

Project 2: Poisson'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

1	Introduction	3
1.1	Overview	3
1.2	Stencils	4
1.3	Solvers	4
1.4	Error Measure	5
2	Questions and Tasks	6
2.1	Analytical Order of Accuracy Studies	6
2.1.1	5-Point Stencil Order of Accuracy	7
2.1.2	9-Point Stencil Order of Accuracy	8
2.1.3	Corrected 9-Point Stencil Order of Accuracy	9
2.2	Implementing Stencils	11
2.2.1	Solving the Exact Solution	11
2.2.2	Convergence Study and Implementing Stencils	12
2.3	Implementing Jacobi Iterative Solver	13
2.3.1	Jacobi Iteration Residuals	13
2.3.2	Jacobi Iteration Errors	14
2.3.3	Meshgrid Size and Convergence	14
2.4	Gauss-Seidel Iteration	15
2.4.1	Gauss-Seidel With Over-Relaxation Value = 1	15
2.4.2	Gauss-Seidel With Over-Relaxation Value = 1.7	17
	Appendices	20
	Appendix A Driving Code for Poisson's Equation	21
	Appendix B Implementing Direct Solver	25
	B.1 Constructing 5-Point Stencil	25
	B.2 Constructing 9-Point Stencil	26
	B.3 Constructing Corrected 9-Point Stencil	27
	Appendix C Implementing Jacobi Method	28
	Appendix D Implementing Gauss-Seidel Method	29

Appendix E Miscellaneous Functions	33
E.1 Exact Solution Function	33
E.2 L_2 Error Function	34
E.3 Function to Determine the Boundary Elements	34

List of Figures

1 Poisson's equation in two dimensions.	3
2 Red and black node designations for an $N = 4$ lattice.	5
3 Implementing stencils for convergence study.	12
4 Residuals for Jacobi iteration.	13
5 L_2 error for Jacobi iteration.	14
6 Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1$	15
7 L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1$	16
8 Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1.7$	17
9 L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1.7$	18
10 Residuals for Jacobi-Iteration with an over-relaxation value $\omega = 1.7$	19
11 L_2 error for Jacobi-Iteration with an over-relaxation value $\omega = 1.7$	20

List of Algorithms

1 Main function matlab code for simulating Possion's Equation.	21
2 Constructing the 5-point stencil A matrix.	25
3 Constructing the 9-point stencil A matrix.	26
4 Constructing the corrected 9-point stencil A matrix.	27
5 Matlab implementation of Jacobi method.	28
6 Matlab implementation of Gauss-Seidel method.	29
7 Function to compute the exact solution.	33
8 Matlab L_2 error function implementation.	34
9 Matlab function to solve for boundary nodes.	34

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

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

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.

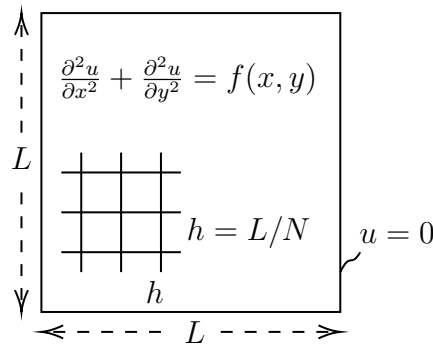


Figure 1: Poisson's equation in two dimensions.

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,

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

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:

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{\omega}{S_{0,0}} r_{i,j}^n, \quad \text{where} \quad r_{i,j}^n = \tilde{f}_{i,j} - \sum_{-1 \leq l, m \leq 1} S_{l,m} u_{i+1,j+m}^n.$$

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. $r_{i,j}^n$ 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.

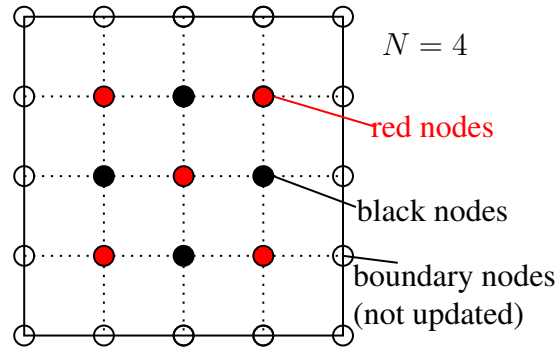


Figure 2: Red and black node designations for an $N = 4$ lattice.

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

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

where $u_{i,j}^{\text{exact}}$ 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:

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

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,

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

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:

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

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,

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

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,

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

Then the expression becomes

$$u_{xx} + u_{yy} + \frac{h^2}{12}(u_{xxxx} + 2u_{xxyy} + u_{yyyy}) = f_{i,j} + \left[+\frac{1}{144} (h^4u_{xxyyyy} + h^4u_{yyyyyy}) + \frac{1}{144} (12h^2u_{xxxx} + 24h^2u_{xxyy} + 12h^2u_{yyyy}) \right]$$

$$f_{i,j} = u_{xx} + u_{yy} - \frac{h^4}{144}(u_{xxxxx} + u_{xxxxy} + u_{xxyyy} + u_{yyyyy})$$

$$f_{i,j} \approx u_{xx} + u_{yy} + \mathcal{O}(h^4)$$

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

Poisson'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.

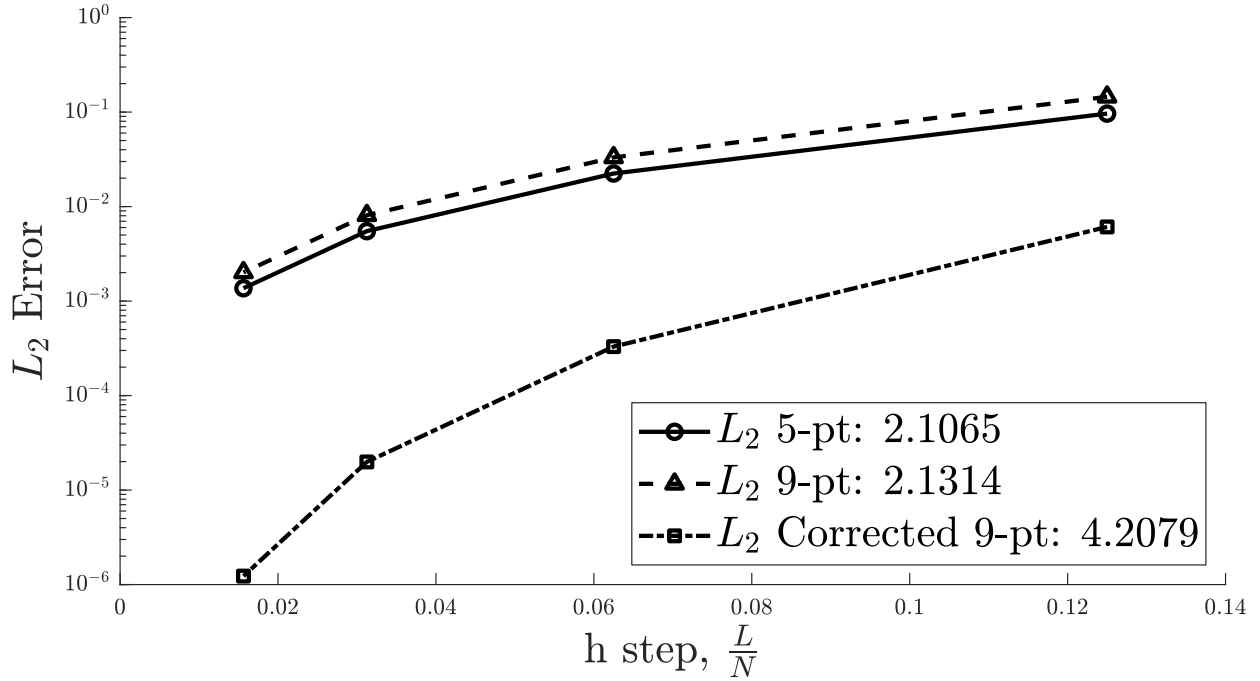


Figure 3: Implementing stencils for convergence study.

