Project 2: Possion's Equation

Aerospace 423: Computational Methods in Aerospace Engineering
Undergraduate Aerospace Engineering
University of Michigan, Ann Arbor

By: Dan Card, dcard@umich.edu
Date: March 27, 2020

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Overview

Poisson's equation arises in the modeling of many physical phenomena, including heat transfer, fluid dynamics, species transport, and structural dynamics. The model equation is

$$\nabla^2 u = f(\vec{x}), \tag{1}$$

where u is the unknown scalar state, and $f(\vec{x})$ is a known (given) function. In this project, I will solve this equation in two spatial dimensions on a square domain with homogeneous Dirichlet boundary conditions, as illustrated in Figure 1. I will use the finite difference method with various stencils, and I will implement both direct and iterative solution strategies.

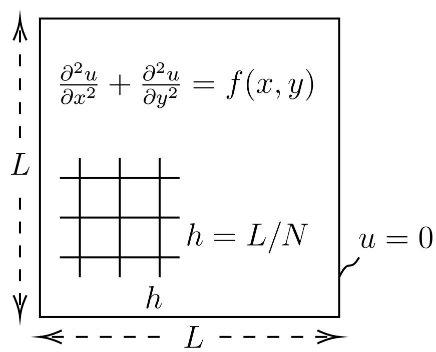

Figure 1: Poisson's equation in two dimensions.

Discretize Poisson's equation using a finite-difference method on a uniform grid of N intervals in each direction. This means that, including the boundaries, there are $(N+1)^2$ nodes. Of these, the $(N-1)^2$ interior nodes have unknown values of u. Since the domain is an $L \times L$ square, the spacing between nodes is $h = L/N$ in both the horizontal and the vertical directions. Use $L = 1$ for this project. At each interior node, there will be one unknown and one equation.

1.2 Stencils

I will analyze and implement three finite-difference stencils for solving Equation 1. At node (i, j) of the grid, these are defined as follows:
1. **5-point stencil**:

$$\frac{1}{h^2}\begin{bmatrix} & +u_{i,j+1} & \\ +u_{i-1,j} & -4u_{i,j} & +u_{i+1,j} \\ & +u_{i,j-1} & \end{bmatrix} = f_{i,j}$$

2. **9-point stencil**:

$$\frac{1}{6h^2}\begin{bmatrix} u_{i-1,j+1} + 4u_{i,j+1} + u_{i+1,j+1} \\ +4u_{i-1,j} - 20u_{i,j} + 4u_{i+1,j} \\ +u_{i-1,j-1} + 4u_{i,j-1} + u_{i+1,j-1} \end{bmatrix} = f_{i,j}$$

3. **Corrected 9-point stencil**:

$$\frac{1}{6h^2}\begin{bmatrix} u_{i-1,j+1} + 4u_{i,j+1} + u_{i+1,j+1} \\ +4u_{i-1,j} - 20u_{i,j} + 4u_{i+1,j} \\ +u_{i-1,j-1} + 4u_{i,j-1} + u_{i+1,j-1} \end{bmatrix} = f_{i,j} + \frac{1}{12}\begin{bmatrix} & +f_{i,j+1} & \\ +f_{i-1,j} & -4f_{i,j} & +f_{i+1,j} \\ & +f_{i,j-1} & \end{bmatrix}$$

When analyzing these stencils, I will use the two-dimensional Taylor-series expansion formula,

$$u(x+\Delta x, y+\Delta y) = \sum_{\substack{0\le l,m\le p \\ l+m\le p}} \frac{1}{l!m!}\frac{\partial^{l+m}u}{\partial x^l \partial y^m}\Delta x^l \Delta y^m \tag{2}$$

1.3 Solvers

Consider three solvers:

1. **Direct solver**: in which I build a sparse linear system of equations to solve for the nodal states. The system will take the form

   $$AU = F,$$

   where U is the unrolled state vector of the unknowns. I am free to choose how to unroll the state into one column vector (i.e. how to order the unknowns). Solve this system using a direct solver, such as the backslash operator in Matlab ($U = A \setminus F$).

2. **Jacobi iteration**: starting with an initial guess for the unknown state, $u = u^0$, update it using the following iteration:

   $$u^{n+1}_{i,j} = u^n_{i,j} + \frac{\omega}{S_{0,0}} r^n_{i,j}, \quad \text{where} \quad r^n_{i,j} = \tilde{f}_{i,j} - \sum_{-1\le l,m\le 1} S_{l,m} u^n_{i+1,j+m}.$$

   In this equation, $\tilde{f}_{i,j}$ is the right-hand side of the stencil formula at node i, j. This is the same as $f_{i,j}$ for the 5 and 9-point stencils but includes the extra f finite difference for the corrected 9-point stencil. $S_{l,m}$ is the 3×3 stencil matrix (the factors on the left-hand sides in the given formulas), with the $(0, 0)$ index pair corresponding to the middle entry. $r^n_{i,j}$ are components of the *residual*. Also, ω is an under/over-relaxation factor: use $\omega = 1$ for the Jacobi iteration.

3. **Red-black Gauss-Seidel iteration**: starting with an initial guess for the unknown state, $u = u^0$, update it using the following two-step iteration:

$$\begin{aligned} u^{n+1/2} &= \text{Jacobi}(u^n, \text{at red nodes}) \\ u^{n+1} &= \text{Jacobi}(u^{n+1/2}, \text{at black nodes}) \end{aligned}$$

The red/black nodes are defined as illustrated in Figure 2, in a checker-board fashion. Note that in the first step, I apply the Jacobi iteration to the solution at the red nodes: these are the only ones that get updated. In the second step, I update the solution at the black nodes, *using the most recently-updated values at the red nodes*. This means that I actually do not need to store a separate state for the updated solution: I can perform the update in-place.

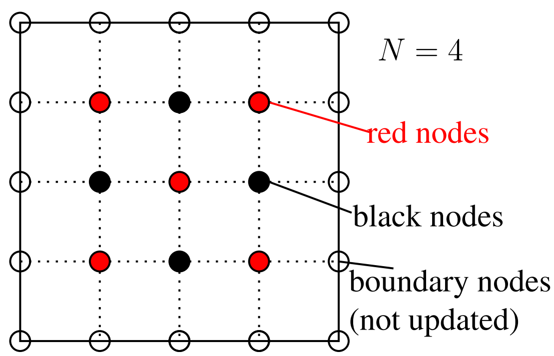

Figure 2: Red and black node designations for an $N = 4$ lattice.

1.4 Error Measure

Calculate the error in a discrete solution using an L_2 norm relative to an exact solution. On a discrete grid of $N \times N$ intervals, define the error norm as

$$L_2 error = E \equiv \sqrt{\frac{1}{L^2} \sum_{1 \le i,j \le N} h^2 (u_{i,j} - u^{\text{exact}}_{i,j})^2},$$

where $u^{\text{exact}}_{i,j}$ is an analytically-computed exact solution evaluated at node (i, j).

2 Questions and Tasks

2.1 Analytical Order of Accuracy Studies

Formulating Taylor Series

Firstly to conduct an Order of Accuracy study, I will Taylor series all the components:

$$
\begin{aligned}
u_{i-1,j} &= u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5) \\
u_{i+1,j} &= u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5) \\
u_{i,j-1} &= u - hu_y + \frac{h^2}{2}u_{yy} - \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5) \\
u_{i,j+1} &= u + hu_y + \frac{h^2}{2}u_{yy} + \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5) \\
u_{i-1,j-1} &= u - h(u_x + u_y) + \frac{h^2}{2}(u_{xx} + 2u_{xy} + u_{yy}) \\
&\quad - \frac{h^3}{6}(u_{xxx} + 3u_{xxy} + 3u_{xyy} + u_{yyy}) \\
&\quad + \frac{h^4}{24}(u_{xxxx} + 4u_{xxxy} + 6u_{xxyy} + 4u_{xyyy} + u_{yyyy}) + \mathcal{O}(h^5) \\
u_{i-1,j+1} &= u - h(u_x - u_y) + \frac{h^2}{2}(u_{xx} - 2u_{xy} + u_{yy}) \\
&\quad - \frac{h^3}{6}(u_{xxx} - 3u_{xxy} + 3u_{xyy} - u_{yyy}) \\
&\quad + \frac{h^4}{24}(u_{xxxx} - 4u_{xxxy} + 6u_{xxyy} - 4u_{xyyy} + u_{yyyy}) + \mathcal{O}(h^5) \\
u_{i+1,j-1} &= u + h(u_x - u_y) + \frac{h^2}{2}(u_{xx} - 2u_{xy} + u_{yy}) \\
&\quad + \frac{h^3}{6}(u_{xxx} - 3u_{xxy} + 3u_{xyy} - u_{yyy}) \\
&\quad + \frac{h^4}{24}(u_{xxxx} - 4u_{xxxy} + 6u_{xxyy} - 4u_{xyyy} + u_{yyyy}) + \mathcal{O}(h^5) \\
u_{i+1,j+1} &= u + h(u_x + u_y) + \frac{h^2}{2}(u_{xx} + 2u_{xy} + u_{yy}) \\
&\quad + \frac{h^3}{6}(u_{xxx} + 3u_{xxy} + 3u_{xyy} + u_{yyy}) \\
&\quad + \frac{h^4}{24}(u_{xxxx} + 4u_{xxxy} + 6u_{xxyy} + 4u_{xyyy} + u_{yyyy}) + \mathcal{O}(h^5)
\end{aligned}
$$

Theses Taylor Series expansions will be used to determine the order of accuracies for each stencil; 5-Point, 9-Point, and Corrected 9-point.

