Computational techniques for solving the Poisson equation

Akarsh Shukla
(2020PHY1216)

Brahmanand Mishra
(2020PHY1184)

Shashvat Jain
(2020PHY1114)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

November 13, 2021

Project Report Submitted to

Dr. Mamta and Dr. H. C. Ramo

as part of internal assessment for the course

"32223902 - Computational Physics Skills"

Abstract

The project report brings light to the application of finite difference method to computationally solve a physics problem belonging to a class of poisson equation. It covers the extensive analysis of various iterative schemes, namely Jacobi,Gauss-Seidel and SOR, used for solving the system of linear equations produced by finite difference method. This report also includes the conversion of physical problem which involves the study of the potential scalar field inside a interleaved parallel plate capacitor to a mathematical one by means of normalisation of variables. A brief discussion has been carried out on the local truncation error of finite difference approximations. Further, the iterative schemes have been analysed on the grounds of number of iterations and computational time required for reaching relative tolerance. Results and practices carried out in the making of this report certainly lead the reader to beneficial educational insights into computational physics and its application.
This report is not meant to be a thorough investigation of the problem and should not be seen as a research project instead it should be seen as a motivating discussion on the topic demanding further enquiry in the future.

Contents

1 Introduction

1.1 Motivation

We first encountered Laplace Equation during our course in electricity and magnetism in second semester and we were fascinated with how one can calculate the potential in a region just by knowing the boundary condition, of course the region has to be charge free for applying Laplace Equation. After Laplace Equation, we were introduced to Poisson Equation using which we were able to solve for region having charges(or sources). When we were given the oppurtunity to choose a project in our computational physics this semester, it did not take us long to decide the topic for project.

1.2 General Idea

In our project we will try to tackle the Laplace and Poisson equaton which is an ellipitic linear partial differential equation having application in various fields of physics ranging from thermodynamics, electrostatics etc. We will solve the equation computationally using the method of finite differences in two dimensions for rectangular membrane. We will first convert the partial differential equation into a system of linear difference equation and then use three different iteration schemes for solving those system of linear equation.

1.3 Poisson and Laplace Equation

The general form of Poisson's equation in Euclidean space is given here,

$$\vec{\nabla}^2\varphi(\vec{r}) = f(\vec{r})$$

where, $f(\vec{r})$, $\varphi(\vec{r})$ are the functions of position vector $\vec{r}$. $f(\vec{r})$ is given and $\varphi(\vec{r})$ is sought.
In our case $\varphi(\vec{r})$ is electrostatic potential, $f(\vec{r})$ is the charge density of region in which solution is required. In a special case in which $f(\vec{r}) = 0$ we can apply same process for solving laplace equation for a region free of charge.

1.4 Plan of Report

In theory section we will formulate and explain the physical problem we have chosen to solve using finite difference method. In methodology we will explain our methods and explain the algorithm followed for programming. In numerical anlaysis section we will analyse our results compared to what we expect to obtain and also compare different iterative methods based on our computational results.And in conclusion we would like to discuss our experiences and results in brief.

2 Theory

2.1 Physical Problem

Our problem consists of identical infinitely-long thin metal plates $A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3$ and B_4 are placed in an *interleaved* arrangement as dipicted in figure 1. Group of plates A_i and B_i are connected to the terminal A and B, respectively, by means of gold wires. The terminals are connected to different potential sources U_A and U_B respectively.
This arrangement is called an interleaved capacitor, Our goal will be to approximate the potential distribution (in two dimensions considering symmetry along the the third axis which will be dropped) inside the capacitor after we disconnect the capacitor from sources, treating the interior plates as line

charge distributions.

$$U_A = 5V$$
$$U_B = -5V$$
$$\text{Capacitance } = 0.1\mu F$$
$$\text{Distance between plates } (d) = 0.5\mu m$$
$$\textit{Dimensions: } 4 \times 4.4\mu m$$