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.

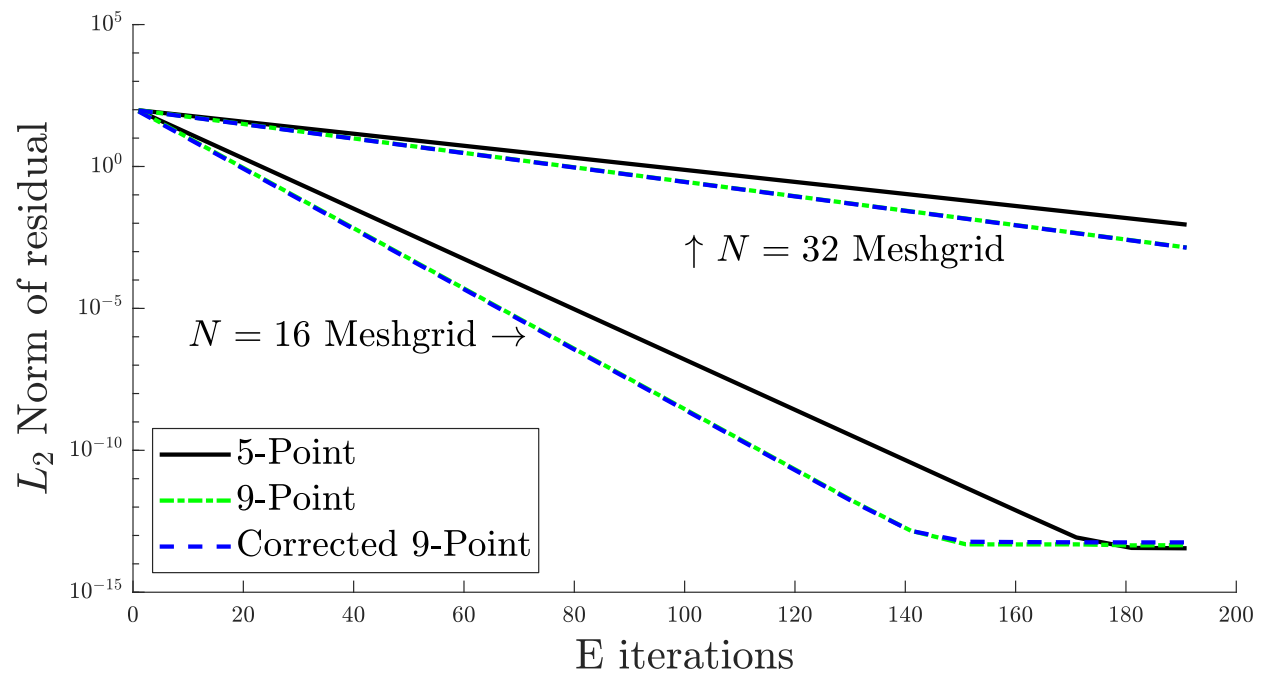


Figure 4: Residuals for Jacobi iteration.

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 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.

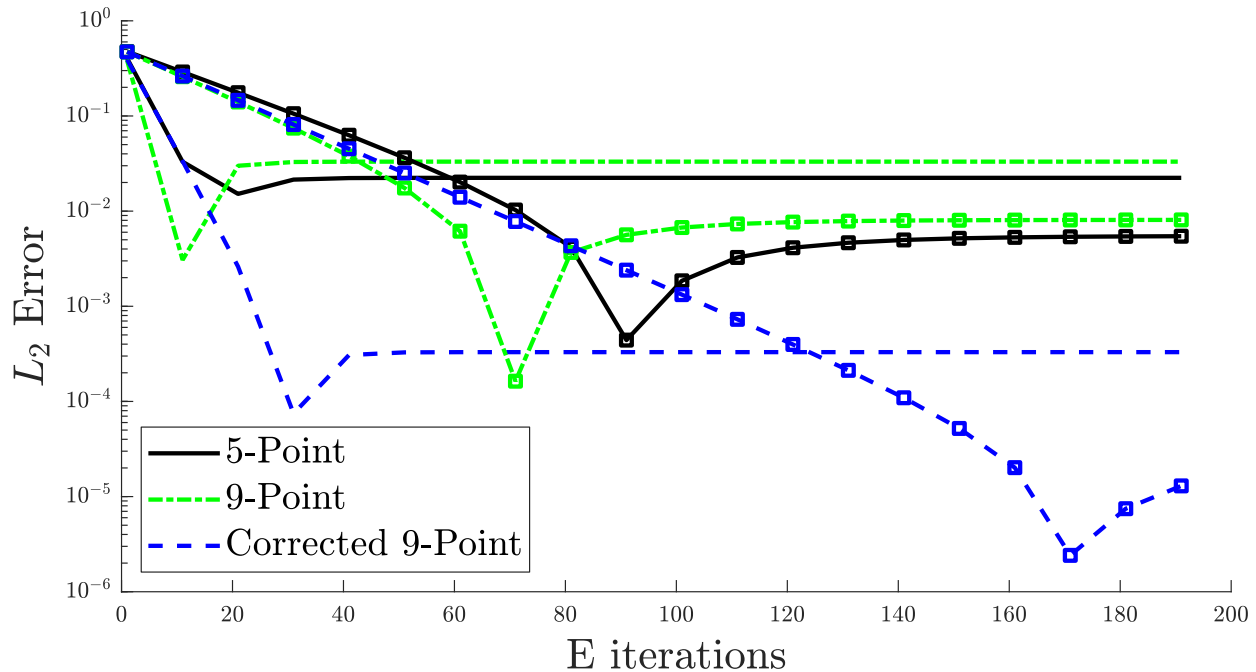


Figure 5: L_2 error for Jacobi iteration.

Note: \square 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 rates 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.

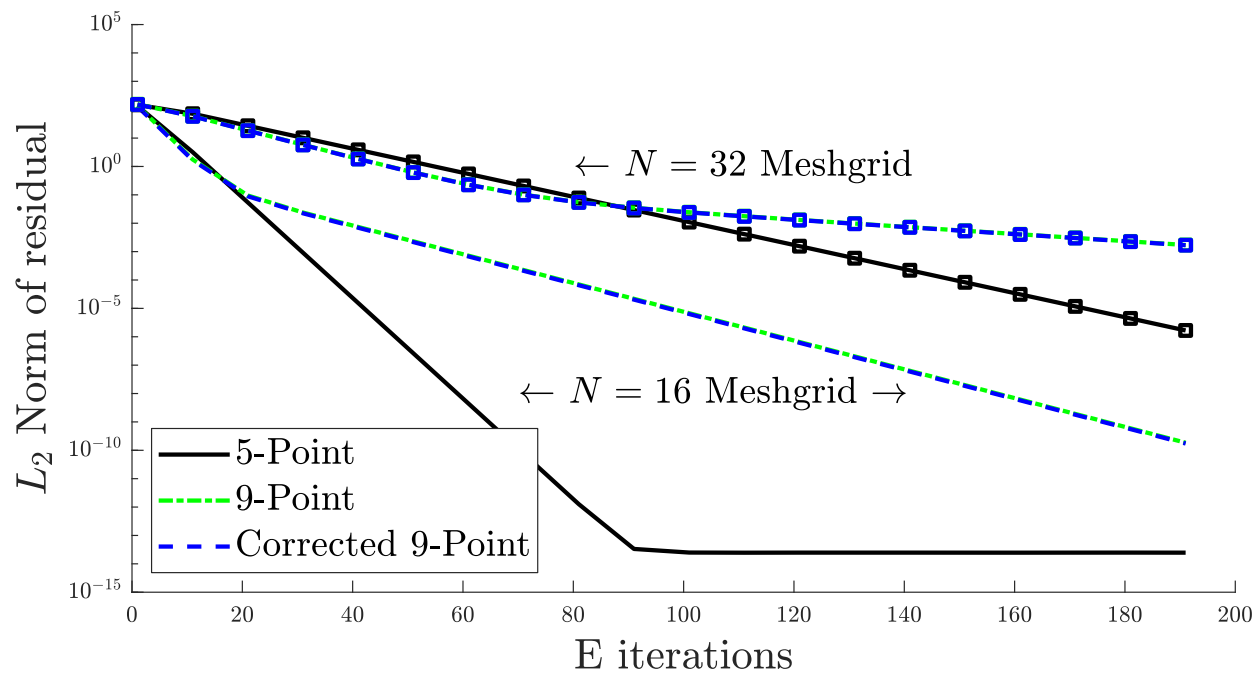


Figure 6: Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1$.