2.1.1 5-Point Stencil Order of Accuracy

Using the Taylor Series expansions given above, and the stencil's given results in the expression below,

$$f_{i,j} = \frac{1}{h^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right)$$

Substituting in the expansions result in,

$$\begin{aligned} f_{i,j} = &\frac{1}{h^2}(u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5)) \\ &+ \frac{1}{h^2}(u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5)) \\ &+ \frac{1}{h^2}(u - hu_y + \frac{h^2}{2}u_{yy} - \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5)) \\ &+ \frac{1}{h^2}(u + hu_y + \frac{h^2}{2}u_{yy} + \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5)) \\ &- \frac{1}{h^2}4u \end{aligned}$$

Simplifying and collecting terms results in,

$$\begin{aligned} f_{i,j} &= u_{xx} + u_{yy} + \frac{h^2}{12}(u_{xxxx} + u_{yyyy}) \\ f_{i,j} &\approx u_{xx} + u_{yy} + \mathcal{O}(h^2) \end{aligned}$$

Conducting the order of accuracy study shown above gives that the 5-Point Stencil is **Second-Order Accurate**.

2.1.2 9-Point Stencil Order of Accuracy

Again, like for the 5-Point Stencil, I will use the expression given for the stencil:

$$
\begin{aligned}
f_{i,j} &= \frac{1}{6h^2}(u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} + 4(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}) - 20u_{i,j}) \\
f_{i,j} &= \frac{1}{6h^2}\left(u - h(u_x + u_y) + \frac{h^2}{2}(u_{xx} + 2u_{xy} + u_{yy}) + \ldots + \mathcal{O}(h^5)\right) \\
&+ \frac{1}{6h^2}\left(u - h(u_x - u_y) + \frac{h^2}{2}(u_{xx} - 2u_{xy} + u_{yy}) + \ldots + \mathcal{O}(h^5)\right) \\
&+ \frac{1}{6h^2}\left(u + h(u_x - u_y) + \frac{h^2}{2}(u_{xx} - 2u_{xy} + u_{yy}) + \ldots + \mathcal{O}(h^5)\right) \\
&+ \frac{1}{6h^2}\left(u + h(u_x + u_y) + \frac{h^2}{2}(u_{xx} + 2u_{xy} + u_{yy}) + \ldots + \mathcal{O}(h^5)\right) \\
&+ \frac{4}{6h^2}\left(u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5)\right) \\
&+ \frac{4}{6h^2}\left(u - hu_y + \frac{h^2}{2}u_{yy} - \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5)\right) \\
&+ \frac{4}{6h^2}\left(u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \mathcal{O}(h^5)\right) \\
&+ \frac{4}{6h^2}\left(u + hu_y + \frac{h^2}{2}u_{yy} + \frac{h^3}{6}u_{yyy} + \frac{h^4}{24}u_{yyyy} + \mathcal{O}(h^5)\right) \\
&- 20u
\end{aligned}
$$

Simplifying and collecting terms results in,

$$
\begin{aligned}
f_{i,j} &= u_{xx} + u_{yy} + \frac{h^2}{12}(u_{xxxx} + 2u_{xxyy} + u_{yyyy}) \\
f_{i,j} &\approx u_{xx} + u_{yy} + \mathcal{O}(h^2)
\end{aligned}
$$

Conducting the order of accuracy study shown above gives that the 9-Point Stencil is **Second-Order Accurate**.

2.1.3 Corrected 9-Point Stencil Order of Accuracy

First I will determine what other Taylor expansions that must take place. I will do so from analyzing the Corrected 9-Point stencil:

$$
\begin{aligned}
f_{i,j} &= \frac{1}{6h^2}(u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} + 4(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}) - 20u_{i,j}) \\
&= u_{xx} + u_{yy} \\
f_{i-1,j} &= f_{i,j} - \Delta x\frac{\partial f_{i,j}}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 f_{i,j}}{\partial x^2} - \frac{\Delta x^3}{6}\frac{\partial^3 f_{i,j}}{\partial x^3} + \frac{\Delta x^4}{24}\frac{\partial^4 f_{i,j}}{\partial x^4} + \mathcal{O}(\Delta x^5) \\
f_{i+1,j} &= f_{i,j} + \Delta x\frac{\partial f_{i,j}}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 f_{i,j}}{\partial x^2} + \frac{\Delta x^3}{6}\frac{\partial^3 f_{i,j}}{\partial x^3} + \frac{\Delta x^4}{24}\frac{\partial^4 f_{i,j}}{\partial x^4} + \mathcal{O}(\Delta x^5) \\
f_{i,j-1} &= f_{i,j} - \Delta y\frac{\partial f_{i,j}}{\partial y} + \frac{\Delta y^2}{2}\frac{\partial^2 f_{i,j}}{\partial y^2} - \frac{\Delta y^3}{6}\frac{\partial^3 f_{i,j}}{\partial y^3} + \frac{\Delta y^4}{24}\frac{\partial^4 f_{i,j}}{\partial y^4} + \mathcal{O}(\Delta y^5) \\
f_{i,j+1} &= f_{i,j} + \Delta y\frac{\partial f_{i,j}}{\partial y} + \frac{\Delta y^2}{2}\frac{\partial^2 f_{i,j}}{\partial y^2} + \frac{\Delta y^3}{6}\frac{\partial^3 f_{i,j}}{\partial y^3} + \frac{\Delta y^4}{24}\frac{\partial^4 f_{i,j}}{\partial y^4} + \mathcal{O}(\Delta y^5)
\end{aligned}
$$

Realizing that $f_{i,j} = u_{xx} + u_{yy}$, the Taylor-Series expansions in terms of u become,

$$
\begin{aligned}
f_{i,j} &= u_{xx} + u_{yy} \\
f_{i-1,j} &= u_{xx} + u_{yy} - \Delta x(u_{xxx} + u_{xyy}) + \frac{\Delta x^2}{2}(u_{xxxx} + u_{xxyy}) - \frac{\Delta x^3}{6}(u_{xxxxx} + u_{xxxyy}) \\
&\quad + \frac{\Delta x^4}{24}(u_{xxxxxx} + u_{xxxxyy}) + \mathcal{O}(\Delta x^5) \\
f_{i+1,j} &= u_{xx} + u_{yy} + \Delta x(u_{xxx} + u_{xyy}) + \frac{\Delta x^2}{2}(u_{xxxx} + u_{xxyy}) + \frac{\Delta x^3}{6}(u_{xxxxx} + u_{xxxyy}) \\
&\quad + \frac{\Delta x^4}{24}(u_{xxxxxx} + u_{xxxxyy}) + \mathcal{O}(\Delta x^5) \\
f_{i,j-1} &= u_{xx} + u_{yy} - \Delta y(u_{xxy} + u_{yyy}) + \frac{\Delta y^2}{2}(u_{xxyy} + u_{yyyy}) - \frac{\Delta y^3}{6}(u_{xxyyy} + u_{yyyyy}) \\
&\quad + \frac{\Delta y^4}{24}(u_{xxyyyy} + u_{yyyyyy}) + \mathcal{O}(\Delta y^5) \\
f_{i,j+1} &= u_{xx} + u_{yy} + \Delta y(u_{xxy} + u_{yyy}) + \frac{\Delta y^2}{2}(u_{xxyy} + u_{yyyy}) + \frac{\Delta y^3}{6}(u_{xxyyy} + u_{yyyyy}) \\
&\quad + \frac{\Delta y^4}{24}(u_{xxyyyy} + u_{yyyyyy}) + \mathcal{O}(\Delta y^5)
\end{aligned}
$$

From here I will conduct a separate analysis of the left-hand side and right-hand side of the Corrected 9-Point stencil to determine the order of accuracy.

Looking back to the expressions for the Corrected 9-Point Stencil and using results from the 9-Point Order of Accuracy gives,

$$\frac{1}{6h^2}\begin{bmatrix} u_{i-1,j+1}+4u_{i,j+1}+u_{i+1,j+1} \\ +4u_{i-1,j}-20u_{i,j}+4u_{i+1,j} \\ +u_{i-1,j-1}+4u_{i,j-1}+u_{i+1,j-1} \end{bmatrix} = f_{i,j}+\frac{1}{12}\begin{bmatrix} +f_{i,j+1} \\ +f_{i-1,j}-4f_{i,j}+f_{i+1,j} \\ +f_{i,j-1} \end{bmatrix}$$

$$u_{xx}+u_{yy}+\frac{h^2}{12}(u_{xxxx}+2u_{xxyy}+u_{yyyy}) = f_{i,j}+\frac{1}{12}\begin{bmatrix} +f_{i,j+1} \\ +f_{i-1,j}-4f_{i,j}+f_{i+1,j} \\ +f_{i,j-1} \end{bmatrix}$$