This arrangement can be seen as a number of parallel plate capacitors connected in parallel to each other as seen in fig1(b), if C_0 is the capacitance of each capacitor in parallel and C is the capacitance of the entire arrangement then,

$$C_0 = C/8$$

Also we know that for parallel plate capacitors with cross-section area A and distance d between the plates,

$$C_0 = \epsilon_0 \frac{A}{d} \implies A = \frac{C_0 d}{\epsilon_0} \tag{2.1}$$

Therefore the charge distribution on any plate A_i is given by,

$$\rho_A = \frac{(C \times V)}{5A} \implies \rho_A = 2\epsilon_0 \times 10^5 Cm^{-2} \tag{2.2}$$

Similarly on B_i,

$$\rho_B = -2\epsilon_0 \times 10^5 Cm^{-2} \tag{2.3}$$

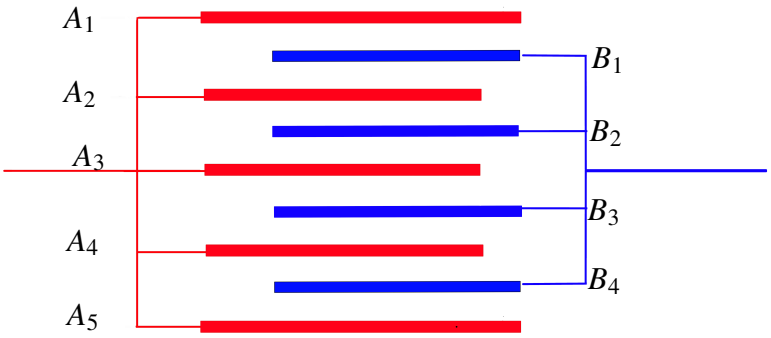


Figure 1: Cross-sectional view of interleaved capacitor

2.2 Mathematical Formulation

Let $U(x, y)$ and $\vec{E}(x, y)$ be the potential and Electric field distribution defined in the region of our arrangement $(x, y) \in \Omega := [0, 4\mu m] \times [0, 4.4\mu m]$.
We know from maxwell's laws that for static electric fields $\vec{\nabla} \cdot \vec{E} = \rho/\epsilon_0$, $\vec{\nabla} \times \vec{E} = 0$
From the later we get $E = -\vec{\nabla}U$, substituting this back into the former we get,

$$\vec{\nabla}^2 U = \frac{-\rho}{\epsilon_0}$$

In two dimensions with euclidean coordinate system the equation reduces to,

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = -\frac{\rho(x,y)}{\epsilon_0} \tag{2.4}$$

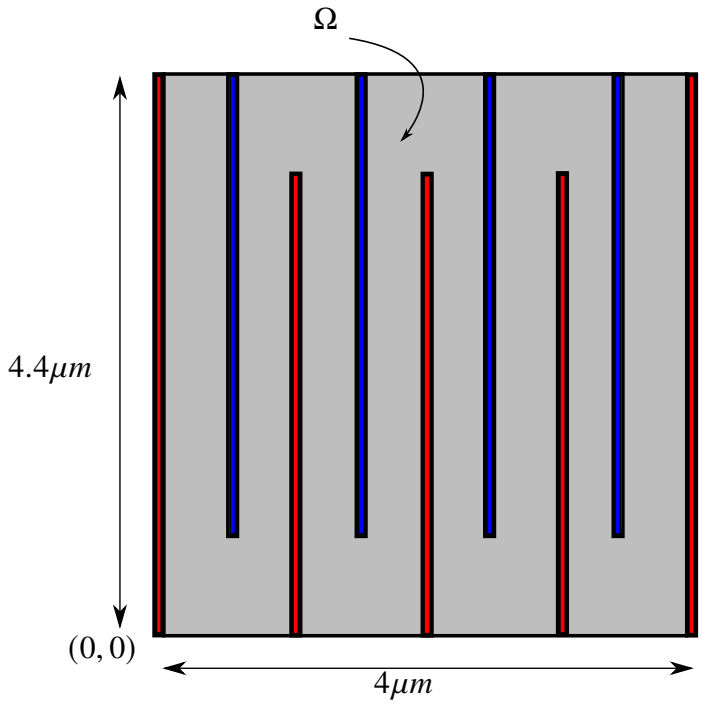


Figure 2: Cross-sectional view of the region Ω in question.