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.

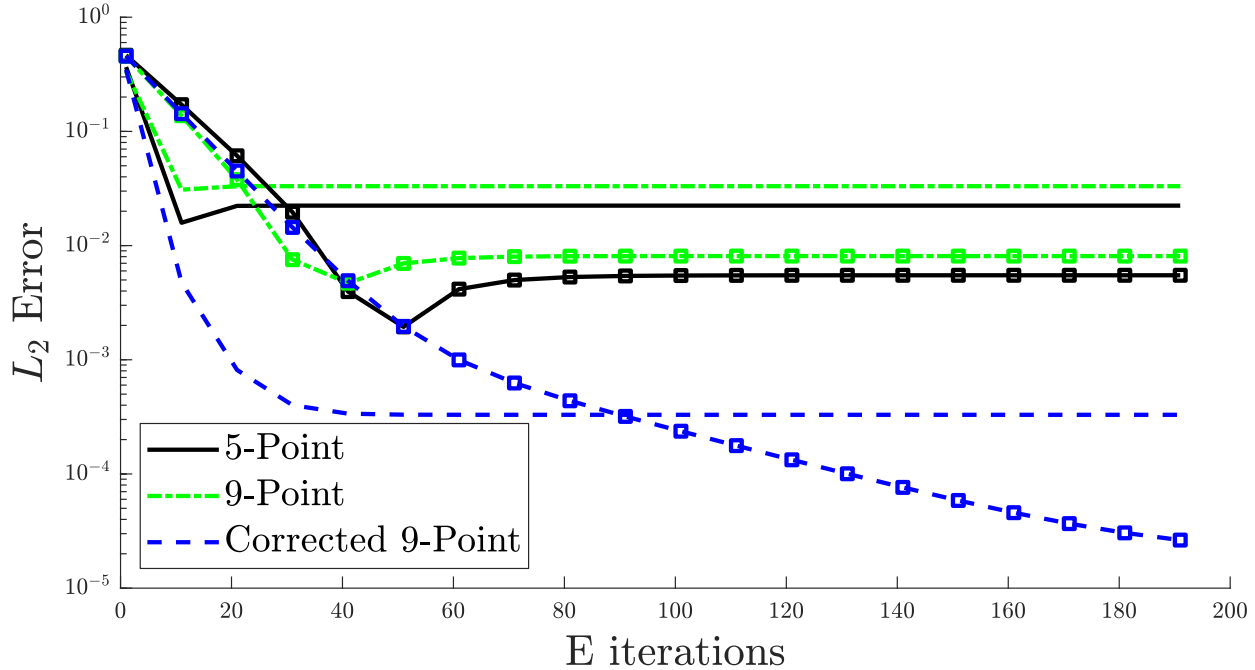


Figure 7: L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1$.

Note: \square 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.

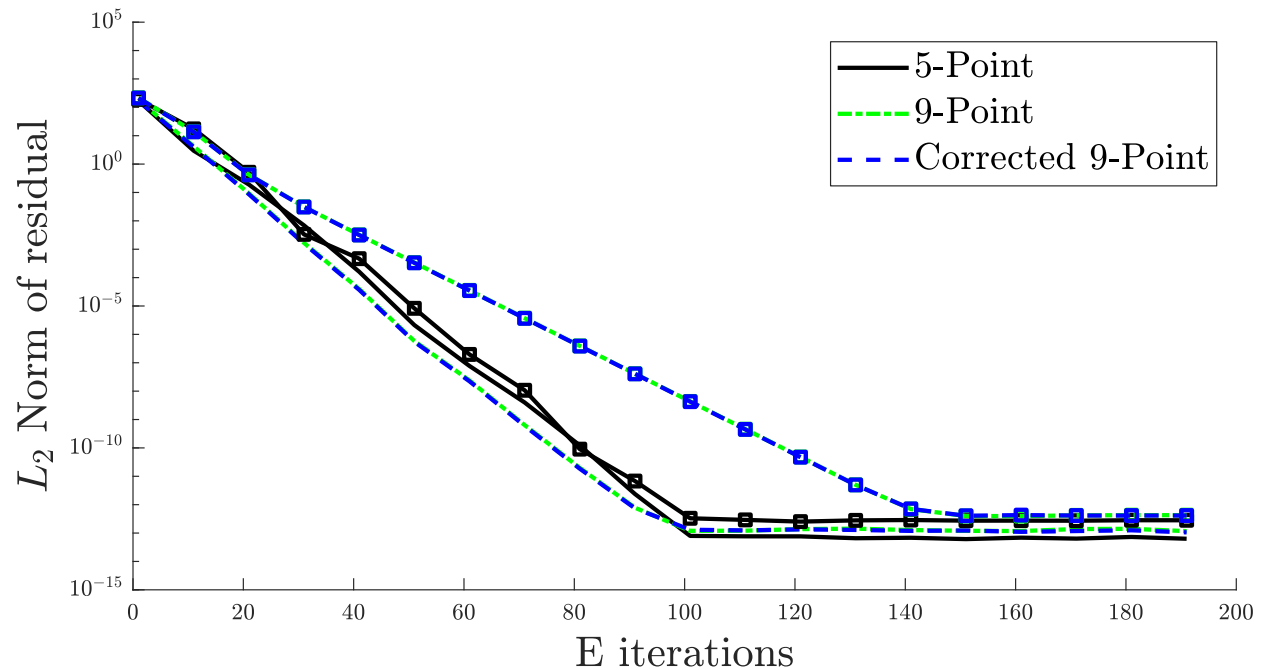


Figure 8: Residuals for Gauss-Seidel with an over-relaxation value $\omega = 1.7$.

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.