Using the expressions for the $f_{i,j}, f_{i-1,j}, f_{i+1,j}, f_{i,j-1}, f_{i,j+1}$ terms knowing that $\Delta x = \Delta y = h$ results in,

$$\begin{aligned}
12\cdot\text{RHS} = {} & u_{xx}+u_{yy}-h(u_{xxx}+u_{xyy})+\frac{h^2}{2}(u_{xxxx}+u_{xxyy})-\frac{h^3}{6}(u_{xxxxx}+u_{xxxyy}) \\
& +\frac{h^4}{24}(u_{xxxxxx}+u_{xxxxyy})+\mathcal{O}(h^5) \\
& +u_{xx}+u_{yy}+h(u_{xxx}+u_{xyy})+\frac{h^2}{2}(u_{xxxx}+u_{xxyy})+\frac{h^3}{6}(u_{xxxxx}+u_{xxxyy}) \\
& +\frac{h^4}{24}(u_{xxxxxx}+u_{xxxxyy})+\mathcal{O}(h^5) \\
& +u_{xx}+u_{yy}-h(u_{xxy}+u_{yyy})+\frac{h^2}{2}(u_{xxyy}+u_{yyyy})-\frac{h^3}{6}(u_{xxyyy}+u_{yyyyy}) \\
& +\frac{h^4}{24}(u_{xxyyyy}+u_{yyyyyy})+\mathcal{O}(h^5) \\
& +u_{xx}+u_{yy}+h(u_{xxy}+u_{yyy})+\frac{h^2}{2}(u_{xxyy}+u_{yyyy})+\frac{h^3}{6}(u_{xxyyy}+u_{yyyyy}) \\
& +\frac{h^4}{24}(u_{xxyyyy}+u_{yyyyyy})+\mathcal{O}(h^5) \\
& -4(u_{xx}+u_{yy}) \\
\text{RHS} = {} & \frac{1}{144}h^2\left(h^2u_{xxyyyy}+h^2u_{yyyyyy}+12u_{xxxx}+24u_{xxyy}+12u_{yyyy}\right)
\end{aligned}$$

Then the expression becomes

$$\begin{aligned}
u_{xx}+u_{yy}+\frac{h^2}{12}(u_{xxxx}+2u_{xxyy}+u_{yyyy}) &= f_{i,j}+\begin{bmatrix} +\frac{1}{144}\left(h^4u_{xxyyyy}+h^4u_{yyyyyy}\right) \\ +\frac{1}{144}\left(12h^2u_{xxxx}+24h^2u_{xxyy}+12h^2u_{yyyy}\right) \end{bmatrix} \\
f_{i,j} &= u_{xx}+u_{yy}-\frac{h^4}{144}(u_{xxxxxx}+u_{xxxxyy}+u_{xxyyyy}+u_{yyyyyy}) \\
f_{i,j} &\approx u_{xx}+u_{yy}+\mathcal{O}(h^4)
\end{aligned}$$

Conducting the order of accuracy study shown above gives that the Corrected 9-Point Stencil is **Fourth-Order Accurate**.

2.2 Implementing Stencils

Implement all stencils using a direct matrix solver approach. For

$$f(x,y) = -20\pi^2 \sin(2\pi x)\sin(4\pi y)$$

perform a convergence study on grid sizes of $N = 8, 16, 32, 64$. Compute the exact solution analytically and plot the L_2 error versus h. Verify that you obtain the rates predicted in the previous part.

2.2.1 Solving the Exact Solution

Possion's Equation with the corresponding forcing function is given to be,

$$\nabla^2 u = -20\pi^2 \sin(2\pi x)\sin(4\pi y)$$

Say that

$$\begin{aligned} u &= A\sin(2\pi x)\sin(4\pi y) \\ u_{xx} &= -4\pi^2 A\sin(2\pi x)\sin(4\pi y) \\ u_{yy} &= -16\pi^2 A\sin(2\pi x)\sin(4\pi y) \end{aligned}$$

Substituting back into the original equation results in,

$$\begin{aligned} -4\pi^2 A\sin(2\pi x)\sin(4\pi y) + -16\pi^2 A\sin(2\pi x)\sin(4\pi y) &= -20\pi^2 \sin(2\pi x)\sin(4\pi y) \\ -4\pi^2 A - 16\pi^2 A &= -20\pi^2 \\ -20\pi^2 A &= -20\pi^2 \\ A &= 1 \end{aligned}$$

Then the exact solution is given as,

$$u_{\text{exact}} = \sin(2\pi x)\sin(4\pi y)$$

2.2.2 Convergence Study and Implementing Stencils

In this task, I will verify that the convergence rates are the same as the analytical convergence rates derived previously. I will implement an L_2 error to determine the convergence rates to the analytical solution. Below in Figure 3 is my implementation of this method and the results I concluded.

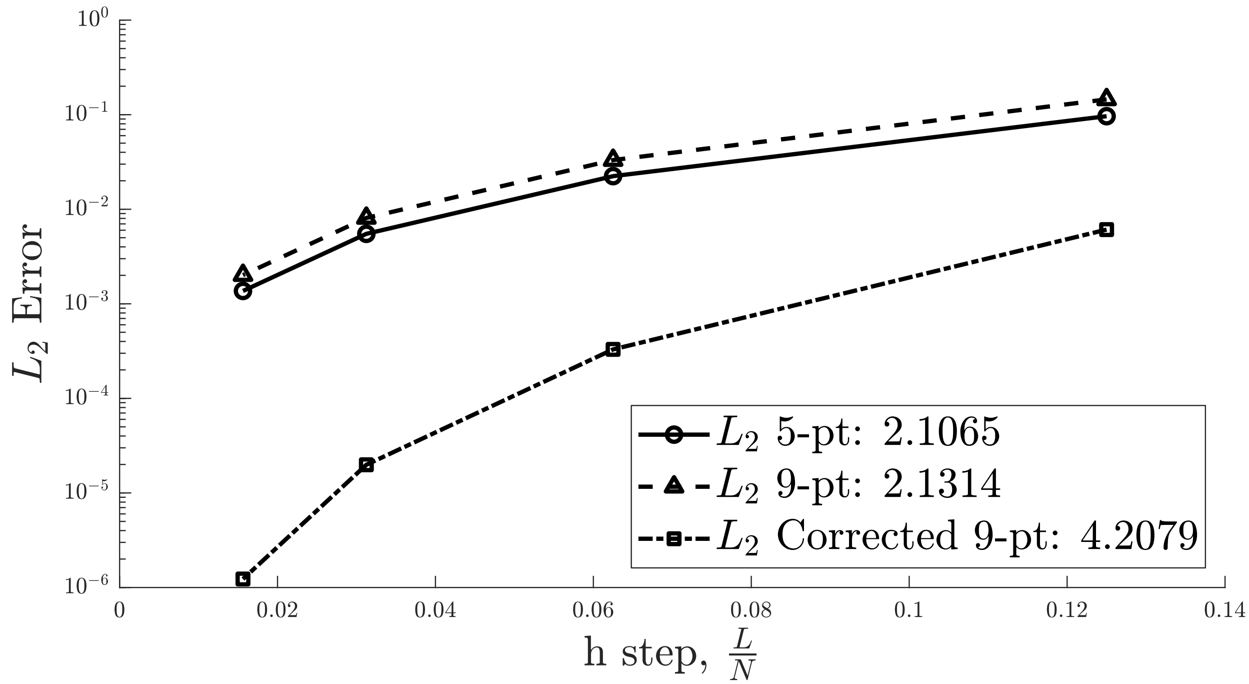

Figure 3: Implementing stencils for convergence study.

After conducting the L_2 error shown above in Figure 3, I successfully shown that the convergence rates of each stencil approximates to the analytical orders of accuracy shown in the previous task. For the 5-Point and the 9-Point stencils, the slope is approximated to be $\approx \mathcal{O}(h^2)$. For the corrected 9-Point stencil the order of accuracy was shown to be $\approx \mathcal{O}(h^4)$. These are denoted as the slopes shown in the legends for each corresponding stencil.

2.3 Implementing Jacobi Iterative Solver

In this task I will implement the Jacobi iterative solver for all stencils. Starting with a state identically equal to zero, and the same $f(x, y)$ function as in the previous part, I will run simulations for 200 iterations for each stencil, using $N = 16$ and $N = 32$.

2.3.1 Jacobi Iteration Residuals

Performing the Jacobi iteration, I solved for the L_2 norm of the residuals at each iteration. The result can be found below in Figure 4.

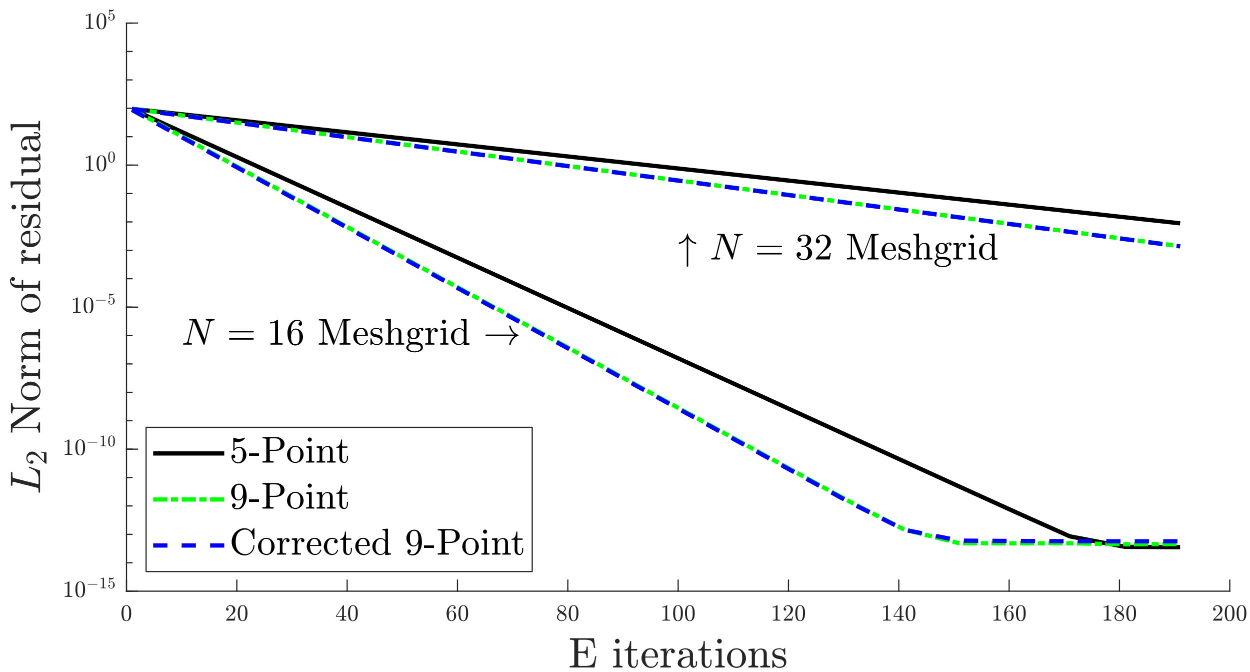

Figure 4: Residuals for Jacobi iteration.