where,

$$\rho(x,y) = \begin{cases} -2\epsilon_0 \times 10^5 Cm^{-2} & : \text{if} \quad (x,y) \in B_i \quad \text{where} \quad i = 1,2,3,4 \\ 2\epsilon_0 \times 10^5 Cm^{-2} & : \text{if} \quad (x,y) \in A_i \quad \text{where} \quad i = 2,3,4 \\ 0Cm^{-2} & : \quad \text{elsewhere} \end{cases} \tag{2.5}$$

Now according to our given arrangement,

$$B_1 = \{(x^*, y^*) \quad : \quad x^* = 0.5\mu m \quad ; \quad 0.4\mu m \leq y^* \leq 4.4\mu m\} \tag{2.6}$$
$$B_2 = \{(x^*, y^*) \quad : \quad x^* = 1.5\mu m \quad ; \quad 0.4\mu m \leq y^* \leq 4.4\mu m\} \tag{2.7}$$
$$B_3 = \{(x^*, y^*) \quad : \quad x^* = 2.5\mu m \quad ; \quad 0.4\mu m \leq y^* \leq 4.4\mu m\} \tag{2.8}$$
$$B_4 = \{(x^*, y^*) \quad : \quad x^* = 3.5\mu m \quad ; \quad 0.4\mu m \leq y^* \leq 4.4\mu m\} \tag{2.9}$$
$$A_2 = \{(x^*, y^*) \quad : \quad x^* = 1\mu m \quad ; \quad 0\mu m \leq y^* \leq 4\mu m\} \tag{2.10}$$
$$A_3 = \{(x^*, y^*) \quad : \quad x^* = 2\mu m \quad ; \quad 0\mu m \leq y^* \leq 4\mu m\} \tag{2.11}$$
$$A_4 = \{(x^*, y^*) \quad : \quad x^* = 3\mu m \quad ; \quad 0\mu m \leq y^* \leq 4\mu m\} \tag{2.12}$$

We get the following boundary conditions for the above boundary value problem,

$$U(0,y) = +5V \quad ; \quad U(4,y) = +5V \tag{2.13}$$
$$U_y(x,0) = 0V/m \quad ; \quad U_y(x,4.4) = 0V/m \tag{2.14}$$

Non-Dimensionalizing the variables,[1]
Let new dimensionless variables,

$$x' = \frac{x}{s} \qquad y' = \frac{y}{s} \qquad U' = \frac{U}{\nu} \tag{2.15}$$

where s and ν are known constant scaling factors having dimensions of length and electric potential respectively.

The B.V.P reduces to,

$$\frac{\partial^2 U'(x',y')}{\partial x'^2} + \frac{\partial^2 U'(x',y')}{\partial y'^2} = -\frac{\rho'(x',y')s^2}{\epsilon_0 \nu} \tag{2.16}$$

where,

$$\rho'(x', y') = \begin{cases} -2\epsilon_0 \times 10^5 & : \text{if} \quad (x', y') \in B_i \quad \text{where} \quad i = 1, 2, 3, 4 \\ 2\epsilon_0 \times 10^5 & : \text{if} \quad (x', y') \in A_i \quad \text{where} \quad i = 2, 3, 4 \\ 0 & : \quad \text{elsewhere} \end{cases} \tag{2.17}$$