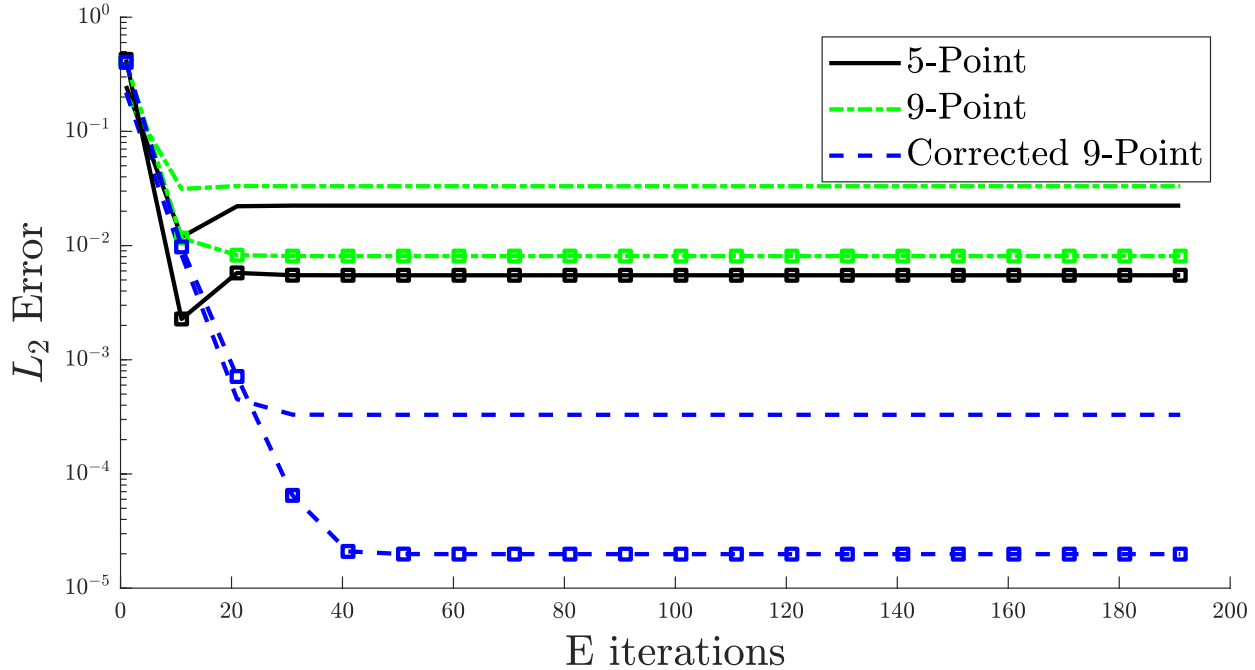


Figure 9: L_2 error for Gauss-Seidel with an over-relaxation value $\omega = 1.7$.

Note: \square 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 Jacobi-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.

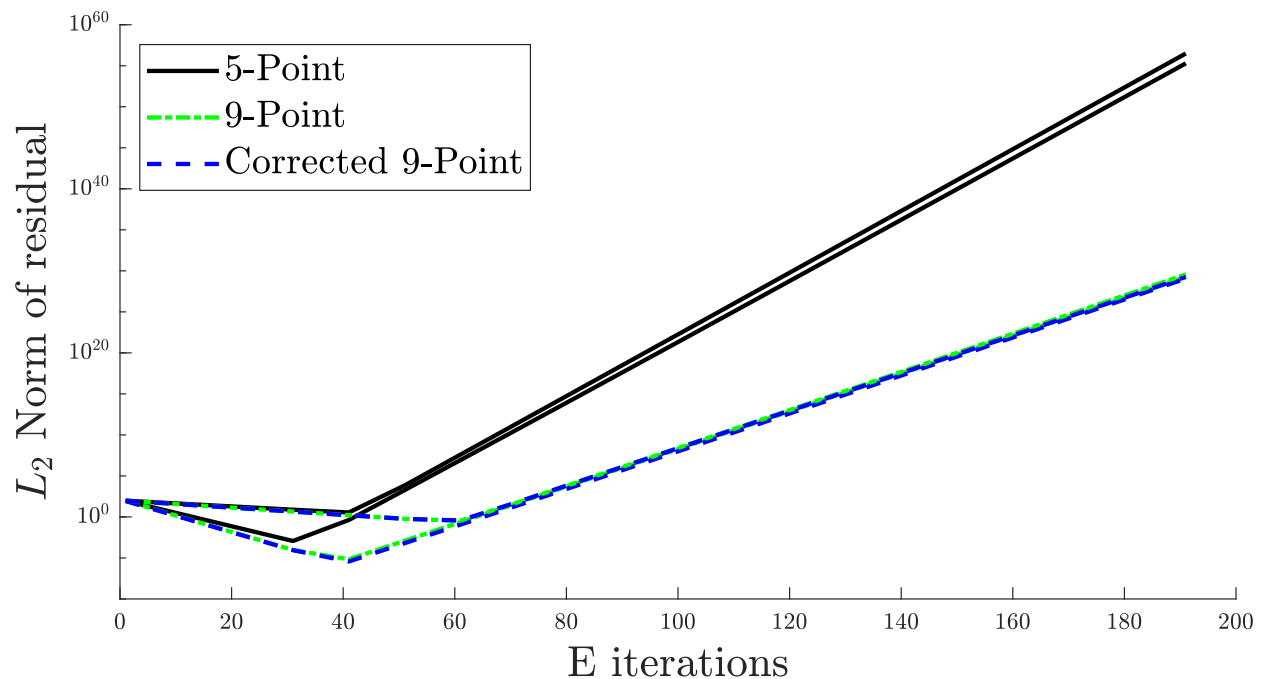


Figure 10: Residuals for Jacobi-Iteration with an over-relaxation value $\omega = 1.7$.

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.

