3.5})$. This difference is again most likely due to the spherical symmetry of System 2, as opposed to the periodicity of System 1. Further, as we can see in Figure 18 below, because System 2 decays exponentially, the contribution of the error near the boundaries is essentially nonexistent, which could also contribute to the improved decay rate of the PNN and PN operators. The solution plots featured in Figure 18a attest to this assessment and are, again, identical to the exact solution in Figure 7. The error plots, on the other hand, have a variety of shapes. Beginning with the bottom three plots in Figure 18b, they are all spherically symmetric, which is a feature that was expected for all of the



(a) Plots of the solutions calculated by the operators.



(b) Plots of the errors of the operators.

Figure 18: These are the solution and error plots for each of the operators for System 2.

operators. However, the errors on the top row are more radially symmetric, where the difference between radial and spherical symmetry is rotational. Also, the highest error lies in the center for all operators, and more so near where the solution changes concavity for the M operator.

4.4 System 3

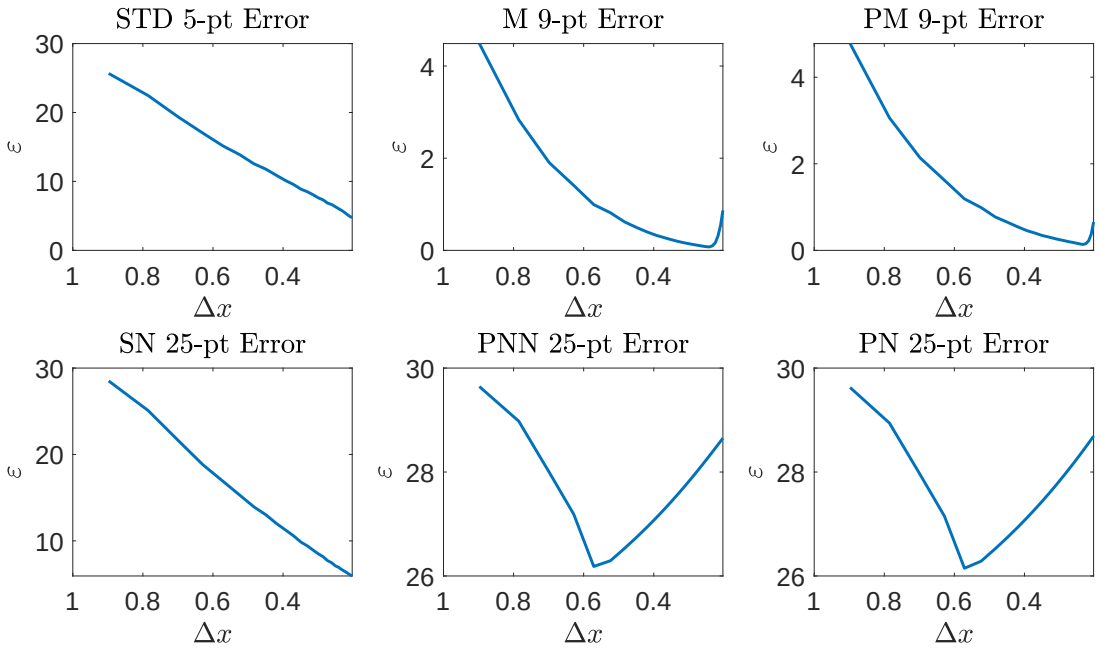


Figure 19: The errors of each solver as the step-size decreases for System 3.

For System 3 we find, in Figure 19, that the behavior of the errors looks more like that in Figure 13, that is, a smooth decrease as opposed to an oscillatory decrease. Further, notice that only the STD and SN operators have a linear decrease in the error as the step-size decreases. The PNN and PN operators begin to decrease, but then increase as the step-size decreases, which looks like a combination of the behaviors described in Systems 1 and Systems 2. The M and PM operators have a much smaller error scale than the other four operators and they also decrease much faster than the other methods as well, as expected. However, we can also see that moving beyond a step-size of about 0.2 leads to an increase

in the error, even with 1000 iterations for convergence. In examining the orders, reported in Figure 20 below, we can see that the order of the STD and SN have an order of about 1, which is consistent with Systems 1 and 2. The PNN and PN operators have an order of approximately zero, meaning that the error for this particular system decreases as $\mathcal{O}(\Delta x)$. Then, like with System 2, the orders for the M and PM operators are 3 and 2.4, with decays in error of $\mathcal{O}(\Delta x^4)$ and $\mathcal{O}(\Delta x^{3.4})$, respectively.