Above in Figure 4, the plot shows that the larger mesh ($N = 32$) residual will converge at a rate that is approximately half that of the smaller grid ($N = 16$). This shows that the the rate of converge decreases as the number of sections in the meshgrid increases.

2.3.2 Jacobi Iteration Errors

Similar to the residuals, conducting the L_2 error through every iteration for every stencil for both meshes results in Figure 5.

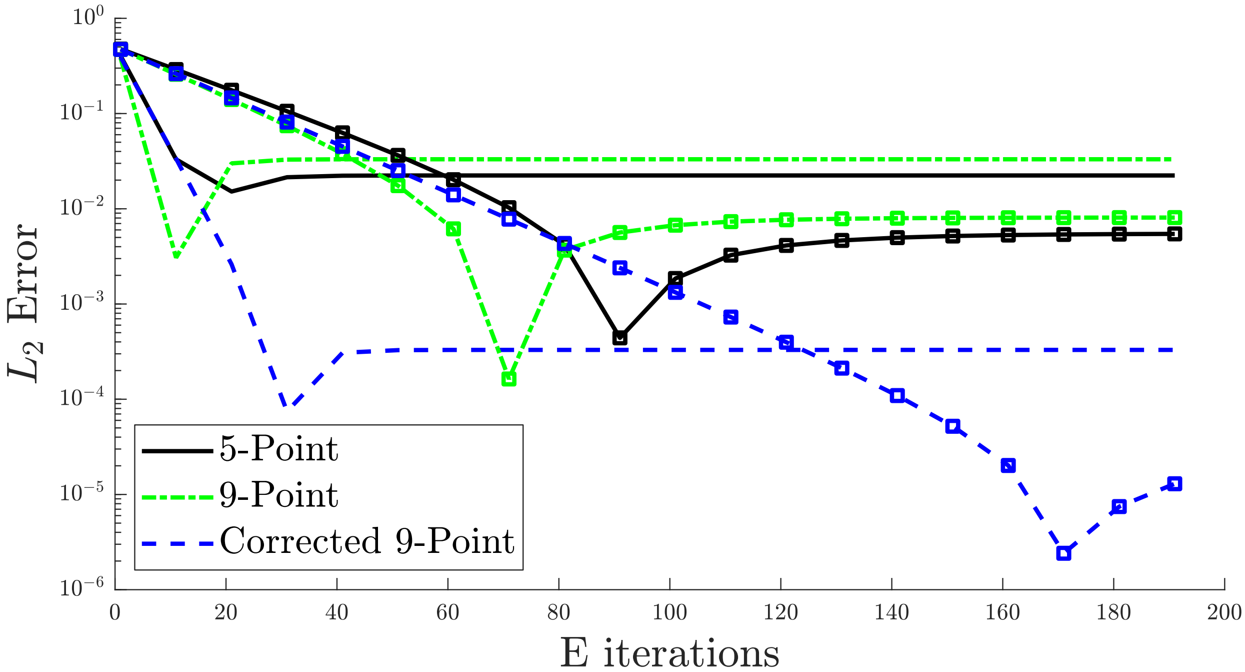

Figure 5: L_2 error for Jacobi iteration.

Note: □ are the markers for the $N = 32$ meshgrid.

Shown above in Figure 5, the larger the mesh, the slower the converge appears to be. **However**, the overall L_2 error appears to decrease by an order of magnitude. All reach a local minimum and then cusp out and reach a stable solution.

2.3.3 Meshgrid Size and Convergence

Shown from Figures 4,5 both show that the <u>rates</u> of converge are lower they will result in more refined approximate answers. Figure 4 shows best the inverse proportionality of the meshgrid size to the rate of convergence. Figure 5 highlights how the finer the mesh is, the more accurate(smaller) the L_2 error is.

2.4 Gauss-Seidel Iteration

In this task I will Repeat the previous part with the Gauss-Seidel iteration. Here, I will present results for $\omega = 1$ and $\omega = 1.7$. Discussing the relative performance of Gauss-Seidel versus Jacobi, and the effect of ω. In addition, I will indicate what happens when I run Jacobi with $\omega > 1$.

2.4.1 Gauss-Seidel With Over-Relaxation Value = 1

Gauss-Seidel Residuals

Below in Figure 6, is the L_2 norm of the residuals for the Gauss-Seidel iteration scheme. This plot highlights the convergence of the scheme by the rates at which the residual decreases.

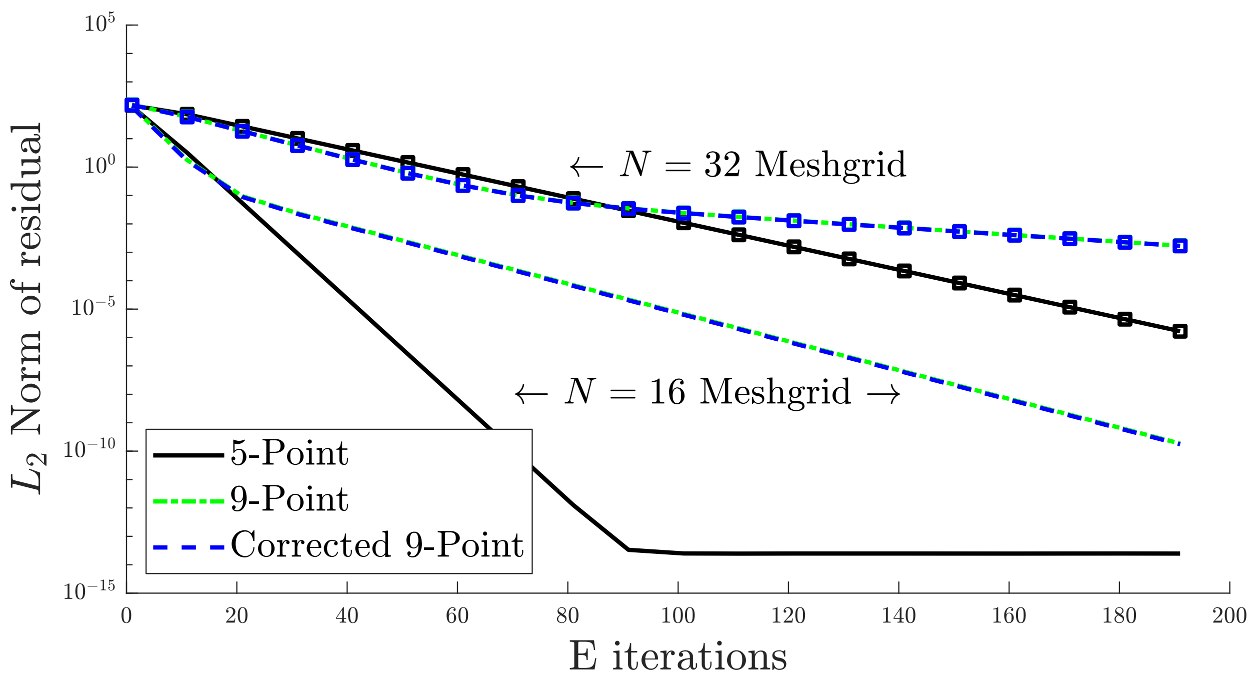

Figure 6: Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1$.

Similar to the Jacobi iteration scheme, this scheme also shows that the convergence rate is approximately half that of the smaller grid size by an order of magnitude. This change in rates can be viewed above in Figure 6, where the $N = 32$ meshgrid takes longer for the residuals to converge to their analytical answers.

Gauss-Seidel L_2 Error

Below in Figure 7, is the L_2 error of the approximated answer to the analytical solution for the Gauss-Seidel iteration scheme. This plot highlights the convergence of the scheme to the approximate solution and how accurate each stencil is once the scheme has converged to an approximate solution.