Now according to our given arrangement,

$$B_1 = \{(x^*, y^*) \quad : \quad x^* = 0.5/s \quad ; \quad 0.4/s \leq y^* \leq 4.4/s\} \tag{2.18}$$
$$B_2 = \{(x^*, y^*) \quad : \quad x^* = 1.5/s \quad ; \quad 0.4/s \leq y^* \leq 4.4/s\} \tag{2.19}$$
$$B_3 = \{(x^*, y^*) \quad : \quad x^* = 2.5/s \quad ; \quad 0.4/s \leq y^* \leq 4.4/s\} \tag{2.20}$$
$$B_4 = \{(x^*, y^*) \quad : \quad x^* = 3.5/s \quad ; \quad 0.4/s \leq y^* \leq 4.4/s\} \tag{2.21}$$
$$A_2 = \{(x^*, y^*) \quad : \quad x^* = 1/s \quad ; \quad 0/s \leq y^* \leq 4/s\} \tag{2.22}$$
$$A_3 = \{(x^*, y^*) \quad : \quad x^* = 2/s \quad ; \quad 0/s \leq y^* \leq 4/s\} \tag{2.23}$$
$$A_4 = \{(x^*, y^*) \quad : \quad x^* = 3/s \quad ; \quad 0/s \leq y^* \leq 4/s\} \tag{2.24}$$

We get the following boundary conditions for the above boundary value problem,

$$U'(0, y) = +5/\nu \quad ; \quad U'(4, y) = +5/\nu \tag{2.25}$$
$$U'_y(x, 0) = 0 \quad ; \quad U'_y(x, 4.4) = 0 \tag{2.26}$$

3 Methodology

3.1 Finite Difference Method

Finite Difference Methods(FDM) are used for approximating the solution of partial differential equations over a set of finite points, arranged in a geometrical structure called a **mesh**[1], in the continous domain of solution. The methods involve the idea of reducing the given PDE, by means of truncated taylor series approximation of the derivatives, to a difference equation which is much easier to digest numerically.

3.1.1 Finite Difference Approximations

The quality of the solution depends on the quality of approximations made to the derivatives. Consider this one-dimensional structured mesh of nodes $(x_0, x_1, x_2, .., x_i, .., x_n)$ at which the solution $U(x_i)$ is to be found, such that the difference $h = x_{i+1} - x_i$ is constant throughout the mesh and $x_i \equiv x_0 + ih$.

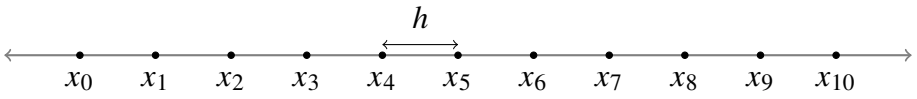
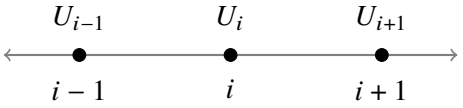


Figure 3: 1D mesh with 11 nodes and a meshsize h



Let U_i represent the solution at the i-th node and

$$\left.\frac{\partial U}{\partial x}\right|_{x_i} = U_x(x_0 + ih) \equiv U_x|_i$$

$$\left.\frac{\partial^2 U}{\partial x^2}\right|_{x_i} = U_{xx}(x_0 + ih) \equiv U_{xx}|_i$$

[1]An object which consists of points which are spaced in a specific geometrical pattern is referred to as a **mesh** and each point in this mesh is called a **node**. The distance between any two adjacent nodes in a mesh with uniform spacing is called its **meshsize**

The first order derivative can be defined as,

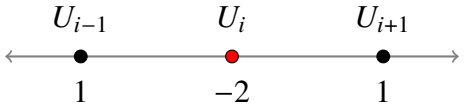
$$U_x|_i = \lim_{h\to 0} \frac{U_{i+1} - U_i}{h}$$
$$\text{or,} \qquad U_x|_i = \lim_{h\to 0} \frac{U_i - U_{i-1}}{h}$$
$$\text{or,} \qquad U_x|_i = \lim_{h\to 0} \frac{U_{i+1} - U_{i-1}}{2h}$$