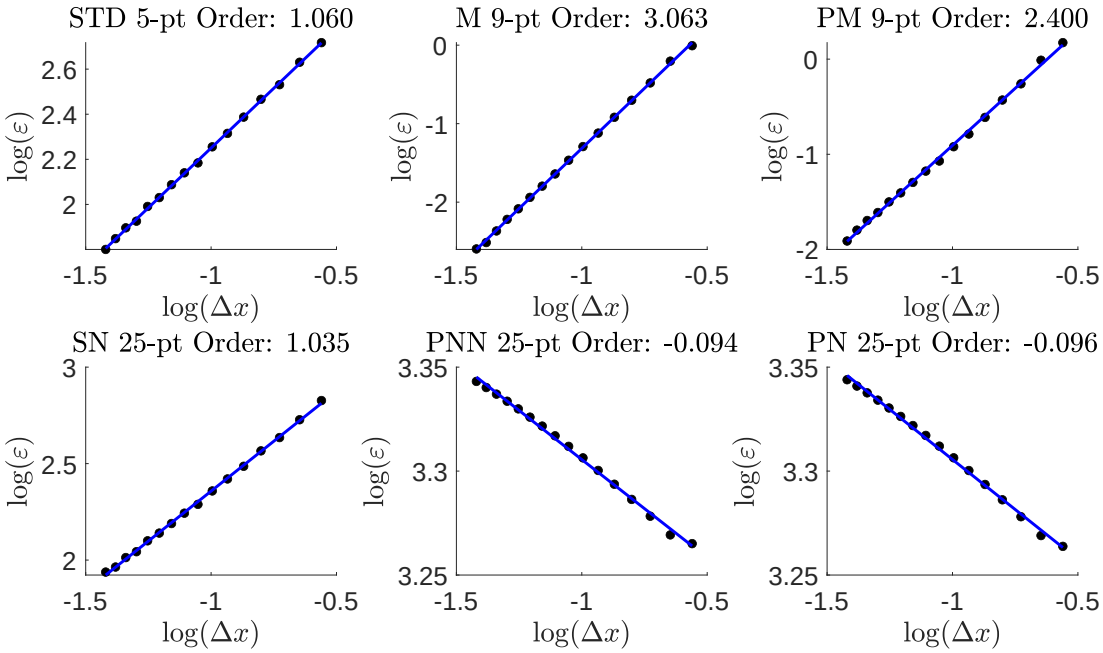
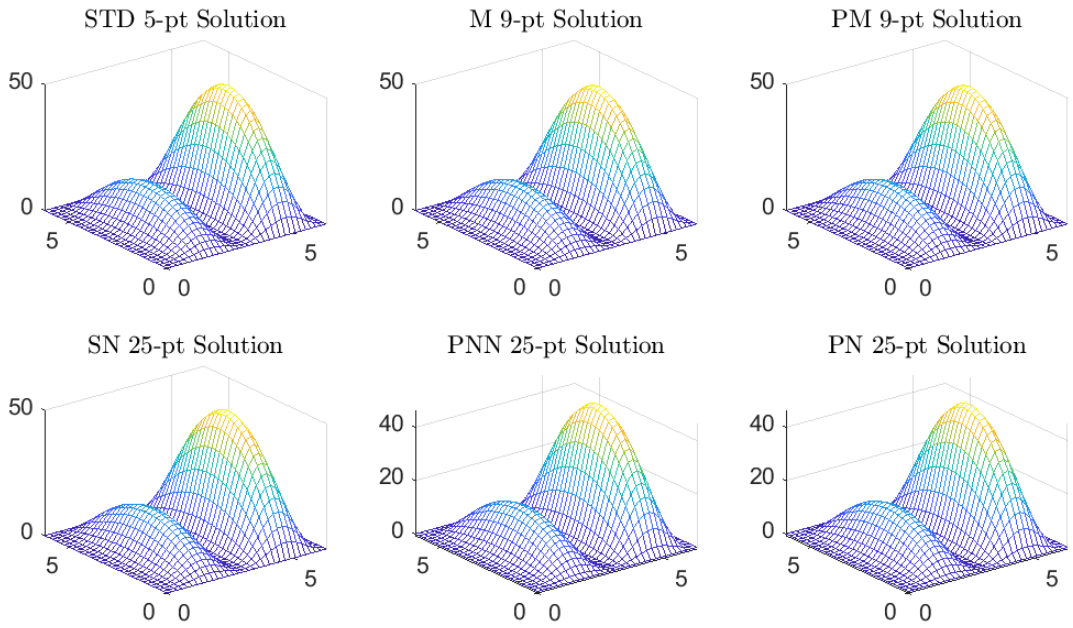
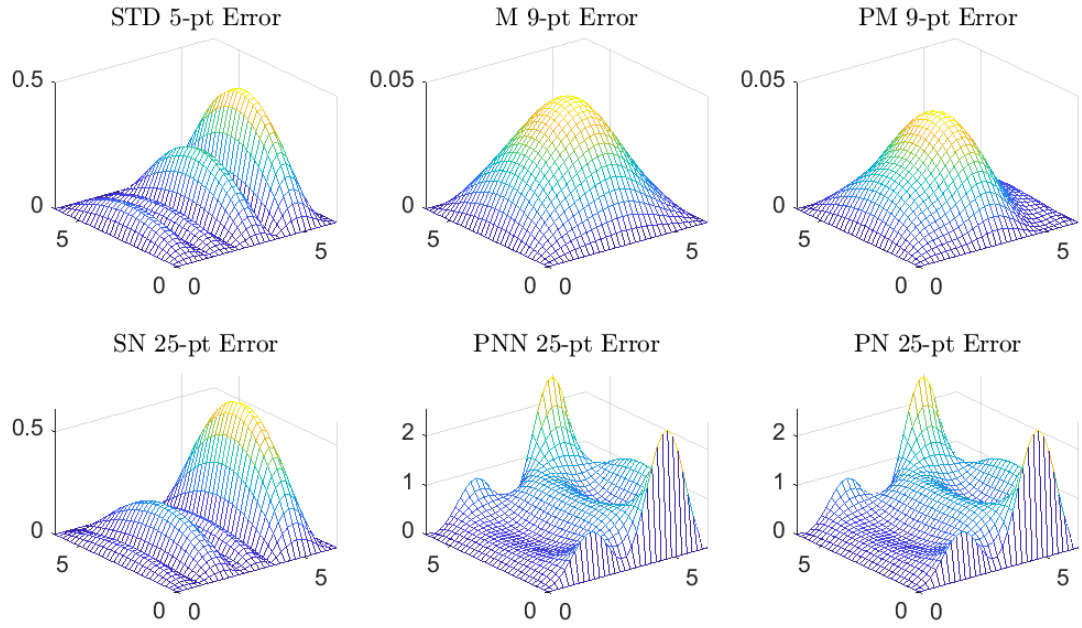


Figure 20: Plots of the orders of each operator for System 3.

The plots of the solutions and errors for each solver are below in Figure 21. Much like the other systems, the solver solution plots in Figure 21a are identical to the plot of the exact solution in Figure 7. However, the error plots in Figure 21b show properties similar to the System 1 errors. The errors for the STD and SN operators are most similar, with differences mainly in the first peak and trough of the system. Secondly, we see that only two of the four boundaries for the PNN and PN operators are nonzero, compared to the errors in System 1. For the last two operators, the M and PM operators, the error scales are comparable and the error of the M operator is more uniform, compared to the error of the PM operator.



(a) Plots of the solutions calculated by the operators.



(b) Plots of the errors of the operators.

Figure 21: These are the solution and error plots for each of the operators for System 3.

4.5 System 4

To analyze System 4, we analyze each region together because the results are almost identical for each operator. We can see in the next three figures, Figures 22, 23, and 24, the errors for the square, circular, and triangular regions, respectively, the errors steadily increase for every method. Further, the errors increase in an oscillatory manner as opposed to decreasing from what we observed in System 2. Although not obvious, there is an interesting result contrary to what has been observed for the PNN and PN operators for the previous Systems. The mean errors of the PNN and PN operators averaged over all matrix indices, from 6 to 30, are consistently lower than the other four operators. The orders for each of the methods are similar as well, provided in Figures 25, 26, and 27, for the square, circular,

Operator	**Square**	**Circle**	**Triangle**
STD	4.5147	2.8320	4.6304
M	4.6123	2.8781	4.7307
PM	4.6059	2.8740	4.7245
SN	4.4588	2.7948	4.5724
PNN	4.4020	2.7684	4.5234
PN	4.4014	2.7681	4.5228
	Mean Error		

Table 4: Summary of the mean error of each operator over all matrix indices.

and triangular regions, respectively. We can see that the orders for the STD, SN, PNN, and PN operators are all about the same, with an order of approximately -1.2. The orders of the M and PM operators is close to -1.1. Further, despite the PNN and PN operators having consistently lower errors than the other operators, the M and PM operators still have the highest orders. In the context of negative orders, this means that when compared to the other operators, when the grid size decreases the error increases slower for the M and PM operators. Considering all these factors, the explanation as to why the PNN and PN operators have consistently lower errors, even though not by much, is most likely because System 4 is piecewise. This would mean that for both parameterized operators, the interior operators (ρ_1) are much more "perfect" compared to the near-boundary operators ($\rho_2 - \rho_6$).