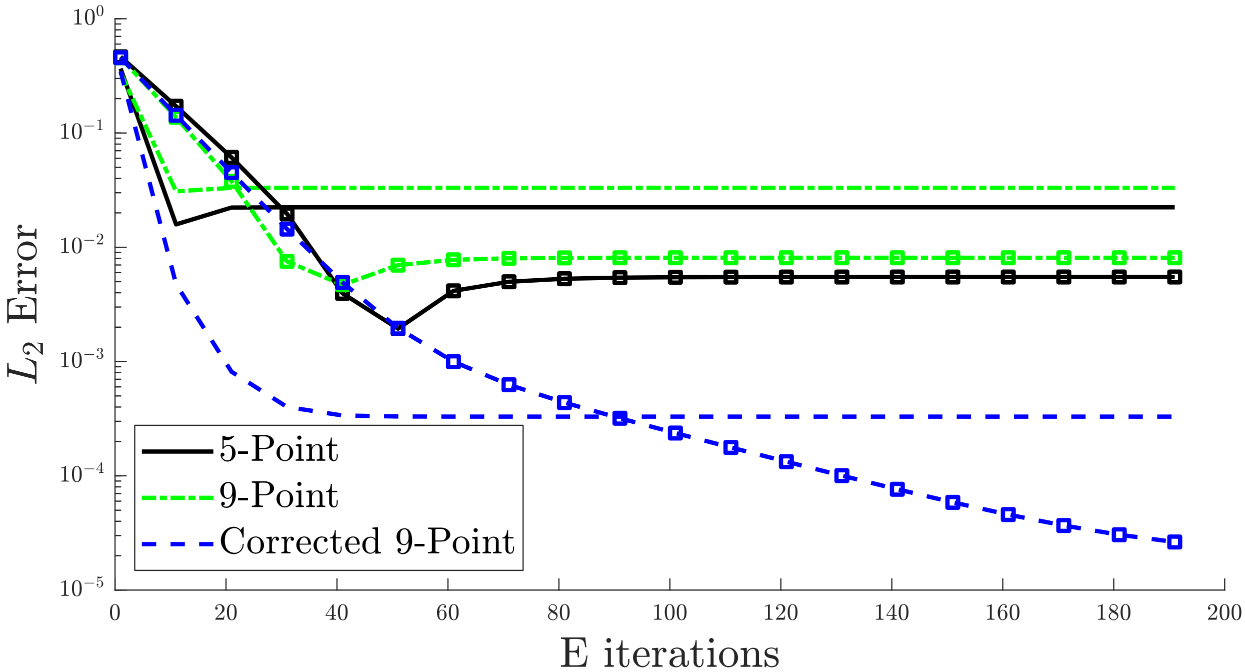

Figure 7: L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1$.

Note: □ are the markers for the $N = 32$ meshgrid.

In Figure 7, the approximate solutions converge to the analytical solutions with a decreased "over-shoot" of the converged answer when compared to the Jacobi iteration method. Also noteworthy, these schemes appear to converge to the asymptotic answers much faster than the Jacobi iteration method. Comparing Figure 5 to Figure 7 I found that for the $N = 16$ meshgrid for Gauss-Seidel converges to their asymptotic values approximately 10 iterations faster than Jacobi iteration method, when looking at the $N = 32$ meshgrid they converge 30 iterations faster.

Worth noting is that the corrected 9-point stencil converges monotonically to its asymptotic analytical solution without the "over-shoot" found with the other stencils. This monotonic decay is unique to the Gauss-Seidel iteration for this stencil and is not seen in the Jacobi iteration scheme or the direct solver.

2.4.2 Gauss-Seidel With Over-Relaxation Value = 1.7

Gauss-Seidel Residuals

Below in Figure 8, is the L_2 norm of the residuals for the Gauss-Seidel iteration scheme. This plot highlights the convergence of the scheme by the rates at which the residual decreases.

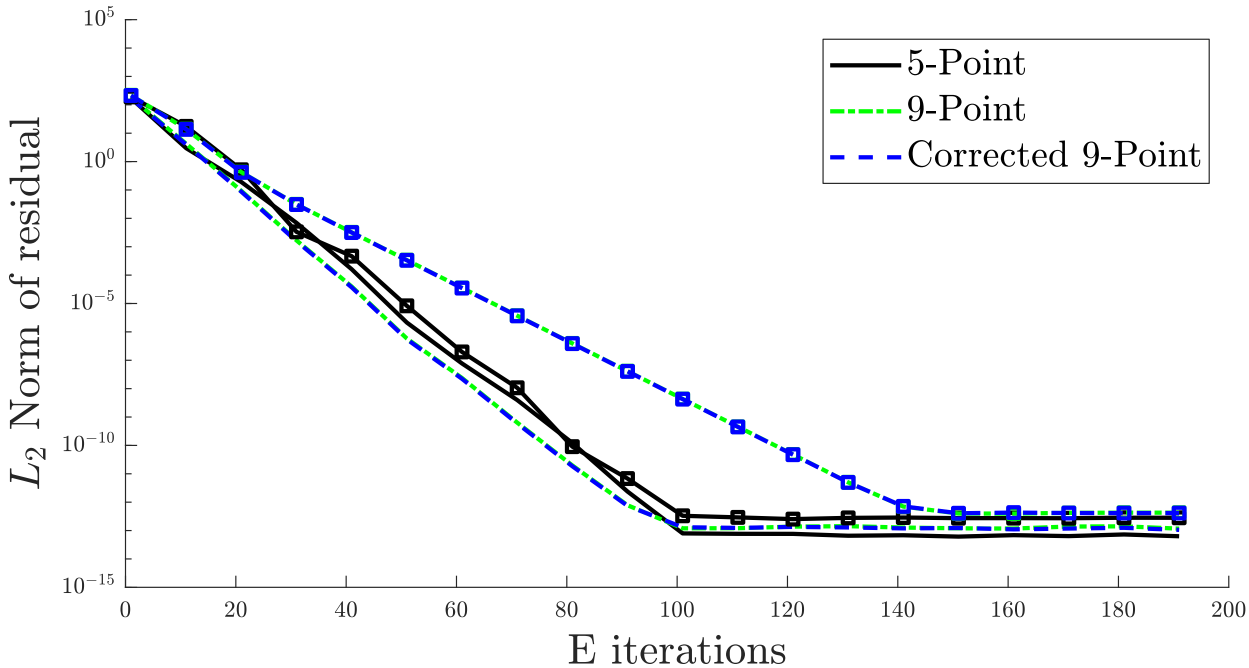

Figure 8: Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1.7$.

Note: $\square$ are the markers for the $N = 32$ meshgrid.

Similar to the Jacobi iteration scheme, this scheme also shows that the convergence rate is approximately half that of the smaller grid size by an order of magnitude. **However**, this correlation is not expressed with the 5-point stencil as both meshgrids follow the same rate of convergence. This change in rates can be viewed above in Figure 8, where the $N = 32$ meshgrid takes longer for the residuals to converge to their analytical answers.

Gauss-Seidel L_2 Error

Below in Figure 9, is the L_2 error of the approximated answer to the analytical solution for the Gauss-Seidel iteration scheme. This plot highlights the convergence of the scheme to the approximate solution and how accurate each stencil is once the scheme has converged to an approximate solution.

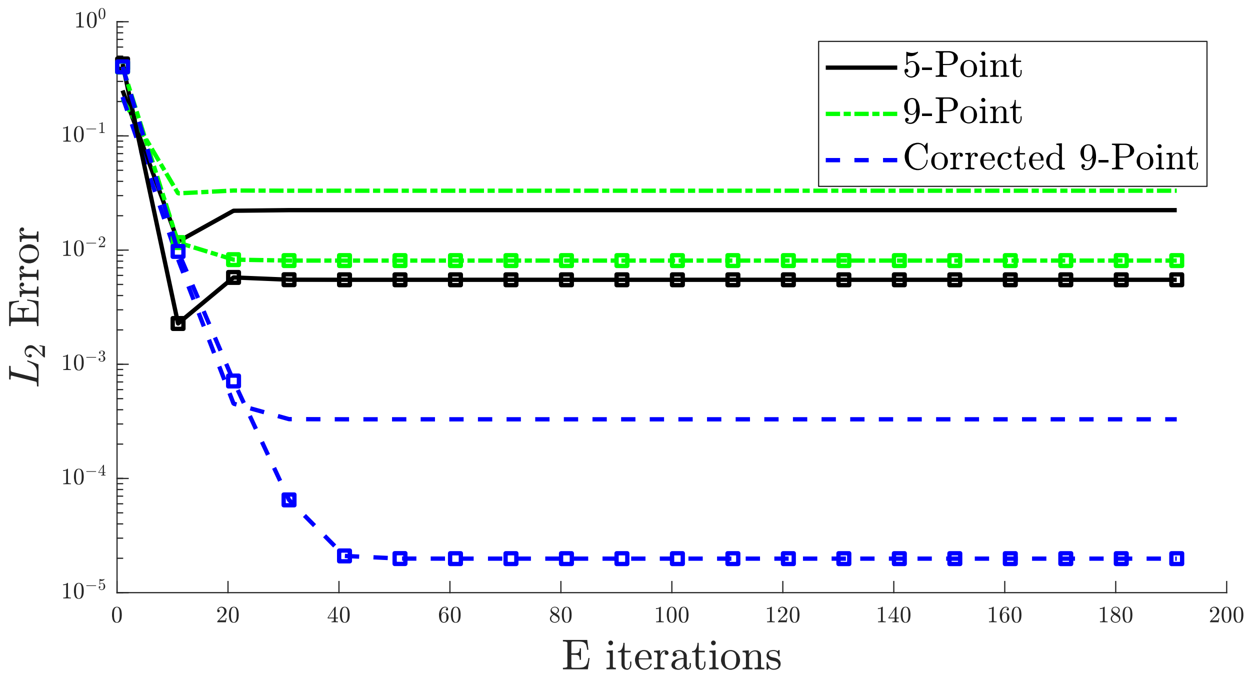

Figure 9: L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1.7$.

Note: □ are the markers for the $N = 32$ meshgrid.

In Figure 9, the approximate solutions converge to the analytical solutions with a decreased "over-shoot" of the converged answer when compared to the Jacobi iteration method. Also noteworthy, these schemes appear to converge to the asymptotic answers much faster than the Jacobi iteration method. Comparing Figure 5 to Figure 7 I found that for the $N = 16$ meshgrid for Gauss-Seidel converges to their asymptotic values approximately 10 iterations faster than Jacobi iteration method, when looking at the $N = 32$ meshgrid they converge 20 iterations faster.

Effects of Increasing ω

From analyzing Figures 6,7 and comparing to Figures 8,9 I can conclude that the over-relaxation factor ω determines the rate of convergence of a stencil. Looking at Figures 8,9 they converge significantly faster when $\omega = 1.7$ when compared to Figures 6,7 whose $\omega = 1$.