Finite difference approximations are obtained by dropping the limit and can be written as,

$$\begin{aligned}
&\text{Forward Difference} && U_x|_i \approx \frac{U_{i+1} - U_i}{h} \equiv \delta_x^+ U_i \\
&\text{Backward Difference} && U_x|_i \approx \frac{U_i - U_{i-1}}{h} \equiv \delta_x^- U_i \\
&\text{Central Difference} && U_x|_i \approx \frac{U_{i+1} - U_{i-1}}{2h} \equiv \delta_{2x} U_i
\end{aligned}$$

Where $\delta_x^+, \delta_x^-, \delta_{2x}$ are called the **finite diference operators** for approximating **first-order derivatives** and their expansion is called the **finite difference quotient**, each representing forward,backward and centered respectively. Second and Higher order finite difference Quotients can also be obtained using these these operators,

$$\begin{aligned}
U_{xx}|_i &= \lim_{h\to 0} \frac{U_x(x_i + \frac{h}{2}) - U_x(x_i - \frac{h}{2})}{h} \\
&= \lim_{h\to 0} \frac{1}{h}\left[\frac{U(x+h) - U(x)}{h} - \frac{(U(x) - U(x-h))}{h}\right] \\
&= \lim_{h\to 0} \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2} \\
&\approx \boxed{\delta_x^2 U_i \equiv \frac{1}{h^2}(U_{i+1} - 2U_i + U_{i-1})} \qquad \text{[Central second-order Difference]}
\end{aligned}$$



The vector of coefficients of the function values at various nodes forms what is called the **stencil** of the finite difference operator and it uniquely idetifies the operator. The combination $(1, -2, 1)$ is called a **three point stencil** as it combines function values from three different points on the mesh. It is fairly obvious to notice that any finite difference operator for any derivative at any node is just a linear combination of the function values at various neighbourhood nodes.

3.1.2 Local Truncation Error of Finite Difference Approximations

The *"error"* that accompanies *"approximations"* in the method must also be acccounted for. In this section, the truncation error in the derivative approximations is ascertained which will later help us deduce the error in PDE's solved using these approximations.

The local truncation error for derivative approximations is defined here as the difference between the exact value of the derivate and the approximated value at node i, it can be calculated using Taylor series expansions about i,

For Forward difference operator,[2][3]

$$\begin{aligned}\tau &\equiv \delta_x^+ U_i - U_x|_i \\ &= \frac{1}{\Delta x}\left(U_{i+1} - U_i\right) - U_x|_i \\ &= \frac{1}{\Delta x}\left[\left(U_i + \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i + O((\Delta x)^3)\right) - U_i\right] - U_x|_i \\ &= \frac{1}{2}\Delta x U_{xx}|_i + O((\Delta x)^2) = O(\Delta x)\end{aligned}$$

For Backward difference operator,

$$\begin{aligned}\tau &\equiv \delta_x^- U_i - U_x|_i \\ &= \frac{1}{\Delta x}\left(U_i - U_{i-1}\right) - U_x|_i \\ &= \frac{1}{\Delta x}\left[U_i - \left(U_i - \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i + O((\Delta x)^3)\right)\right] - U_x|_i \\ &= -\frac{1}{2}\Delta x U_{xx}|_i + O((\Delta x)^2) = O(\Delta x)\end{aligned}$$