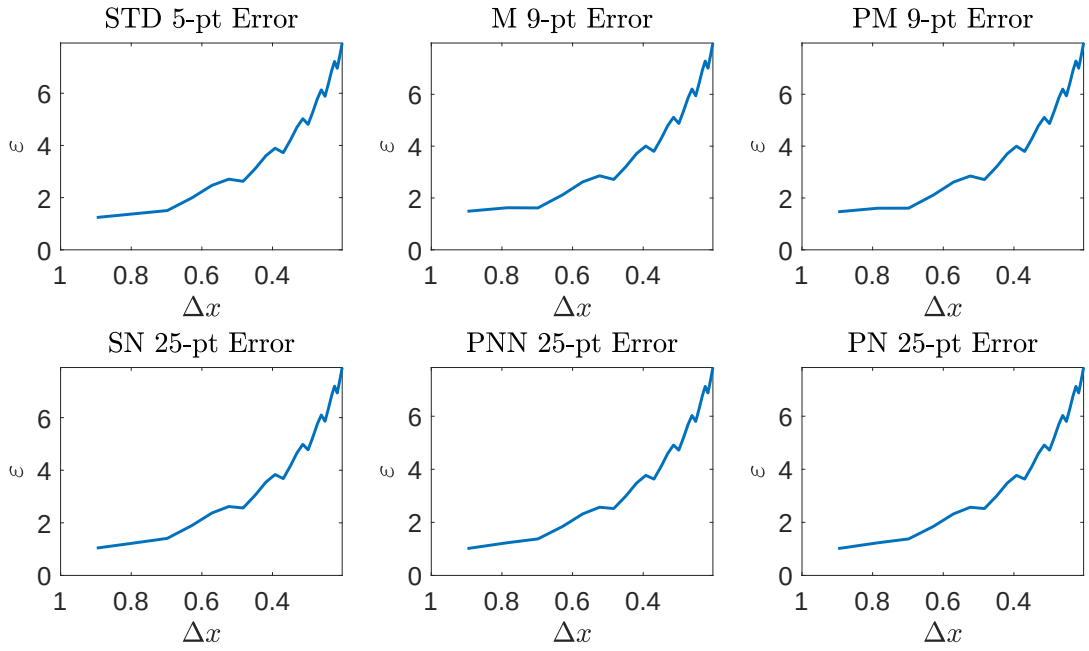


Figure 22: The errors of each solver as the step-size decreases for the square region.

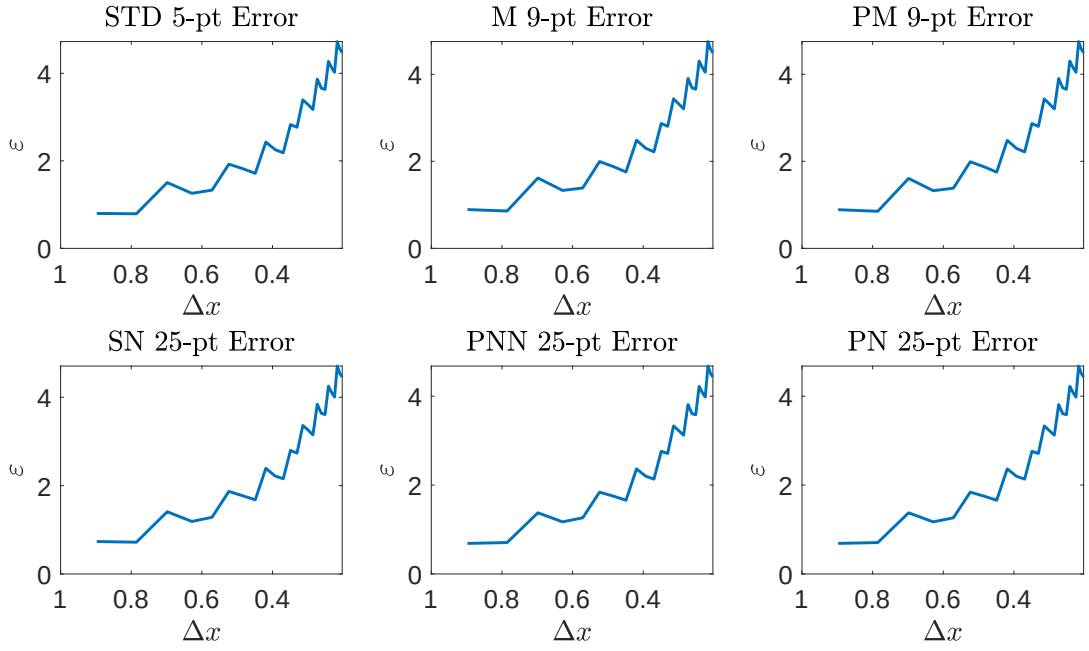


Figure 23: The errors of each solver as the step-size decreases for the circular region.