Effects of Increasing ω for Jacobi-Iteration

Jacobi-Iteration Residuals

Altering the over-relaxation values for the Jacboi-Iteration method shows that it does not have the same effect as it does for the Gauss-Seidel iteration method. It is shown above that increasing the over-relaxation value results in a faster convergence for Gauss-Seidel whereas the Figures 10, 11 disagree.

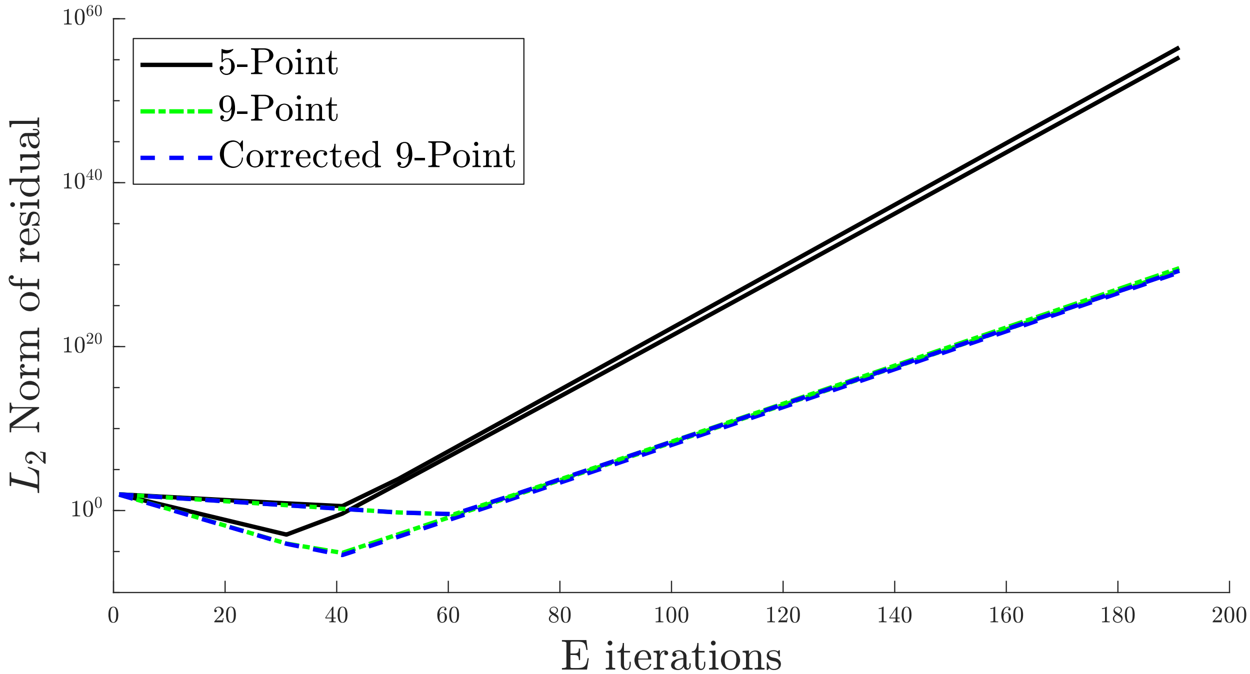

Figure 10: Residuals for Jacobi-Iteration with an over-relaxation value $\omega = 1.7$.

Above in Figure 10 is the effect of using $\omega = 1.7$, causing the Jacobi-Iteration method to grow unstable and overall result in an inaccurate approximation of the solution being solved for. It appears that the Jacobi-Iteration scheme may converge to the same solution when $\omega = 1$, but machine precisions may cause errors to propagate and ultimately cause the residuals to increase towards infinity and cause this scheme to be unstable for any over-relaxation value greater than one.

Jacobi-Iteration L_2 Error

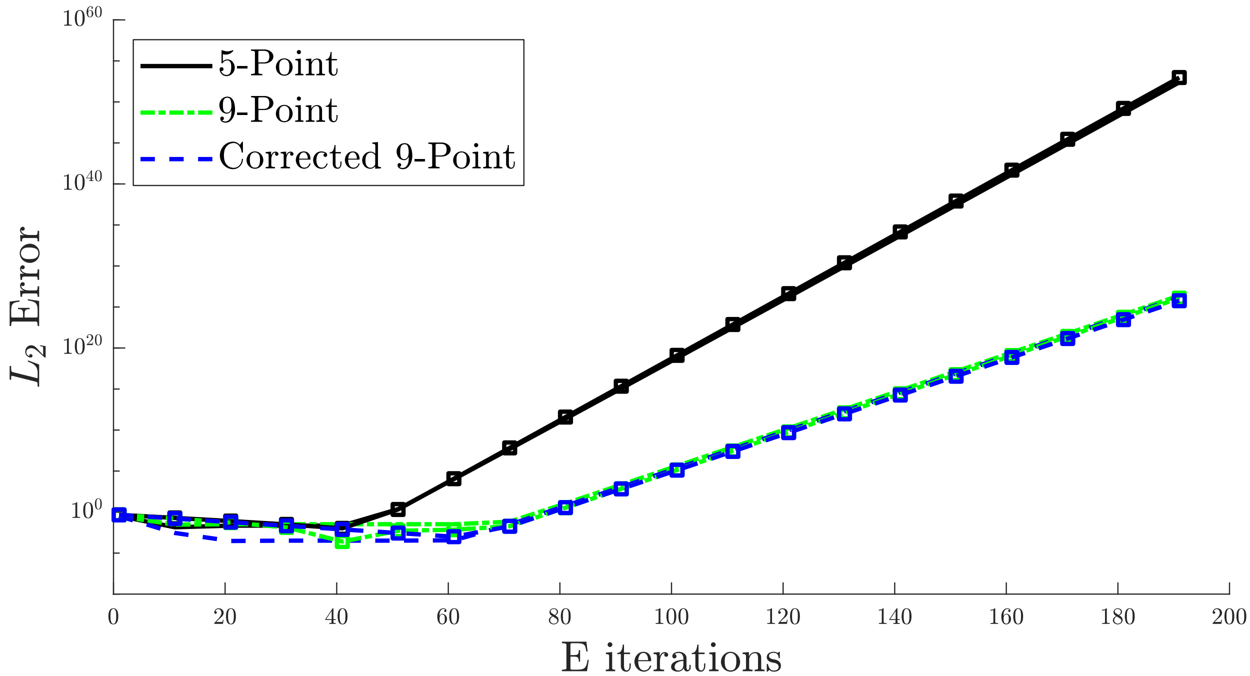

Figure 11: L_2 error for Jacobi-Iteration with an over-relaxation value $\omega = 1.7$.

Shown above in Figure 11, is the effect of running Jacobi-Iteration scheme with $\omega = 1.7$ resulting in an unstable iteration causing inaccuracies. Similar to the residuals the L_2 error may converge initially but machine precisions may cause the error to grow in size and ultimately diverge towards infinity and cause this scheme to be unstable for any over-relaxation value greater than one.

Appendices

A Driving Code for Possion's Equation

The highlighted code below is the main matlab code that would call all function and generate the plots needed in this project. The functions that are being referenced in this Appendix are shown in the following Appendices.

Algorithm 1: Main function matlab code for simulating Possion's Equation.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%       Dan Card   Aero 423 - Project 2
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/test.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
nsteps = 8.*[1,2,4,8];
h = 1./(nsteps);
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Direct Solver - [Task 2]
l2_5pt = zeros(1, length(nsteps));
l2_9pt = l2_5pt;
l2_c9pt = l2_5pt;

k = 1;
for num = nsteps

[xs,~, uf, uexact] = direct_solve(num, '5pt');
l2_5pt(k) = l2error(uf, uexact,xs);

[xs,~, uf, uexact] = direct_solve(num, '9pt');
l2_9pt(k) = l2error(uf, uexact,xs);

[xs,~, uf, uexact] = direct_solve(num, 'c9pt');
l2_c9pt(k) = l2error(uf, uexact,xs);

k = k + 1;
end
m_5pt = log10(l2_5pt(2)/l2_5pt(1))/log10(h(2)/h(1));
m_9pt = log10(l2_9pt(2)/l2_9pt(1))/log10(h(2)/h(1));
m_c9pt = log10(l2_c9pt(2)/l2_c9pt(1))/log10(h(2)/h(1));

figure()
hold on
plot(h, (l2_5pt),'k-o','linewidth', 1.8)
plot(h, (l2_9pt),'k--^','linewidth', 1.8)
plot(h, (l2_c9pt),'k-.s','linewidth', 1.8)
set(gca, 'yscale', 'log')
xlabel('h␣step,␣$\frac{L}{N}$','fontsize', 18)
ylabel('$L_{2}$␣Error','fontsize', 18)
legend({['$L_2$␣5-pt:␣', num2str(m_5pt)],['$L_2$␣9-pt:␣', num2str(m_9pt)],['$L_2$
    ␣Corrected␣9-pt:␣', num2str(m_c9pt)]},'location', 'southeast','fontsize', 18)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/stencils_l2.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Direct Solver - [Task 3]
% Pre-allocations and variables
nsteps = 8.*[2,4];
l2_5pt = zeros(2,200);
resid_5pt = l2_5pt;
l2_9pt = l2_5pt;
resid_9pt = l2_5pt;
l2_c9pt = l2_5pt;
```

```
resid_c9pt = l2_5pt;
k = 1;
for num = nsteps
    [l2, resid] = jacobi(num, '5pt', 1);
    l2_5pt(k,:) = l2;
    resid_5pt(k,:) = resid;