For Central difference operator,

$$\begin{aligned}\tau &\equiv \delta_{2x} U_i - U_x|_i \\ &= \frac{1}{2\Delta x}\left(U_{i+1} - U_{i-1}\right) - U_x|_i \\ &= \frac{1}{2\Delta x}\Bigg[\left(U_i + \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i + \frac{1}{6}(\Delta x)^3 U_{xxx}|_i + \frac{1}{12}(\Delta x)^4 U_{xxxx}|_i + O((\Delta x)^5)\right) \\ &\quad - \left(U_i - \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i - \frac{1}{6}(\Delta x)^3 U_{xxx}|_i + \frac{1}{12}(\Delta x)^4 U_{xxxx}|_i + O((\Delta x)^5)\right)\Bigg] - U_x|_i \\ &= -\frac{1}{6}\Delta x^2 U_{xxx_i} + O((\Delta x)^4) = O((\Delta x)^2)\end{aligned}$$

where in the above expressions we assume that the Higher order derivatives of U at i are well defined. For a fairly small Δx (less than 1) we can confidently say that $O(\Delta x^2)$ is samller than $O(\Delta x)$[1]. Thus we note that the centered difference approximation (second-order accurate) approximates the derivative more accurately than either of the *one-sided diferences* which are first-order accurate.[2]

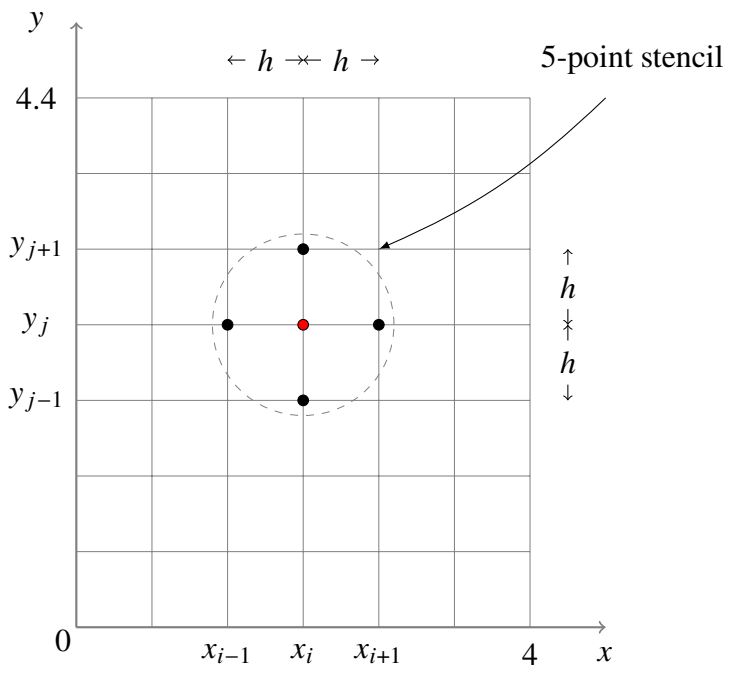
Similarly, Approximation of second-order derivative,

$$\begin{aligned}\tau &\equiv \delta_x^2 U_i - U_{xx}|_i \\ &= \frac{1}{(\Delta x)^2}(U_{i+1} - 2U_i + U_{i-1}) - U_{xx}|_i \\ &= \frac{1}{(\Delta x)^2}\Bigg[\left(U_i + \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i + \frac{1}{6}(\Delta x)^3 U_{xxx}|_i + \frac{1}{12}(\Delta x)^4 U_{xxxx}|_i + O((\Delta x)^5)\right) - 2U_i \\ &+ \left(U_i - \Delta x U_x|_i + \frac{1}{2}(\Delta x)^2 U_{xx}|_i - \frac{1}{6}(\Delta x)^3 U_{xxx}|_i + \frac{1}{12}(\Delta x)^4 U_{xxxx}|_i + O((\Delta x)^5)\right)\Bigg] - U_{xx}|_i \\ &= O((\Delta x)^2)\end{aligned}$$

Thus, the second-order derivative approximator is also second order accurate.