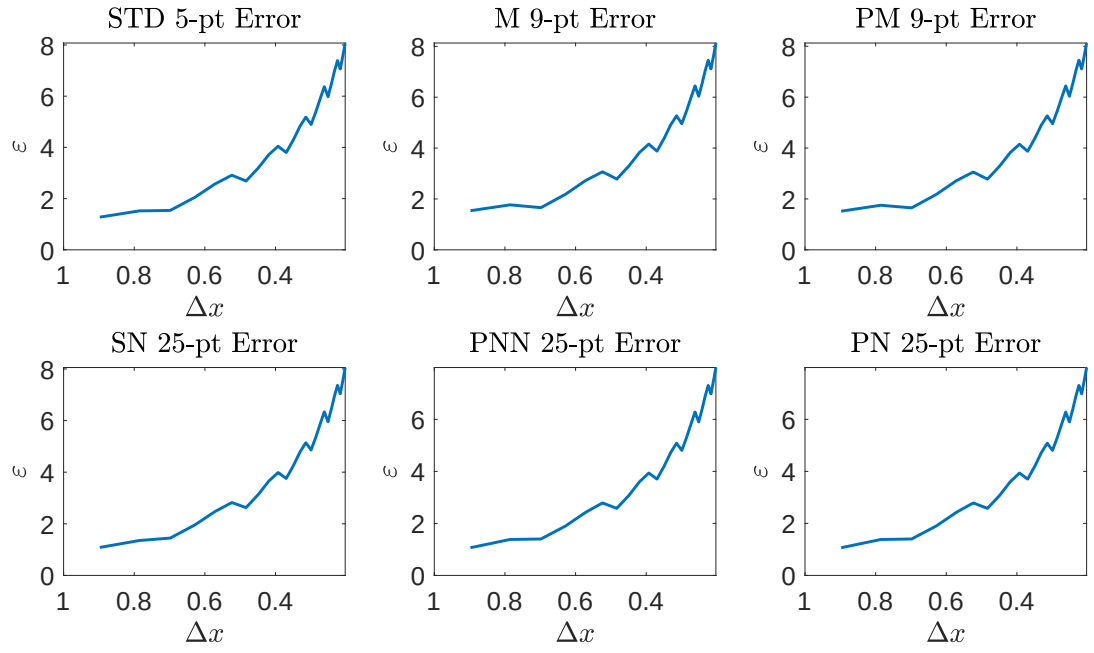


Figure 24: The errors of each solver as the step-size decreases for the triangular region.

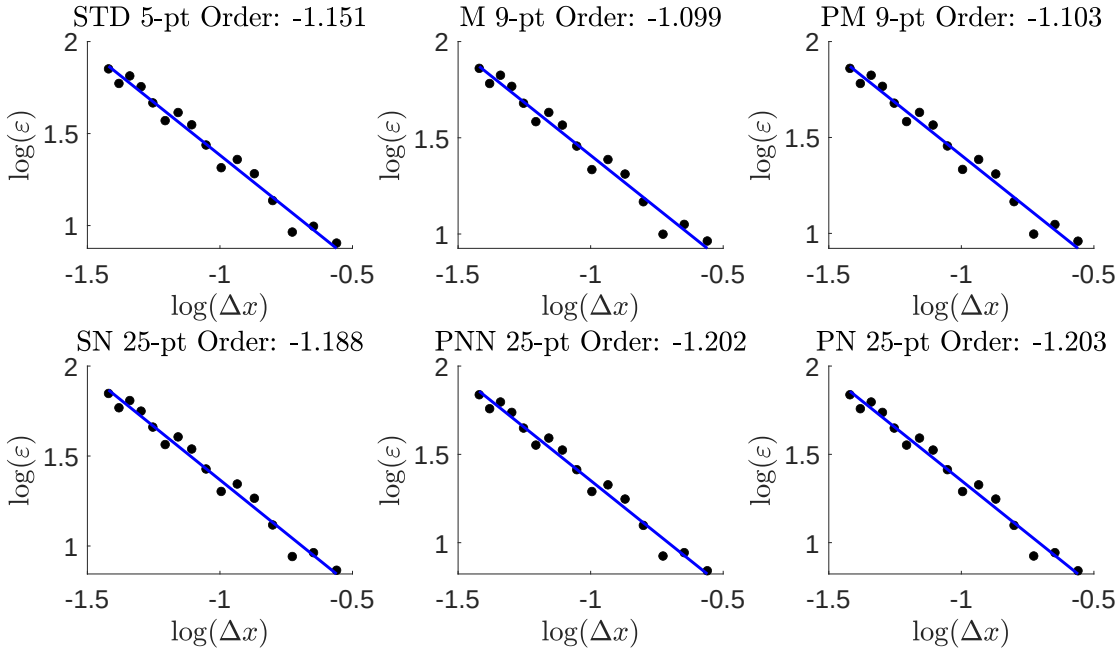


Figure 25: Plots of the orders of each operator for the square region.

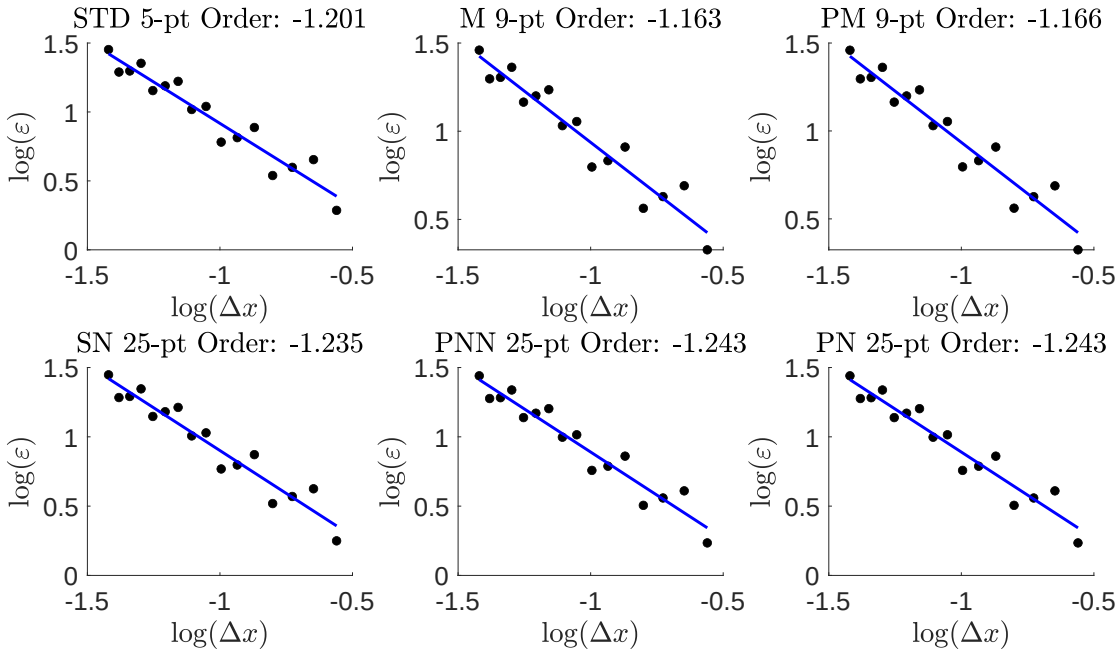


Figure 26: Plots of the orders of each operator for the circular region.