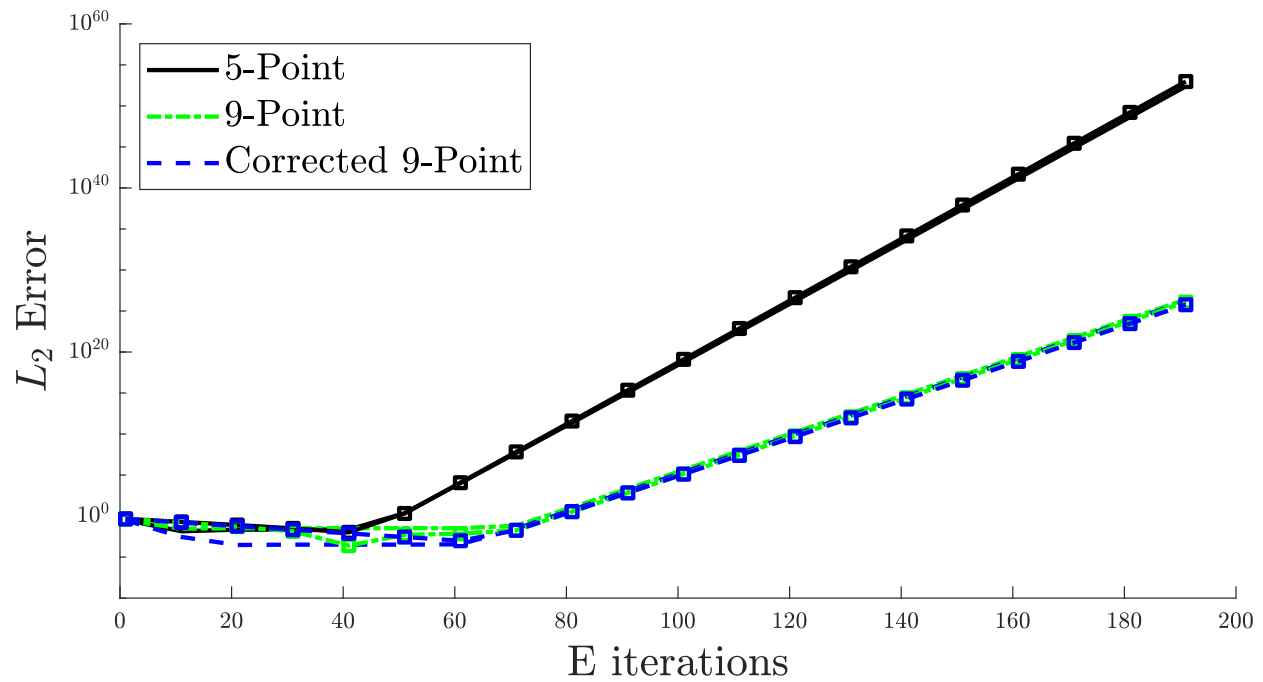
Jacobi-Iteration L_2 Error

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.

```

1  %~~~~~
2  %      Dan Card  Aero 423 - Project 2
3  %~~~~~
4  clear all; clc; close all
5  set(groot,'defaulttextinterpreter','latex');
6  set(groot, 'defaultAxesTickLabelInterpreter','latex');
7  set(groot, 'defaultLegendInterpreter','latex');
8  %set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
9  %export_fig('figs/test.eps')
10 %~~~~~
11 nsteps = 8.*[1,2,4,8];
12 h = 1./(nsteps);
13 %~~~~~
14 % Direct Solver - [Task 2]
15 l2_5pt = zeros(1, length(nsteps));
16 l2_9pt = l2_5pt;
17 l2_c9pt = l2_5pt;
18
19 k = 1;
20 for num = nsteps
21
22 [xs,~, uf, uexact] = direct_solve(num, '5pt');
23 l2_5pt(k) = l2error(uf, uexact,xs);
24
25 [xs,~, uf, uexact] = direct_solve(num, '9pt');
26 l2_9pt(k) = l2error(uf, uexact,xs);
27
28 [xs,~, uf, uexact] = direct_solve(num, 'c9pt');
29 l2_c9pt(k) = l2error(uf, uexact,xs);
30
31 k = k + 1;
32 end
33 m_5pt = log10(l2_5pt(2)/l2_5pt(1))/log10(h(2)/h(1));
34 m_9pt = log10(l2_9pt(2)/l2_9pt(1))/log10(h(2)/h(1));
35 m_c9pt = log10(l2_c9pt(2)/l2_c9pt(1))/log10(h(2)/h(1));
36
37 figure()
38 hold on
39 plot(h, (l2_5pt),'k-o','linewidth', 1.8)
40 plot(h, (l2_9pt),'k--^','linewidth', 1.8)
41 plot(h, (l2_c9pt),'k-.s','linewidth', 1.8)
42 set(gca, 'yscale', 'log')
43 xlabel('h\step, \frac{L}{N}$', 'fontsize', 18)
44 ylabel('$L_{2}$\Error', 'fontsize', 18)
45 legend({'$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)
46 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
47 export_fig('figs/stencils_l2.eps')
48 %~~~~~
49 % Direct Solver - [Task 3]
50 % Pre-allocations and variables
51 nsteps = 8.*[2,4];
52 l2_5pt = zeros(2,200);
53 resid_5pt = l2_5pt;
54 l2_9pt = l2_5pt;
55 resid_9pt = l2_5pt;
56 l2_c9pt = l2_5pt;

```

```

57 resid_c9pt = 12_5pt;
58 k = 1;
59 for num = nsteps
60     [l2, resid] = jacobi(num, '5pt', 1);
61     l2_5pt(k,:) = l2;
62     resid_5pt(k,:) = resid;
63
64     [l2, resid] = jacobi(num, '9pt', 1);
65     l2_9pt(k,:) = l2;
66     resid_9pt(k,:) = resid;
67
68     [l2, resid] = jacobi(num, 'c9pt', 1);
69     l2_c9pt(k,:) = l2;
70     resid_c9pt(k,:) = resid;
71
72     k = k + 1;
73 end
74 steps = 1:10:200;
75
76 figure()
77 hold on
78 plot(steps, resid_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
79 plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
80 plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
81 plot(steps, resid_5pt(2,steps), 'k-', 'linewidth', 1.8)
82 plot(steps, resid_9pt(2,steps), 'g-.', 'linewidth', 1.8)
83 plot(steps, resid_c9pt(2,steps), 'b--', 'linewidth', 1.8)
84 set(gca, 'yscale', 'log')
85 xlabel('E_iterations', 'fontsize', 18)
86 ylabel('$L_{\{2\}}$Norm_of_residual', 'fontsize', 18)
87 legend({'5-Point', '9-Point', 'Corrected_9-Point'}, 'fontsize', 16, 'location', 'southwest')
88 text(100, 10^-3, '$\uparrow$N=32 Meshgrid', 'fontsize', 16)
89 text(10, 10^-6, '$N=16$ Meshgrid $\rightarrow$', 'fontsize', 16)
90 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
91 export_fig('figs/jacobi_residuals.eps')
92
93 figure()
94 hold on
95 plot(steps, l2_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
96 plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
97 plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
98 plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
99 plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
100 plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
101 set(gca, 'yscale', 'log')
102 xlabel('E_iterations', 'fontsize', 18)
103 ylabel('$L_{\{2\}}$Error', 'fontsize', 18)
104 legend({'5-Point', '9-Point', 'Corrected_9-Point'}, 'fontsize', 16, 'location', 'southwest')
105 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
106 export_fig('figs/jacobi_l2.eps')
107 %~~~~~
108 % Gauss-Seidel - [Task 4]
109
110 nsteps = 8.*[2,4];
111 l2_5pt = zeros(2,200);
112 resid_5pt = l2_5pt;
113 l2_9pt = l2_5pt;
114 resid_9pt = l2_5pt;
115 l2_c9pt = l2_5pt;
116 resid_c9pt = l2_5pt;
117 k = 1;
118 for num = nsteps
119     [l2, resid] = gauss(num, '5pt', 1);
120     l2_5pt(k,:) = l2;
121     resid_5pt(k,:) = resid;
122
123     [l2, resid] = gauss(num, '9pt', 1);
124     l2_9pt(k,:) = l2;
125     resid_9pt(k,:) = resid;

```

```

126
127     [l2, resid] = gauss(num, 'c9pt', 1);
128     l2_c9pt(k,:) = l2;
129     resid_c9pt(k,:) = resid;
130
131     k = k + 1;
132 end
133 steps = 1:10:200;
134
135 figure()
136 hold on
137 plot(steps, resid_5pt(1,steps), 'k-', 'linewidth', 1.8)
138 plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
139 plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
140 plot(steps, resid_5pt(2,steps), 'k-s', 'linewidth', 1.8)
141 plot(steps, resid_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
142 plot(steps, resid_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
143 set(gca, 'yscale', 'log')
144 xlabel('E_iterations', 'fontsize', 18)
145 ylabel('$L_{2}$Norm of residual', 'fontsize', 18)
146 legend({'5-Point', '9-Point', 'Corrected_9-Point'}, 'fontsize', 16, 'location', 'southwest')
147 text(80, 1, '$\leftarrow N=32$ Meshgrid', 'fontsize', 16)
148 text(70, 10^-8, '$\leftarrow N=16$ Meshgrid $\rightarrow$', 'fontsize', 16)
149 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
150 export_fig('figs/gauss1_residuals.eps')
151
152 figure()
153 hold on
154 plot(steps, l2_5pt(1,steps), 'k-', 'linewidth', 1.8)
155 plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
156 plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
157 plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
158 plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
159 plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
160 set(gca, 'yscale', 'log')
161
162 xlabel('E_iterations', 'fontsize', 18)
163 ylabel('$L_{2}$Error', 'fontsize', 18)
164 legend({'5-Point', '9-Point', 'Corrected_9-Point'}, 'fontsize', 16, 'location', 'southwest')
165 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
166 export_fig('figs/gauss1_l2.eps')
167 %-----
168 k = 1;
169 for num = nsteps
170     [l2, resid] = gauss(num, '5pt', 1.7);
171     l2_5pt(k,:) = l2;
172     resid_5pt(k,:) = resid;
173
174     [l2, resid] = gauss(num, '9pt', 1.7);
175     l2_9pt(k,:) = l2;
176     resid_9pt(k,:) = resid;
177
178     [l2, resid] = gauss(num, 'c9pt', 1.7);
179     l2_c9pt(k,:) = l2;
180     resid_c9pt(k,:) = resid;
181
182     k = k + 1;
183 end
184 steps = 1:10:200;
185
186 figure()
187 hold on
188 plot(steps, resid_5pt(1,steps), 'k-', 'linewidth', 1.8)
189 plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
190 plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
191 plot(steps, resid_5pt(2,steps), 'k-s', 'linewidth', 1.8)
192 plot(steps, resid_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
193 plot(steps, resid_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
194 set(gca, 'yscale', 'log')