[1]The definition of the "big O" notation says that if for given functions $f(x)$ and $g(x)$ for $x \in S$ where S is some subset of $\mathbf{R}$, there exists a positive constant A such that $|f(x)| \leq A|g(x)| \; \forall \; x \in S$, we say that $f(x)$ is the "big O" of $g(x)$ or that $f(x)$ is of order of $g(x)$, mathematically given by $f(x) = O(g(x))$

[2]Forward and Backward differences are also called one-sided differences



3.1.3 Reducing PDE to a discretised difference equation

First we decompose our continous domain Ω of $U(x, y)$ to a discretised one by overlaying it with a uniformly structured rectangular mesh of meshsize $\triangle x = \triangle y = h$ and working only on the nodes of the mesh.

Therefore we have,$U_{i,j} = U(x_i, y_j)$ and $\rho_{i,j} = \rho(x_i, y_j)$ $\forall \quad i \in \{0, 1, \ldots, N_x\}; \quad j \in \{0, 1, \ldots, N_y\}$.

where $N_x = \frac{4}{h}$ and $N_y = \frac{4.4}{h}$

We replace the second-order derivatives in partial differential equation (2.16) with central difference operators,

$$\delta_x^2 U_{i,j} + \delta_y^2 U_{i,j} = -\frac{\rho'_{i,j} s^2}{\epsilon_0 \nu} \tag{3.1}$$

$$\frac{1}{h^2}(U_{i+1,j} + U_{i-1,j} - 4U_{i,j} + U_{i,j+1} + U_{i,j-1}) = -\frac{\rho'_{i,j} s^2}{\epsilon_0 \nu} \tag{3.2}$$

After rearranging we obtain the useful relation,

$$U_{i,j} = \frac{1}{4}\left[U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} + h^2 \frac{\rho'_{i,j} s^2}{\epsilon_0 \nu}\right] \tag{3.3}$$

$$\forall \quad i \in \{1, 2, \ldots, N_x - 1\}; \quad j \in \{0, 1, \ldots, N_y\} \tag{3.4}$$

The Dirichlet boundary conditions get translated to,

$$U_{i,j} = 5 \quad \forall \quad i \in \{0, N_x\}; \quad j \in \{0, 1, \ldots, N_y\}$$

and Neumann to,

$$U_{i,j+1} = U_{i,j-1} \quad \forall \quad j = N_y; \quad i \in \{0, 1, \ldots, N_x\}$$
$$U_{i,j-1} = U_{i,j+1} \quad \forall \quad j = 0; \quad i \in \{0, 1, \ldots, N_x\}$$

3.2 Iterative methods to solve linear algebraic equations

In the last section we have discussed how to reduce a PDE to a linear combination of function values at various nodes by means of the method of finite differences. If we let the function value at any node as

an unknown variable then the stencil when applied to all interior nodes gives rise to a system of linear algebraic equations, which may be very large. A two-dimensional problem like ours may lead to a system of several thousand unknowns and the three-dimensional problems involving several hundred thousand unknowns, are common in real engineering situations. The solution of such a system is a major problem in itself because traditional methods like Gaussian-elimination result in large computational times, we are therefore forced to employ faster methods. As we have seen above, the system of equations produced by a discretisation has many special features and an efficient solution procedure must exploit these. The most obvious property of the system is that it is extremely sparse. Even when there are many thousand unknowns, each equation will involve one unknown and the unknowns at its immediate neighbourhood. In particular, if we write the equations in the conventional notation,

$$A\vec{x} = \vec{b} \tag{3.5}$$

where A is an N × N matrix, b the given data vector and x the vector of N unknown interior mesh values, there is an implied one-dimensional ordering of these values which is somewhat unnatural and obscures the important property that only immediate neighbours are involved. Each row of the matrix A involves only a very small number of non-zero elements, commonly five or seven; moreover in many problems a suitable ordering of the unknowns will lead to a matrix in which these non-zero elements occur in a regular pattern. Instead of solving for this vector equation in matrix form we solve by applying the Finite difference equation to each node in the mesh which in practice is the same as solving the vector equation 3.5 but without explicitly stating the matrices A and b. Following sections describe the various methods we successfully employed for the same.[45]

3.2.1 Jacobi Method

Jacobi method starts with a set of initial guess values for the unknowns and then gets new value for the unkowns by substituting the quess values in the equations one at a time,the process is repeated with the newly produced value of the unkowns.