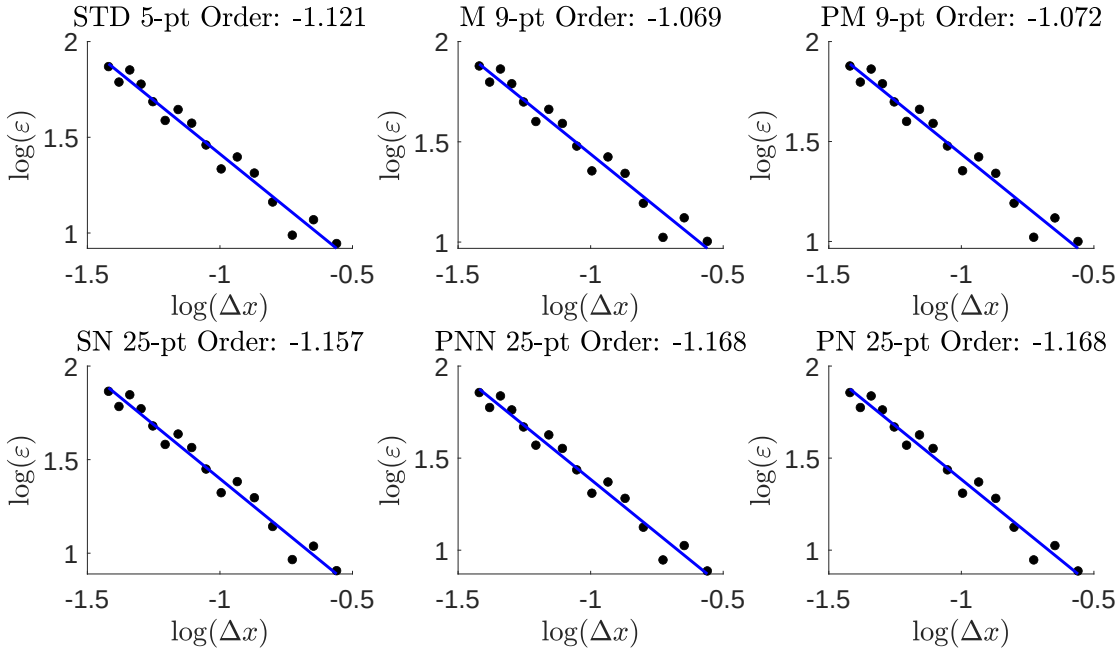


Figure 27: Plots of the orders of each operator for the triangular region.

Next, for System 4, we examine the solver solutions and the error plots, provided in in Figures 28, 29, and 30. Firstly, none of the solution plots perfectly resemble the plot of the exact solution, provided in Figure 7, like the solution plots for Systems 1-3. This is primarily due to the diffusive nature of the Jacobi method of iteration, in addition to the piecewise nature of the system. Further, the solutions for the square and circular regions look similar, aside from differing maximum values. Thus, with the diffusive nature of the Jacobi method, the visual differences between the solutions for the square and circular regions are lost. On the other hand, the solution for the triangular region most resembles the exact solution, but this is primarily due to how differently the triangular region is compared to the similarities between the square and circular regions. The plots of the errors for each of the solvers provides more structural information. For each of the three regions, the errors look the same for each operator. This is a direct consequence of the piecewise nature of System 4.

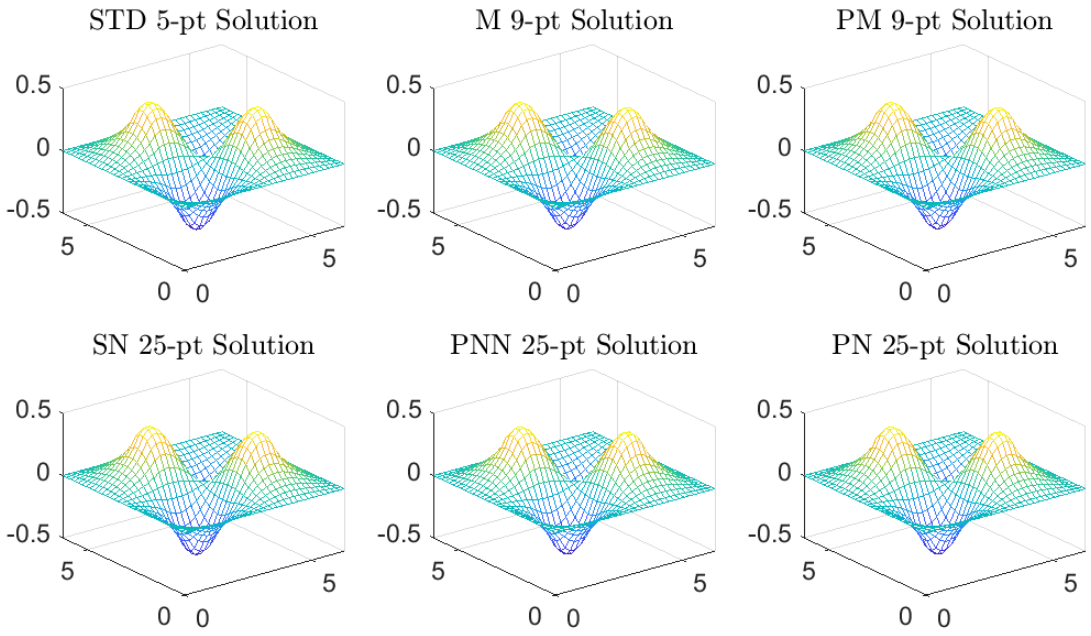
4.6 Summary of Analysis and Discussion

In Table 5 below, we summarize the rate of decay of each method for each system. For System 4, the rates of decay contain the averages of the orders for that operator because