    [l2, resid] = jacobi(num, '9pt', 1);
    l2_9pt(k,:) = l2;
    resid_9pt(k,:) = resid;

    [l2, resid] = jacobi(num, 'c9pt', 1);
    l2_c9pt(k,:) = l2;
    resid_c9pt(k,:) = resid;

    k = k + 1;
end
steps = 1:10:200;

figure()
hold on
plot(steps, resid_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, resid_5pt(2,steps), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(2,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(2,steps), 'b--', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Norm␣of␣residual','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', 'southwest')
text(100, 10^-3, '$\uparrow$␣$N=32$␣Meshgrid', 'fontsize', 16)
text(10, 10^-6, '$N=16$␣Meshgrid␣$\rightarrow$', 'fontsize', 16)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/jacobi_residuals.eps')

figure()
hold on
plot(steps, l2_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Error','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', 'southwest')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/jacobi_l2.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Gauss-Seidel - [Task 4]

nsteps = 8.*[2,4];
l2_5pt = zeros(2,200);
resid_5pt = l2_5pt;
l2_9pt = l2_5pt;
resid_9pt = l2_5pt;
l2_c9pt = l2_5pt;
resid_c9pt = l2_5pt;
k = 1;
for num = nsteps
    [l2, resid] = gauss(num, '5pt', 1);
    l2_5pt(k,:) = l2;
    resid_5pt(k,:) = resid;

    [l2, resid] = gauss(num, '9pt', 1);
    l2_9pt(k,:) = l2;
    resid_9pt(k,:) = resid;
```

```matlab

    [l2, resid] = gauss(num, 'c9pt', 1);
    l2_c9pt(k,:) = l2;
    resid_c9pt(k,:) = resid;

    k = k + 1;
end
steps = 1:10:200;

figure()
hold on
plot(steps, resid_5pt(1,steps), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, resid_5pt(2,steps), 'k-s', 'linewidth', 1.8)
plot(steps, resid_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, resid_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E iterations','fontsize',18)
ylabel('$L_{2}$ Norm of residual','fontsize',18)
legend({'5-Point','9-Point','Corrected 9-Point'},'fontsize',16,'location', 'southwest')
text(80, 1, '$\leftarrow$ $N=32$ Meshgrid', 'fontsize', 16)
text(70, 10^-8, '$\leftarrow$ $N=16$ Meshgrid $\rightarrow$', 'fontsize', 16)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/gauss1_residuals.eps')

figure()
hold on
plot(steps, l2_5pt(1,steps), 'k-', 'linewidth', 1.8)
plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')

xlabel('E iterations','fontsize',18)
ylabel('$L_{2}$ Error','fontsize',18)
legend({'5-Point','9-Point','Corrected 9-Point'},'fontsize',16,'location', 'southwest')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/gauss1_l2.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
k = 1;
for num = nsteps
    [l2, resid] = gauss(num, '5pt', 1.7);
    l2_5pt(k,:) = l2;
    resid_5pt(k,:) = resid;

    [l2, resid] = gauss(num, '9pt', 1.7);
    l2_9pt(k,:) = l2;
    resid_9pt(k,:) = resid;

    [l2, resid] = gauss(num, 'c9pt', 1.7);
    l2_c9pt(k,:) = l2;
    resid_c9pt(k,:) = resid;

    k = k + 1;
end
steps = 1:10:200;

figure()
hold on
plot(steps, resid_5pt(1,steps), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, resid_5pt(2,steps), 'k-s', 'linewidth', 1.8)
plot(steps, resid_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, resid_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')
```

```
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Norm␣of␣residual','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', 'northeast')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/gauss17_residuals.eps')

figure()
hold on
plot(steps, l2_5pt(1,steps), 'k-', 'linewidth', 1.8)
plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Error','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', 'northeast')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/gauss17_l2.eps')

% Jacobi at omega = 1.7
nsteps = 8.*[2,4];
l2_5pt = zeros(2,200);
resid_5pt = l2_5pt;
l2_9pt = l2_5pt;
resid_9pt = l2_5pt;
l2_c9pt = l2_5pt;
resid_c9pt = l2_5pt;
k = 1;
for num = nsteps
    [l2, resid] = jacobi(num, '5pt', 1.7);
    l2_5pt(k,:) = l2;
    resid_5pt(k,:) = resid;

    [l2, resid] = jacobi(num, '9pt', 1.7);
    l2_9pt(k,:) = l2;
    resid_9pt(k,:) = resid;

    [l2, resid] = jacobi(num, 'c9pt', 1.7);
    l2_c9pt(k,:) = l2;
    resid_c9pt(k,:) = resid;

    k = k + 1;
end
steps = 1:10:200;

figure()
hold on
plot(steps, resid_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, resid_5pt(2,steps), 'k-', 'linewidth', 1.8)
plot(steps, resid_9pt(2,steps), 'g-.', 'linewidth', 1.8)
plot(steps, resid_c9pt(2,steps), 'b--', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Norm␣of␣residual','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', 'northwest')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/jacobi_residuals_17.eps')

figure()
hold on
plot(steps, l2_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
```

```
plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
set(gca, 'yscale','log')
xlabel('E␣iterations','fontsize',18)
ylabel('$L_{2}$␣Error','fontsize',18)
legend({'5-Point','9-Point','Corrected␣9-Point'},'fontsize',16,'location', '
    northwest')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig('figs/jacobi_l2_17.eps')

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

B Implementing Direct Solver

Implementing the direct solver, will be done through the upcoming stencil implementation functions.

B.1 Constructing 5-Point Stencil

Below is the Matlab interpretation of implementing the 5-point stencil.

Algorithm 2: Constructing the 5-point stencil A matrix.

```
function [A,F] = construct_5pt(num)
% Returns [A,F] for 5-Point stencil

    % Pre-allocate values
    A = sparse((num+1)^2, (num+1)^2);
    [edges, nodes] = detect_edges((num+1)^2);
    F = zeros((num+1)^2,1);
    h = 1/num;

    % Iterate through all the nodes
    for k = 1:length(edges)

        % If the node is a boundary element set the index to "1" and F to 0
        if edges(k,2) == 1
            A(k,k) = 1;
            F(k) = 0;
        end

        % If not a boundary element then conduct the 5-point stencil
        if edges(k,2) == 0
            % Get the x,y positions
            test = nodes(k,2:3);
            i = test(1); % x-position
            j = test(2); % y-position
            for count = 1:length(edges)
                if nodes(count,2) == i-1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 1/h^2 ;
                elseif nodes(count,2) == i+1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 1/h^2;
                elseif nodes(count,2) == i && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 1/h^2;
                elseif nodes(count,2) == i && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 1/h^2;
                end
            end
            A(k,k) = A(k,k) - 4/h^2;
            F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);

            % Machine precisioning
             if abs(F(k)) < 1.0e-16
                F(k) = 0;
```

```
            end
        end
    end
end
```

B.2 Constructing 9-Point Stencil

Below is the Matlab interpretation of implementing the 9-point stencil.

Algorithm 3: Constructing the 9-point stencil A matrix.

```
function [A,F] = construct_9pt(num)
% Returns [A,F] for 9-Point stencil

    % Pre-allocate values
    A = sparse((num+1)^2, (num+1)^2);
    [edges, nodes] = detect_edges((num+1)^2);
    F = zeros((num+1)^2,1);
    h = 1/num;

    % Iterate through all the nodes
    for k = 1:length(edges)

        % If the node is a boundary element set the index to "1" and F to 0
        if edges(k,2) == 1
            A(k,k) = 1;
            F(k) = 0;
        end

        % If not a boundary element then conduct the 9-point stencil
        if edges(k,2) == 0
            % Get the x,y positions
            test = nodes(k,2:3);
            i = test(1); % x-positions
            j = test(2); % y-positions
            for count = 1:length(edges)
                if nodes(count,2) == i-1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i-1 && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i-1 && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                end
            end
            A(k,k) = A(k,k) - 20/(6*h^2);
            F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);

            % Machine precisioning
            if abs(F(k)) < 1.0e-16
                F(k) = 0;
            end
        end
    end
end
```

B.3 Constructing Corrected 9-Point Stencil

Below is the Matlab interpretation of implementing the corrected 9-point stencil.

Algorithm 4: Constructing the corrected 9-point stencil A matrix.

```
function [A,F] = construct_c9pt(num)
% Returns [A,F] for corrected 9-Point stencil

    % Pre-allocate values
    A = sparse((num+1)^2, (num+1)^2);
    [edges, nodes] = detect_edges((num+1)^2);
    F = zeros((num+1)^2,1);
    h = 1/num;

    % Iterate through all the nodes
    for k = 1:length(edges)

        % If the node is a boundary element set the index to "1" and F to 0
        if edges(k,2) == 1
            A(k,k) = 1;
            F(k) = 0;
        end

        % If not a boundary element then conduct the corrected 9-point stencil
        if edges(k,2) == 0
            % Get the x,y positions
            test = nodes(k,2:3);
            i = test(1); % x-position
            j = test(2); % y-position
            for count = 1:length(edges)
                if nodes(count,2) == i-1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 4/(6*h^2);
                elseif nodes(count,2) == i-1 && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i-1 && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j-1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                elseif nodes(count,2) == i+1 && nodes(count,3) == j+1
                    A(k,count) = A(k,count) + 1/(6*h^2);
                end
            end
            A(k,k) = A(k,k) - 20/(6*h^2);

            F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h)+...
            1/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h)+...
            1/12*(-20)*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h)-...
            4/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h)+...
            1/12*(-20)*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h)+...
            1/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h);

            % Machine precisioning
             if abs(F(k)) < 1.0e-16
                F(k) = 0;
            end
        end
    end
end
```

C Implementing Jacboi Method

Implementing the jacobi iteration method, will be done through the upcoming stencil implementation functions.