```

```

195 xlabel('E_iterations','fontsize',18)
196 ylabel('$L_{2}$Norm_of_residual','fontsize',18)
197 legend({'5-Point','9-Point','Corrected_9-Point'},'fontsize',16,'location', '
    northeast')
198 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
199 export_fig('figs/gauss17_residuals.eps')
200
201 figure()
202 hold on
203 plot(steps, l2_5pt(1,steps), 'k-', 'linewidth', 1.8)
204 plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
205 plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
206 plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)
207 plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
208 plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
209 set(gca, 'yscale','log')
210 xlabel('E_iterations','fontsize',18)
211 ylabel('$L_{2}$Error','fontsize',18)
212 legend({'5-Point','9-Point','Corrected_9-Point'},'fontsize',16,'location', '
    northeast')
213 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
214 export_fig('figs/gauss17_l2.eps')
215
216 % Jacobi at omega = 1.7
217 nsteps = 8.*[2,4];
218 l2_5pt = zeros(2,200);
219 resid_5pt = l2_5pt;
220 l2_9pt = l2_5pt;
221 resid_9pt = l2_5pt;
222 l2_c9pt = l2_5pt;
223 resid_c9pt = l2_5pt;
224 k = 1;
225 for num = nsteps
226     [l2, resid] = jacobi(num, '5pt', 1.7);
227     l2_5pt(k,:) = l2;
228     resid_5pt(k,:) = resid;
229
230     [l2, resid] = jacobi(num, '9pt', 1.7);
231     l2_9pt(k,:) = l2;
232     resid_9pt(k,:) = resid;
233
234     [l2, resid] = jacobi(num, 'c9pt', 1.7);
235     l2_c9pt(k,:) = l2;
236     resid_c9pt(k,:) = resid;
237
238     k = k + 1;
239 end
240 steps = 1:10:200;
241
242 figure()
243 hold on
244 plot(steps, resid_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
245 plot(steps, resid_9pt(1,steps), 'g-.', 'linewidth', 1.8)
246 plot(steps, resid_c9pt(1,steps), 'b--', 'linewidth', 1.8)
247 plot(steps, resid_5pt(2,steps), 'k-', 'linewidth', 1.8)
248 plot(steps, resid_9pt(2,steps), 'g-.', 'linewidth', 1.8)
249 plot(steps, resid_c9pt(2,steps), 'b--', 'linewidth', 1.8)
250 set(gca, 'yscale','log')
251 xlabel('E_iterations','fontsize',18)
252 ylabel('$L_{2}$Norm_of_residual','fontsize',18)
253 legend({'5-Point','9-Point','Corrected_9-Point'},'fontsize',16,'location', '
    northwest')
254 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
255 export_fig('figs/jacobi_residuals_17.eps')
256
257 figure()
258 hold on
259 plot(steps, l2_5pt(1,1:10:200), 'k-', 'linewidth', 1.8)
260 plot(steps, l2_9pt(1,steps), 'g-.', 'linewidth', 1.8)
261 plot(steps, l2_c9pt(1,steps), 'b--', 'linewidth', 1.8)
262 plot(steps, l2_5pt(2,steps), 'k-s', 'linewidth', 1.8)

```



```

263 plot(steps, l2_9pt(2,steps), 'g-.s', 'linewidth', 1.8)
264 plot(steps, l2_c9pt(2,steps), 'b--s', 'linewidth', 1.8)
265 set(gca, 'yscale','log')
266 xlabel('Eiterations', 'fontsize', 18)
267 ylabel('L2 Error', 'fontsize', 18)
268 legend({'5-Point', '9-Point', 'Corrected_9-Point'}, 'fontsize', 16, 'location', 'northwest')
269 set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
270 export_fig('figs/jacobi_l2_17.eps')
271
272 %~~~~~

```

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.

```

1 function [A,F] = construct_5pt(num)
2 % Returns [A,F] for 5-Point stencil
3
4 % Pre-allocate values
5 A = sparse((num+1)^2, (num+1)^2);
6 [edges, nodes] = detect_edges((num+1)^2);
7 F = zeros((num+1)^2,1);
8 h = 1/num;
9
10 % Iterate through all the nodes
11 for k = 1:length(edges)
12
13     % If the node is a boundary element set the index to "1" and F to 0
14     if edges(k,2) == 1
15         A(k,k) = 1;
16         F(k) = 0;
17     end
18
19     % If not a boundary element then conduct the 5-point stencil
20     if edges(k,2) == 0
21         % Get the x,y positions
22         test = nodes(k,2:3);
23         i = test(1); % x-position
24         j = test(2); % y-position
25         for count = 1:length(edges)
26             if nodes(count,2) == i-1 && nodes(count,3) == j
27                 A(k,count) = A(k,count) + 1/h^2;
28             elseif nodes(count,2) == i+1 && nodes(count,3) == j
29                 A(k,count) = A(k,count) + 1/h^2;
30             elseif nodes(count,2) == i && nodes(count,3) == j-1
31                 A(k,count) = A(k,count) + 1/h^2;
32             elseif nodes(count,2) == i && nodes(count,3) == j+1
33                 A(k,count) = A(k,count) + 1/h^2;
34             end
35         end
36         A(k,k) = A(k,k) - 4/h^2;
37         F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
38
39         % Machine precision
40         if abs(F(k)) < 1.0e-16
41             F(k) = 0;

```

```

42         end
43     end
44 end
45 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.

```

1  function [A,F] = construct_9pt(num)
2  % Returns [A,F] for 9-Point stencil
3
4  % Pre-allocate values
5  A = sparse((num+1)^2, (num+1)^2);
6  [edges, nodes] = detect_edges((num+1)^2);
7  F = zeros((num+1)^2,1);
8  h = 1/num;
9
10 % Iterate through all the nodes
11 for k = 1:length(edges)
12
13     % If the node is a boundary element set the index to "1" and F to 0
14     if edges(k,2) == 1
15         A(k,k) = 1;
16         F(k) = 0;
17     end
18
19     % If not a boundary element then conduct the 9-point stencil
20     if edges(k,2) == 0
21         % Get the x,y positions
22         test = nodes(k,2:3);
23         i = test(1); % x-positions
24         j = test(2); % y-positions
25         for count = 1:length(edges)
26             if nodes(count,2) == i-1 && nodes(count,3) == j
27                 A(k,count) = A(k,count) + 4/(6*h^2);
28             elseif nodes(count,2) == i+1 && nodes(count,3) == j
29                 A(k,count) = A(k,count) + 4/(6*h^2);
30             elseif nodes(count,2) == i && nodes(count,3) == j-1
31                 A(k,count) = A(k,count) + 4/(6*h^2);
32             elseif nodes(count,2) == i && nodes(count,3) == j+1
33                 A(k,count) = A(k,count) + 4/(6*h^2);
34             elseif nodes(count,2) == i-1 && nodes(count,3) == j-1
35                 A(k,count) = A(k,count) + 1/(6*h^2);
36             elseif nodes(count,2) == i-1 && nodes(count,3) == j+1
37                 A(k,count) = A(k,count) + 1/(6*h^2);
38             elseif nodes(count,2) == i+1 && nodes(count,3) == j-1
39                 A(k,count) = A(k,count) + 1/(6*h^2);
40             elseif nodes(count,2) == i+1 && nodes(count,3) == j+1
41                 A(k,count) = A(k,count) + 1/(6*h^2);
42             end
43         end
44         A(k,k) = A(k,k) - 20/(6*h^2);
45         F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
46
47         % Machine precision
48         if abs(F(k)) < 1.0e-16
49             F(k) = 0;
50         end
51     end
52 end
53 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.

```

1 function [A,F] = construct_c9pt(num)
2 % Returns [A,F] for corrected 9-Point stencil
3
4 % Pre-allocate values
5 A = sparse((num+1)^2, (num+1)^2);
6 [edges, nodes] = detect_edges((num+1)^2);
7 F = zeros((num+1)^2,1);
8 h = 1/num;
9
10 % Iterate through all the nodes
11 for k = 1:length(edges)
12
13     % If the node is a boundary element set the index to "1" and F to 0
14     if edges(k,2) == 1
15         A(k,k) = 1;
16         F(k) = 0;
17     end
18
19     % If not a boundary element then conduct the corrected 9-point stencil
20     if edges(k,2) == 0
21         % Get the x,y positions
22         test = nodes(k,2:3);
23         i = test(1); % x-position
24         j = test(2); % y-position
25         for count = 1:length(edges)
26             if nodes(count,2) == i-1 && nodes(count,3) == j
27                 A(k,count) = A(k,count) + 4/(6*h^2);
28             elseif nodes(count,2) == i+1 && nodes(count,3) == j
29                 A(k,count) = A(k,count) + 4/(6*h^2);
30             elseif nodes(count,2) == i && nodes(count,3) == j-1
31                 A(k,count) = A(k,count) + 4/(6*h^2);
32             elseif nodes(count,2) == i && nodes(count,3) == j+1
33                 A(k,count) = A(k,count) + 4/(6*h^2);
34             elseif nodes(count,2) == i-1 && nodes(count,3) == j-1
35                 A(k,count) = A(k,count) + 1/(6*h^2);
36             elseif nodes(count,2) == i-1 && nodes(count,3) == j+1
37                 A(k,count) = A(k,count) + 1/(6*h^2);
38             elseif nodes(count,2) == i+1 && nodes(count,3) == j-1
39                 A(k,count) = A(k,count) + 1/(6*h^2);
40             elseif nodes(count,2) == i+1 && nodes(count,3) == j+1
41                 A(k,count) = A(k,count) + 1/(6*h^2);
42             end
43         end
44         A(k,k) = A(k,k) - 20/(6*h^2);
45
46         F(k) = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h)+...
47             1/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h)+...
48             1/12*(-20)*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h)-...
49             4/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h)+...
50             1/12*(-20)*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h)+...
51             1/12*(-20)*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h);
52
53         % Machine precision
54         if abs(F(k)) < 1.0e-16
55             F(k) = 0;
56         end
57     end
58 end
59 end