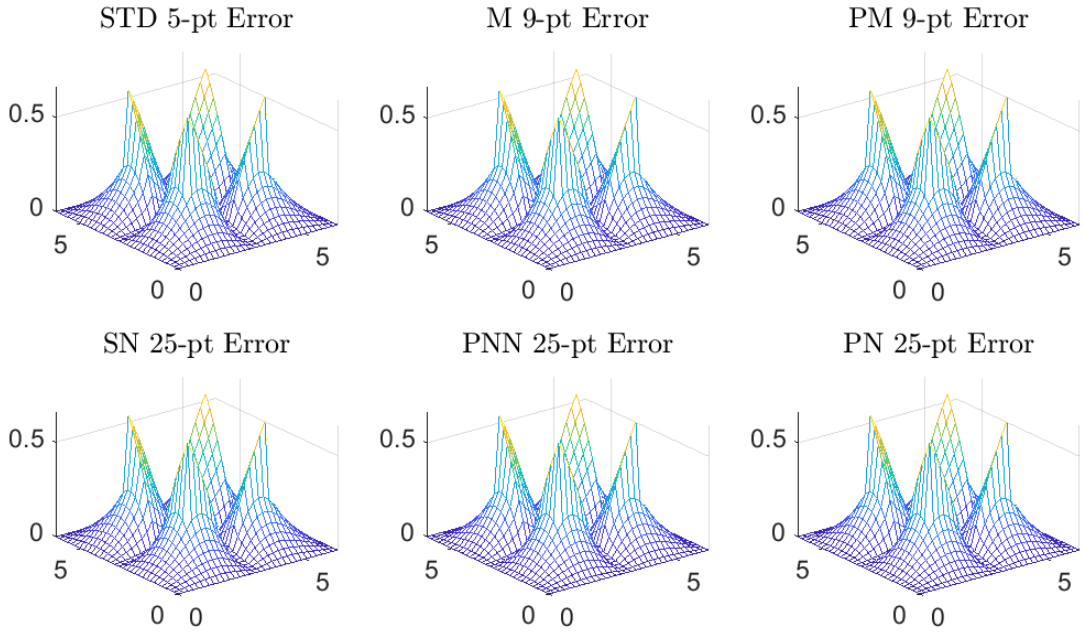
Operators	**System 1**	**System 2**	**System 3**	**System 4**
STD	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^{-0.2})$
M	$\sim \mathcal{O}(\Delta x^4)$	$\sim \mathcal{O}(\Delta x^4)$	$\sim \mathcal{O}(\Delta x^4)$	$\sim \mathcal{O}(\Delta x^{-0.1})$
PM	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^{3.5})$	$\sim \mathcal{O}(\Delta x^{3.4})$	$\sim \mathcal{O}(\Delta x^{-0.1})$
SN	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x^{-0.1})$
PNN	$\sim \mathcal{O}(\Delta x^{0.8})$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x)$	$\sim \mathcal{O}(\Delta x^{-0.2})$
PN	$\sim \mathcal{O}(\Delta x^{0.8})$	$\sim \mathcal{O}(\Delta x^2)$	$\sim \mathcal{O}(\Delta x)$	$\sim \mathcal{O}(\Delta x^{-0.2})$

Table 5: This summarizes the rate of decay of each of the operators for each system.

they were all similar. In comparing the perfect Laplace operators (PM, SN, PNN, PN) with the standard 5 and modified 9-point operators (STD, M), the STD and M operators were the most consistent and reliable operators to use when solving the Poisson systems. The errors were typically smaller and the orders larger. Their consistent results, aside from the piecewise System 4, where all the operators performed poorly, conveys the robustness of the

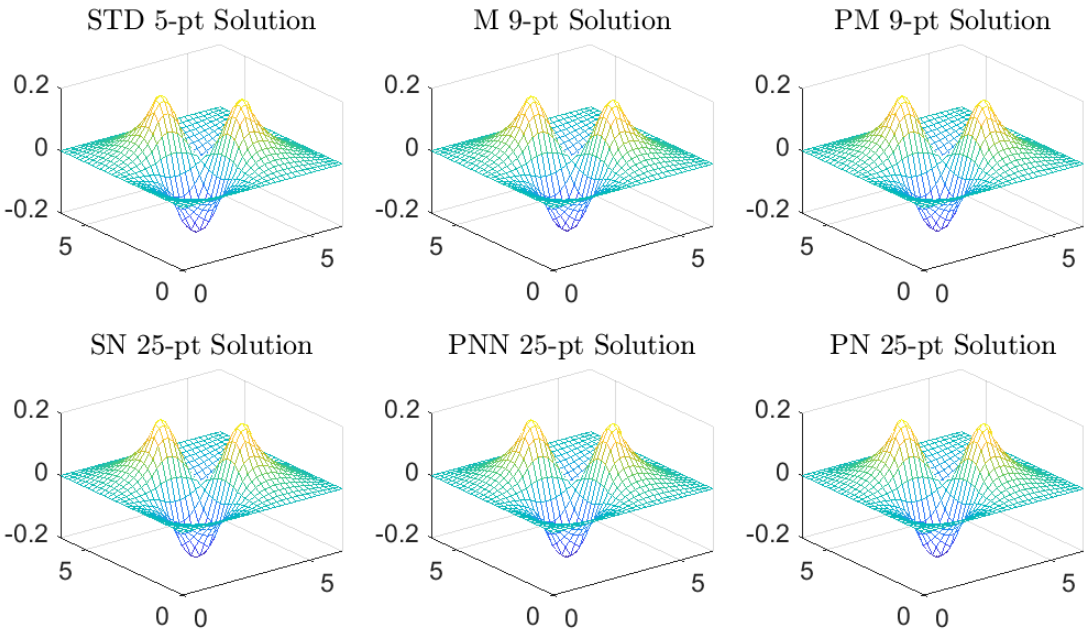


(a) Plots of the solutions calculated by the operators.

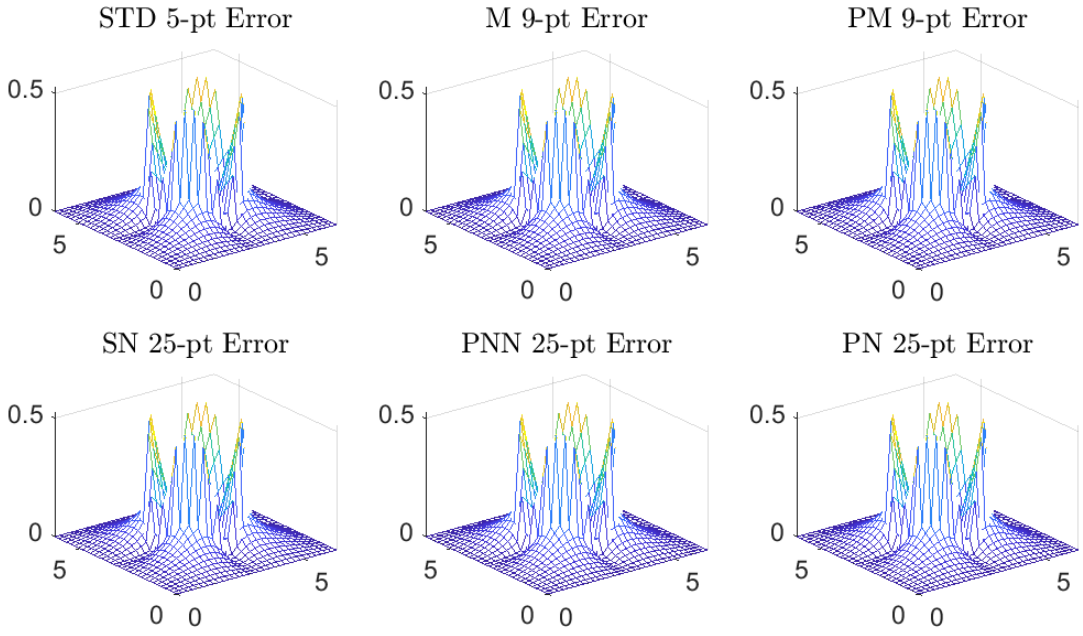


(b) Plots of the errors of the operators.