Algorithm 5: Matlab implementation of Jacobi method.

```
function [l2err, resid] = jacobi(num, stencil, omega)
% Returns L2 error and residuals

    % Pre-allocations
    U = zeros(num+1);
    resid_mat = U;
    resid = zeros(max(size(1:200)),1);
    l2err = resid;
    xs = linspace(0,1, num+1);
    [xs,ys] = meshgrid(xs);
    uexact = exact_sol(xs, ys);
    h = 1/num;

    % Exectue Jacobi Iteration
    for iter = 1:200
        utemp = zeros(num+1);

        % Conduct the 5-point stencil
        if strcmp(stencil, '5pt')
            s0 = (-4/h^2);
            for i = 2:num
                for j = 2:num
                    s_5pt = 0;
                    s_5pt = s_5pt - 4*U(j,i)/h^2;
                    s_5pt = s_5pt + U(j-1,i)/h^2;
                    s_5pt = s_5pt + U(j+1,i)/h^2;
                    s_5pt = s_5pt + U(j,i-1)/h^2;
                    s_5pt = s_5pt + U(j,i+1)/h^2;

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_5pt;
                    resid_mat(j,i) = rij;
                    utemp(j,i) = U(j,i) + omega/s0.*rij;
                end
            end
        % Conduct the 9-point stencil
        elseif strcmp(stencil, '9pt')
            s0 = (-20/(6*h^2));
            for i = 2:num
                for j = 2:num
                    s_9pt = 0;
                    s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
                    s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
                    s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
                    s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
                    s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_9pt;
                    resid_mat(j,i) = rij;
                    utemp(j,i) = U(j,i) + omega/s0.*rij;
                end
            end
        % Conduct the 9-point stencil
        elseif strcmp(stencil, 'c9pt')
            s0 = (-20/(6*h^2));
            for i = 2:num
                for j = 2:num
                    s_c9pt = 0;
```

```matlab
                    s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
                        (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
                    rij = RHS - s_c9pt;
                    resid_mat(j,i) = rij;
                    utemp(j,i) = U(j,i) + omega/s0.*rij;
                end
            end
        end
        % Compute the residuals
        resid(iter) = sqrt(sum(sum((resid_mat.^2)))/(num+1)^2);

        U = utemp;
        % Compute the L2 error
        l2err(iter) = sqrt(sum(sum((U-uexact).^2))*h^2);
    end
end
```

D Implementing Gauss-Seidel Method

Implementing the Gauss-Seidel iteration method, will be done through the upcoming stencil implementation functions.

Algorithm 6: Matlab implementation of Gauss-Seidel method.

```matlab
function [l2err, resid] = gauss(num, stencil, omega)
% Returns L2 error and residual

    % Pre-allocations
    h = 1/num;
    U = zeros(num+1);
    resid_mat = U;
    resid = zeros(max(size(1:200)),1);
    l2err = resid;
    xs = linspace(0,1, num+1);
    [xs,ys] = meshgrid(xs);
    uexact = exact_sol(xs, ys);

    % Exectue Jacobi Iteration
    for iter = 1:200

        % Conduct 5-pt stencil
        if strcmp(stencil, '5pt')
            s0 = (-4/h^2);

            % Update the red nodes
            for j = 2:num
                if mod(j,2) == 0
                    for i = 2:2:num
                        s_5pt = 0;
                        s_5pt = s_5pt - 4*U(j,i)/h^2;
                        s_5pt = s_5pt + U(j-1,i)/h^2;
                        s_5pt = s_5pt + U(j+1,i)/h^2;
```

```
                    s_5pt = s_5pt + U(j,i-1)/h^2;
                    s_5pt = s_5pt + U(j,i+1)/h^2;

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_5pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                end
            else
                for i = 3:2:num-1
                    s_5pt = 0;
                    s_5pt = s_5pt - 4*U(j,i)/h^2;
                    s_5pt = s_5pt + U(j-1,i)/h^2;
                    s_5pt = s_5pt + U(j+1,i)/h^2;
                    s_5pt = s_5pt + U(j,i-1)/h^2;
                    s_5pt = s_5pt + U(j,i+1)/h^2;

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_5pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                end
            end
        end

        % Update the black Nodes
        for j = 2:num
            if mod(j,2) == 0
                for i = 3:2:num-1
                    s_5pt = 0;
                    s_5pt = s_5pt - 4*U(j,i)/h^2;
                    s_5pt = s_5pt + U(j-1,i)/h^2;
                    s_5pt = s_5pt + U(j+1,i)/h^2;
                    s_5pt = s_5pt + U(j,i-1)/h^2;
                    s_5pt = s_5pt + U(j,i+1)/h^2;

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_5pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                end
            else
                for i = 2:2:num
                    s_5pt = 0;
                    s_5pt = s_5pt - 4*U(j,i)/h^2;
                    s_5pt = s_5pt + U(j-1,i)/h^2;
                    s_5pt = s_5pt + U(j+1,i)/h^2;
                    s_5pt = s_5pt + U(j,i-1)/h^2;
                    s_5pt = s_5pt + U(j,i+1)/h^2;

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                    rij = RHS - s_5pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                end
            end
        end

    % Conduct 9-pt stencil
    elseif strcmp(stencil, '9pt')
        s0 = (-20/(6*h^2));

        % Update the red nodes
        for j = 2:num
            if mod(j,2) == 0
                for i = 2:2:num
                    s_9pt = 0;
                    s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
                    s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
```

```
                        s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);

                        RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                        rij = RHS - s_9pt;
                        resid_mat(j,i) = rij;
                        U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                else
                    for i = 3:2:num-1
                        s_9pt = 0;
                        s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);

                        RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                        rij = RHS - s_9pt;
                        resid_mat(j,i) = rij;
                        U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                end
            end

            % Update the black Nodes
            for j = 2:num
                if mod(j,2) == 0
                    for i = 3:2:num-1
                        s_9pt = 0;
                        s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);

                        RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                        rij = RHS - s_9pt;
                        resid_mat(j,i) = rij;
                        U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                else
                    for i = 2:2:num
                        s_9pt = 0;
                        s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
                        s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
                        s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);

                        RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
                        rij = RHS - s_9pt;
                        resid_mat(j,i) = rij;
                        U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                end
```

```
            end

        % Conduct corrected 9-pt stencil
        elseif strcmp(stencil, 'c9pt')
            s0 = (-20/(6*h^2));

            % Update the red nodes
            for j = 2:num
                if mod(j,2) == 0
                    for i = 2:2:num
                    s_c9pt = 0;
                    s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
                        (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
                    rij = RHS - s_c9pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                else
                    for i = 3:2:num-1
                    s_c9pt = 0;
                    s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
                        (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
                    rij = RHS - s_c9pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                end
            end

            % Update the black Nodes
            for j = 2:num
                if mod(j,2) == 0
                    for i = 3:2:num-1
                    s_c9pt = 0;
                    s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);
```

```

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
                        (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
                    rij = RHS - s_c9pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                else
                    for i = 2:2:num
                    s_c9pt = 0;
                    s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
                    s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);

                    RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
                        (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
                        1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
                    rij = RHS - s_c9pt;
                    resid_mat(j,i) = rij;
                    U(j,i) = U(j,i) + omega/s0.*rij;
                    end
                end
            end

        end
        % Compute the resdiduals from the residual matrix
        resid(iter) = sqrt(sum(sum((resid_mat.^2)))/(num+1)^2);

        % Compute the L2 error
        l2err(iter) = sqrt(sum(sum((U-uexact).^2))*h^2);
    end
end
```

E Miscellaneous Functions

This functions are used for ease of functionality to reduce user error.

E.1 Exact Solution Function

This function inputs the x and y values (in a meshedgrid), and returns the analytical solution.

Algorithm 7: Function to compute the exact solution.

```
function an = exact_sol(x, y)
% Returns the analytical soltuion
    an = sin(2*pi.*x).*sin(4*pi.*y);
end
```

E.2 L_2 Error Function

This function will compute the L_2 error given the approximated value and the analytical.

Algorithm 8: Matlab L_2 error function implementation.

```
function err = l2error(uf, uexact,xs)
    h = xs(1,2)-xs(1,1);

    err = 0;
    for i = 1:length(uf)
        for j = 1:length(uf)
            err = err + (uf(i,j) - uexact(i,j))^2;
        end
    end
    err = sqrt(h^2*err);
end
```

E.3 Function to Determine the Boundary Elements

This function inputs the size of the meshedgrid and solves for the boundary nodes.

Algorithm 9: Matlab function to solve for boundary nodes.

```
function [edges, nodes] = detect_edges(num)
% Returns the edges and nodes for each NxN meshgrid

    % Pre-allocate the nodes
    nodes = zeros(num,3);
    edges = zeros(num,2);
    k = 1; j = 1;
    for i = 1:num
        % Create a Nx3 matrix of the nodes with their locations
        nodes(i,:) = [i, k, j];
        k = k + 1;
        if k == sqrt(num)+1
           j = j + 1;
           k = 1;
        end
    end

    % Two logicals to determine the indices of the boundary nodes
    sides = nodes < 1.01;
    sides2 = nodes > (sqrt(num)-0.01);
    edgevals = sides + sides2;

    edges(:,1) = 1:num;
    for i = 1:num
        % Checks to see if either the x,y are a bounadry nodes
        if edgevals(i,2) == 1 || edgevals(i,3) == 1
            edges(i,2) = 1;
        end
    end

end
```