```

C Implementing Jacobi Method

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

Algorithm 5: Matlab implementation of Jacobi method.

```

1 function [l2err, resid] = jacobi(num, stencil, omega)
2 % Returns L2 error and residuals
3
4 % Pre-allocations
5 U = zeros(num+1);
6 resid_mat = U;
7 resid = zeros(max(size(1:200)),1);
8 l2err = resid;
9 xs = linspace(0,1, num+1);
10 [xs,ys] = meshgrid(xs);
11 uexact = exact_sol(xs, ys);
12 h = 1/num;
13
14 % Execute Jacobi Iteration
15 for iter = 1:200
16     utemp = zeros(num+1);
17
18     % Conduct the 5-point stencil
19     if strcmp(stencil, '5pt')
20         s0 = (-4/h^2);
21         for i = 2:num
22             for j = 2:num
23                 s_5pt = 0;
24                 s_5pt = s_5pt - 4*U(j,i)/h^2;
25                 s_5pt = s_5pt + U(j-1,i)/h^2;
26                 s_5pt = s_5pt + U(j+1,i)/h^2;
27                 s_5pt = s_5pt + U(j,i-1)/h^2;
28                 s_5pt = s_5pt + U(j,i+1)/h^2;
29
30                 RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
31                 rij = RHS - s_5pt;
32                 resid_mat(j,i) = rij;
33                 utemp(j,i) = U(j,i) + omega/s0.*rij;
34             end
35         end
36     % Conduct the 9-point stencil
37     elseif strcmp(stencil, '9pt')
38         s0 = (-20/(6*h^2));
39         for i = 2:num
40             for j = 2:num
41                 s_9pt = 0;
42                 s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
43                 s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
44                 s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
45                 s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
46                 s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
47                 s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
48                 s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
49                 s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
50                 s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);
51
52                 RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
53                 rij = RHS - s_9pt;
54                 resid_mat(j,i) = rij;
55                 utemp(j,i) = U(j,i) + omega/s0.*rij;
56             end
57         end
58     % Conduct the 9-point stencil
59     elseif strcmp(stencil, 'c9pt')
60         s0 = (-20/(6*h^2));
61         for i = 2:num
62             for j = 2:num
63                 s_c9pt = 0;

```

```

64         s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
65         s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
66         s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
67         s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
68         s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
69         s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
70         s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
71         s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
72         s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);
73
74         RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
75             1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
76             1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
77             (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
78             1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
79             1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
80         rij = RHS - s_c9pt;
81         resid_mat(j,i) = rij;
82         utemp(j,i) = U(j,i) + omega/s0.*rij;
83     end
84 end
85 end
86 % Compute the residuals
87 resid(iter) = sqrt(sum(sum((resid_mat.^2)))/(num+1)^2);
88
89 U = utemp;
90 % Compute the L2 error
91 l2err(iter) = sqrt(sum(sum((U-uexact).^2))*h^2);
92 end
93 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.

```

1 function [l2err, resid] = gauss(num, stencil, omega)
2 % Returns L2 error and residual
3
4 % Pre-allocations
5 h = 1/num;
6 U = zeros(num+1);
7 resid_mat = U;
8 resid = zeros(max(size(1:200)),1);
9 l2err = resid;
10 xs = linspace(0,1, num+1);
11 [xs,ys] = meshgrid(xs);
12 uexact = exact_sol(xs, ys);
13
14 % Execute Jacobi Iteration
15 for iter = 1:200
16
17     % Conduct 5-pt stencil
18     if strcmp(stencil, '5pt')
19         s0 = (-4/h^2);
20
21         % Update the red nodes
22         for j = 2:num
23             if mod(j,2) == 0
24                 for i = 2:2:num
25                     s_5pt = 0;
26                     s_5pt = s_5pt - 4*U(j,i)/h^2;
27                     s_5pt = s_5pt + U(j-1,i)/h^2;
28                     s_5pt = s_5pt + U(j+1,i)/h^2;

```

```

29         s_5pt = s_5pt + U(j,i-1)/h^2;
30         s_5pt = s_5pt + U(j,i+1)/h^2;
31
32         RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
33         rij = RHS - s_5pt;
34         resid_mat(j,i) = rij;
35         U(j,i) = U(j,i) + omega/s0.*rij;
36     end
37 else
38     for i = 3:2:num-1
39         s_5pt = 0;
40         s_5pt = s_5pt - 4*U(j,i)/h^2;
41         s_5pt = s_5pt + U(j-1,i)/h^2;
42         s_5pt = s_5pt + U(j+1,i)/h^2;
43         s_5pt = s_5pt + U(j,i-1)/h^2;
44         s_5pt = s_5pt + U(j,i+1)/h^2;
45
46         RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
47         rij = RHS - s_5pt;
48         resid_mat(j,i) = rij;
49         U(j,i) = U(j,i) + omega/s0.*rij;
50     end
51 end
52 end
53
54 % Update the black Nodes
55 for j = 2:num
56     if mod(j,2) == 0
57         for i = 3:2:num-1
58             s_5pt = 0;
59             s_5pt = s_5pt - 4*U(j,i)/h^2;
60             s_5pt = s_5pt + U(j-1,i)/h^2;
61             s_5pt = s_5pt + U(j+1,i)/h^2;
62             s_5pt = s_5pt + U(j,i-1)/h^2;
63             s_5pt = s_5pt + U(j,i+1)/h^2;
64
65             RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
66             rij = RHS - s_5pt;
67             resid_mat(j,i) = rij;
68             U(j,i) = U(j,i) + omega/s0.*rij;
69         end
70     else
71         for i = 2:2:num
72             s_5pt = 0;
73             s_5pt = s_5pt - 4*U(j,i)/h^2;
74             s_5pt = s_5pt + U(j-1,i)/h^2;
75             s_5pt = s_5pt + U(j+1,i)/h^2;
76             s_5pt = s_5pt + U(j,i-1)/h^2;
77             s_5pt = s_5pt + U(j,i+1)/h^2;
78
79             RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
80             rij = RHS - s_5pt;
81             resid_mat(j,i) = rij;
82             U(j,i) = U(j,i) + omega/s0.*rij;
83         end
84     end
85 end
86
87 % Conduct 9-pt stencil
88 elseif strcmp(stencil, '9pt')
89     s0 = (-20/(6*h^2));
90
91     % Update the red nodes
92     for j = 2:num
93         if mod(j,2) == 0
94             for i = 2:2:num
95                 s_9pt = 0;
96                 s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
97                 s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
98                 s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
99                 s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);

```