Figure 28: These are the solution and error plots for each of the operators for the square region of System 4.

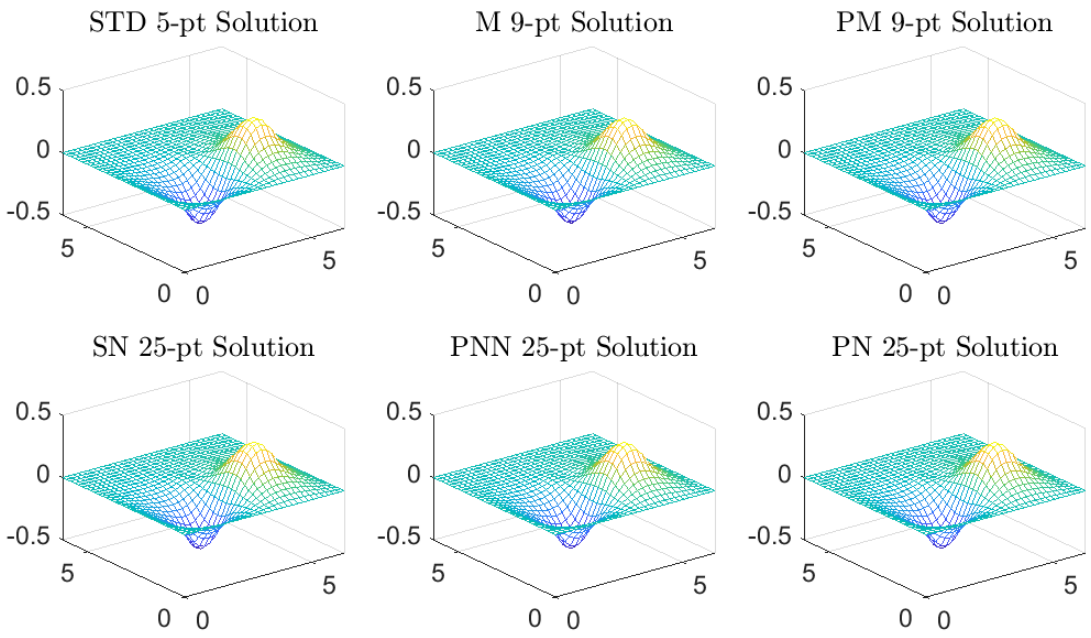


(a) Plots of the solutions calculated by the operators.

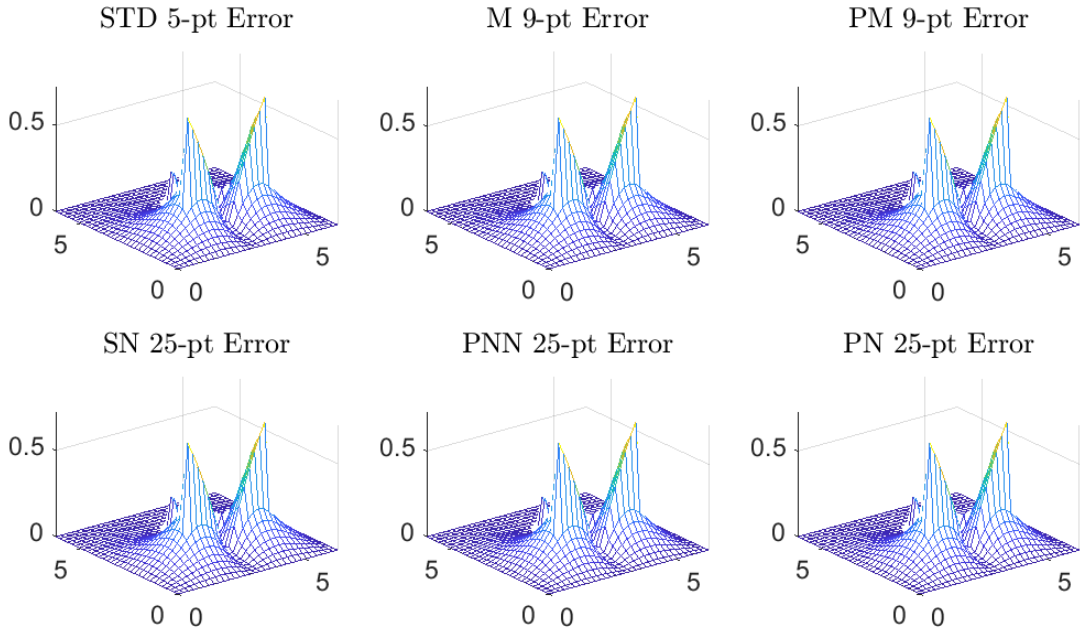


(b) Plots of the errors of the operators.

Figure 29: These are the solution and error plots for each of the operators for the circular System 4.



(a) Plots of the solutions calculated by the operators.



(b) Plots of the errors of the operators.

Figure 30: These are the solution and error plots for each of the operators for the triangular region of System 4.

STD and M operators across different systems compared to the perfect operators. Surprisingly, the SN operator was also consistent for Systems 1-3, showing promise for the perfect Laplace operators. In looking at the graphs of the errors for each method and Systems 1-3, we find that, in agreement with the rates of decay, the M operator decayed the fastest, followed by the PM, STD, and SN operators, with the parameterized PNN and PN operators last. In the construction of the unique perfect Laplace operators (PM, SN, PNN), the idea was to find an operator that would outperform the STD and M operators. Instead, the SN operator is only comparable to the STD operator, even though 15 additional points are used. Even still, the STD operator always had a slightly higher order than the SN operator. In a worse manner, the PM operator, modeled after the M operator, only made the M operator worse with the use of couplings from the perfect Laplace operator. Additionally, the PM operator was inconsistent, but not as inconsistent as the PNN and PN operators.