```

100         s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
101         s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
102         s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
103         s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
104         s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);
105
106         RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
107         rij = RHS - s_9pt;
108         resid_mat(j,i) = rij;
109         U(j,i) = U(j,i) + omega/s0.*rij;
110     end
111 else
112     for i = 3:2:num-1
113         s_9pt = 0;
114         s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
115         s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
116         s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
117         s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
118         s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
119         s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
120         s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
121         s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
122         s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);
123
124         RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
125         rij = RHS - s_9pt;
126         resid_mat(j,i) = rij;
127         U(j,i) = U(j,i) + omega/s0.*rij;
128     end
129 end
130 end
131
132 % Update the black Nodes
133 for j = 2:num
134     if mod(j,2) == 0
135         for i = 3:2:num-1
136             s_9pt = 0;
137             s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
138             s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
139             s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
140             s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
141             s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
142             s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
143             s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
144             s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
145             s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);
146
147             RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
148             rij = RHS - s_9pt;
149             resid_mat(j,i) = rij;
150             U(j,i) = U(j,i) + omega/s0.*rij;
151         end
152     else
153         for i = 2:2:num
154             s_9pt = 0;
155             s_9pt = s_9pt - 20*U(j,i)/(6*h^2);
156             s_9pt = s_9pt + 4*U(j-1,i)/(6*h^2);
157             s_9pt = s_9pt + 4*U(j+1,i)/(6*h^2);
158             s_9pt = s_9pt + 4*U(j,i-1)/(6*h^2);
159             s_9pt = s_9pt + 4*U(j,i+1)/(6*h^2);
160             s_9pt = s_9pt + U(j+1,i-1)/(6*h^2);
161             s_9pt = s_9pt + U(j+1,i+1)/(6*h^2);
162             s_9pt = s_9pt + U(j-1,i-1)/(6*h^2);
163             s_9pt = s_9pt + U(j-1,i+1)/(6*h^2);
164
165             RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h);
166             rij = RHS - s_9pt;
167             resid_mat(j,i) = rij;
168             U(j,i) = U(j,i) + omega/s0.*rij;
169         end
170     end

```

```

171     end
172
173     % Conduct corrected 9-pt stencil
174     elseif strcmp(stencil, 'c9pt')
175         s0 = (-20/(6*h^2));
176
177         % Update the red nodes
178         for j = 2:num
179             if mod(j,2) == 0
180                 for i = 2:2:num
181                     s_c9pt = 0;
182                     s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
183                     s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
184                     s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
185                     s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
186                     s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
187                     s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
188                     s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
189                     s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
190                     s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);
191
192                     RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
193                         1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
194                         1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
195                         (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
196                         1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
197                         1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
198                     rij = RHS - s_c9pt;
199                     resid_mat(j,i) = rij;
200                     U(j,i) = U(j,i) + omega/s0.*rij;
201                 end
202             else
203                 for i = 3:2:num-1
204                     s_c9pt = 0;
205                     s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
206                     s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
207                     s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
208                     s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
209                     s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
210                     s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
211                     s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
212                     s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
213                     s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);
214
215                     RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
216                         1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
217                         1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
218                         (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
219                         1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
220                         1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
221                     rij = RHS - s_c9pt;
222                     resid_mat(j,i) = rij;
223                     U(j,i) = U(j,i) + omega/s0.*rij;
224                 end
225             end
226         end
227
228         % Update the black Nodes
229         for j = 2:num
230             if mod(j,2) == 0
231                 for i = 3:2:num-1
232                     s_c9pt = 0;
233                     s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
234                     s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
235                     s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
236                     s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
237                     s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
238                     s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
239                     s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
240                     s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
241                     s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);

```



```

242     RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
243           1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
244           1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
245           (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
246           1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
247           1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
248     rij = RHS - s_c9pt;
249     resid_mat(j,i) = rij;
250     U(j,i) = U(j,i) + omega/s0.*rij;
251     end
252   else
253     for i = 2:2:num
254       s_c9pt = 0;
255       s_c9pt = s_c9pt - 20*U(j,i)/(6*h^2);
256       s_c9pt = s_c9pt + 4*U(j-1,i)/(6*h^2);
257       s_c9pt = s_c9pt + 4*U(j+1,i)/(6*h^2);
258       s_c9pt = s_c9pt + 4*U(j,i-1)/(6*h^2);
259       s_c9pt = s_c9pt + 4*U(j,i+1)/(6*h^2);
260       s_c9pt = s_c9pt + U(j+1,i-1)/(6*h^2);
261       s_c9pt = s_c9pt + U(j+1,i+1)/(6*h^2);
262       s_c9pt = s_c9pt + U(j-1,i-1)/(6*h^2);
263       s_c9pt = s_c9pt + U(j-1,i+1)/(6*h^2);
264       RHS = -20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h) + ...
265             1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j)*h))+...
266             1/12*(-20*pi^2*sin(2*pi*(i-2)*h)*sin(4*pi*(j-1)*h))+...
267             (-4/12)*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-1)*h))+...
268             1/12*(-20*pi^2*sin(2*pi*(i)*h)*sin(4*pi*(j-1)*h))+...
269             1/12*(-20*pi^2*sin(2*pi*(i-1)*h)*sin(4*pi*(j-2)*h));
270       rij = RHS - s_c9pt;
271       resid_mat(j,i) = rij;
272       U(j,i) = U(j,i) + omega/s0.*rij;
273     end
274   end
275 end
276
277 end
278
279 end
280 % Compute the residuals from the residual matrix
281 resid(iter) = sqrt(sum(sum((resid_mat.^2)))/(num+1)^2);
282
283 % Compute the L2 error
284 l2err(iter) = sqrt(sum(sum((U-uexact).^2)*h^2);
285 end
286 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.

```

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

```

1 function err = l2error(uf, uexact, xs)
2   h = xs(1,2)-xs(1,1);
3
4   err = 0;
5   for i = 1:length(uf)
6     for j = 1:length(uf)
7       err = err + (uf(i,j) - uexact(i,j))^2;
8     end
9   end
10  err = sqrt(h^2*err);
11 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.

```

1 function [edges, nodes] = detect_edges(num)
2 % Returns the edges and nodes for each NxN meshgrid
3
4 % Pre-allocate the nodes
5 nodes = zeros(num,3);
6 edges = zeros(num,2);
7 k = 1; j = 1;
8 for i = 1:num
9   % Create a Nx3 matrix of the nodes with their locations
10  nodes(i,:) = [i, k, j];
11  k = k + 1;
12  if k == sqrt(num)+1
13    j = j + 1;
14    k = 1;
15  end
16 end
17
18 % Two logicals to determine the indices of the boundary nodes
19 sides = nodes < 1.01;
20 sides2 = nodes > (sqrt(num)-0.01);
21 edgevals = sides + sides2;
22
23 edges(:,1) = 1:num;
24 for i = 1:num
25   % Checks to see if either the x,y are a boundary nodes
26   if edgevals(i,2) == 1 || edgevals(i,3) == 1
27     edges(i,2) = 1;
28   end
29 end
30
31 end

```