For the PNN and PN operators, these two were the most inconsistent operators. The only times that the PNN or PN operators performed somewhat decently was with System 4. However, this is most likely due to the piecewise nature. As such, the boundaries made no contribution to neither the error and nor the order of the methods. This suggests that the interior operator (ρ_1) is more accurate than those closer to the boundaries ($\rho_2 - \rho_6$). This is in agreement with what we observe for the SN operator, which only utilizes the ρ_1 operator. It would be worthwhile to revisit how the PNN and PN operators were constructed. The PNN operator was constructed by utilizing the values listed in Table 2 in conjunction with equations (2.24) and (2.25) to construct the operators $\hat{\rho}_2 - \hat{\rho}_6$. Likewise, for the PN operator, the ρ_1 operator was derived first and then normalized. Then the external operators, $\rho_2 - \rho_6$, were constructed from ρ_1 also using equations (2.24) and (2.25). Thus, it is a possibility that they were implemented sub optimally. The STD, M, and PM operators were written explicitly as sums (see this code block in Section 3.2), whereas the perfect Laplace operators were written using matrix-vector operations for convenience. Although the code accomplishes the same thing in both cases, it would be interesting to implement the SN, PNN, and PN operators using direct sums. However, the fact that all operators successfully solved each

system serves as an indicator that this may not be the case. Another potential issue that may arise is the use of the Jacobi method of iteration, for which solutions converge slowly compared to the Gauss-Seidel or SOR methods. There may be better methods with which the perfect Laplacian couplings may be used to solve the Poisson or Laplace equations.

In mentioning the Laplace equation, we encounter the most important issue concerning all the operators involved in the analysis: the applicability of the solvers with different types of equations, which includes the boundary conditions. All the operators successfully produced solutions to the Poisson equation with homogeneous boundary conditions. But suppose we wanted to analyze the Poisson equation with Neumann or Robin boundary conditions? Or, what if we wanted to analyze the Laplace equation, considering that the Laplace equation needs but one nonzero boundary condition on one side of the square? This is perhaps the greatest disadvantage of the SN, PNN, and PN operators, since the perfect Laplacian was derived in the context of trivial (zero/homogeneous) Dirichlet boundary conditions. On the other hand, the STD, M, and PM operators can solve (virtually) any Laplace or Poisson system, with non-trivial boundary conditions if the discretized system is well-conditioned.

5 Conclusion

In conclusion, based on the performance of the perfect Laplace operators in solving the systems described, it is far better to utilize the standard 5 and modified 9-point Laplace operators obtained from finite differences. In comparing the finite difference Laplace operators with the perfect Laplace, the robustness of the former operators was confirmed and the inconsistency of the latter was discovered, due to the potential of a sub optimal implementation of the operators. However, given that the perfect operators successfully produced solutions to each of the systems analyzed indicates otherwise. Additionally, it would be worthwhile to continue exploring with perfect Laplace operators. As a future endeavor, rewriting the perfect Laplace operators as direct sums instead of using matrix-vector operations would be a reasonable next step. However, as it stands, the standard stencils are the optimum discrete operators to use, despite the exact nature of the perfect Laplacian.

References

[1] Blatter, C. (2011). Intuitive interpretation of the Laplacian. Retrieved 17 October 2020, from https://math.stackexchange.com/questions/50274/intuitive-interpretation-of-the-laplacian

[2] Delamotte, B. (2004). A hint of renormalization. *American Journal of Physics*, 72(2), 170–184. https://doi.org/10.1119/1.1624112

[3] Gallego, F. M. (2018). *Navier-Stokes equations. Some real world problems arising in Fluid Mechanics.* [Lecture Slides]. University of Cádiz, Cádiz, Spain. https://www.imus.us.es/DOC-COURSE18/assets/media/docs/Unit1/Course1_Session1.pdf

[4] Hasenfratz P., Kleefeld F., & Kraus T. (1997). *Fascinating field theory: Quantum field theory, renormalization group, and lattice regularization.* In: Lenz F., Grießhammer H., Stoll D. (eds.), *Lectures on QCD.* Lecture Notes in Physics, vol 481. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0105683

[5] Hasenfratz, P., & Niedermayer, F. (1994). Perfect lattice action for asymptotically free theories. Nuclear Physics B, 414(3), 785-814. doi: 10.1016/0550-3213(94)90261-5

[6] Hauswirth, S. (2000, March 14). *Perfect Discretizations of Differential Operators.*

[7] Iserles, A. (2009). *A first course in the numerical analysis of differential equations.* Cambridge: Cambridge University Press.

[8] Kuhlmann, M. (2020). *History of Field Theory.* In M. Kuhlmann (ed.), *The Stanford Encyclopedia of Philosophy.* https://plato.stanford.edu/entries/quantum-field-theory/qft-history.html

[9] Kuhlmann, M. (2020). *Quantum Field Theory.* In M. Kuhlmann (ed.), *The Stanford Encyclopedia of Philosophy.* https://plato.stanford.edu/entries/quantum-field-theory/index.html

[10] LeVeque, R. (2007). *Finite difference methods for ordinary and partial differential equations.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

[11] Schey, H. M. (2005). *Div, grad, curl, and all that: An informal text on vector calculus* (4th ed.). New York: W. W. Norton.

[12] Schweber, S. (2002). *Quantum Field Theory: From QED to the Standard Model.* In M. Nye (Ed.), The Cambridge History of Science (The Cambridge History of Science, pp. 375-393). Cambridge: Cambridge University Press. doi:10.1017/CHOL9780521571999.021

[13] Sigal, I. M. (2019). *Lectures on Applied Partial Differential Equations.* [Lecture Notes]. University of Toronto, Toronto, Ontario. http://www.math.toronto.edu/sigal/ApplPDE2019S.pdf

[14] Tong, D. (2006). *Quantum Field Theory.* [Lecture Notes]. University of Cambridge, Cambridge, England. https://www.damtp.cam.ac.uk/user/tong/qft/qft.pdf

[15] Weinberg, S. (1977). The Search for Unity: Notes for a History of Quantum Field Theory. *Daedalus*, 106(4), 17-35. http://www.jstor.org/stable/20024506

Appendix

The purpose of this appendix is to include the explicit GitHub link to all functions used in the analysis of this thesis:

https://github.com/lharrisw/Senior-Thesis-Perfect-Discretizations