Jacobi Method (X,P,h,ϵ_r)
This verison of jacobi method is optimised for poisson on a uniformly structured rectangular mesh, It takes the initial guess values at all nodes, the stencil and iteratively solves the stencil over all interior nodes unitl the relative tolerance is reached.

Input: $\mathbf{X^{(0)}}$: $N_x \times N_y$ matrix of initial guess values, value of step size h, a matrix containing the initial charge configuration **P**, N an upper bound on the number of iterations
Output: $\mathbf{X^{(n*)}}$ matrix containing the values of potential on all satisfying the relation 3.3
for *n in* $1, \ldots N$ **do**
  for $i = 1, 2, 3, \cdots, N_x - 1$ **do**
    for $j = 0, 1, 2, \cdots, N_y$ **do**
      if $j = 0$ **then**
        $X^{(n)}_{i,j-1} = X^{(n)}_{i,j+1}$
      if $j = N_y$ **then**
        $X^{(n)}_{i,j+1} = X^{(n)}_{i,j-1}$
      $X^{(n+1)}_{i,j} = \frac{1}{4}\left[X^{(n)}_{i+1,j} + X^{(n)}_{i-1,j} + X^{(n)}_{i,j+1} + X^{(n)}_{i,j-1} + h^2\frac{P_{i,j}s^2}{\epsilon_0 \nu}\right]$
  if $\max_{i,j}\{\frac{X^{(n+1)} - X^{(n)}}{X^{(n+1)}}\} \leq \epsilon_r$ **then**
    return Output
    break

Algorithm 1: Jacobi Method

3.2.2 Gauss-Seidel Method

The Gauss-Seidel method applies the stencil on the new unkowns as it iterates over the system of equations unlike Jacobi method which only works on the unknown values of the previous iteration.

Gauss-Seidel Method (X,P,h,ϵ_r)

This verison of Gauss-Seidel method is optimised for poisson on a uniformly structured rectangular mesh, It takes the initial guess values at all nodes, the stencil and iteratively solves the stencil over all interior nodes unitl the relative tolerance is reached.

Input: $\mathbf{X^{(0)}}$: $N_x \times N_y$ matrix of initial guess values, value of step size h, a matrix containing the initial charge configuration **P**, N an upper bound on the number of iterations

Output: $\mathbf{X^{(n*)}}$ matrix containing the values of potential on all satisfying the relation 3.3

for *n in* $1, \dots N$ **do**
  for $i = 1, 2, 3, \cdots, N_x - 1$ **do**
    for $j = 0, 1, 2, \cdots, N_y$ **do**
      if $j = 0$ **then**
        $X_{i,j-1}^{(n+1)} = X_{i,j+1}^{(n)}$
      if $j = N_y$ **then**
        $X_{i,j+1}^{(n)} = X_{i,j-1}^{(n)}$
      $X_{i,j}^{(n+1)} = \frac{1}{4}\left[X_{i+1,j}^{(n)} + X_{i-1,j}^{(n+1)} + X_{i,j+1}^{(n)} + X_{i,j-1}^{(n+1)} + h^2\frac{P_{i,j}s^2}{\epsilon_0\nu}\right]$
  if $\max_{i,j}\{\frac{X^{(n+1)}-X^{(n)}}{X^{(n+1)}}\} \leq \epsilon_r$ **then**
    return Output
    break

Algorithm 2: Jacobi Method

3.2.3 Successive Over-Relaxation(SOR)

Successive over-relaxation or SOR is a method that belongs to a class of methods called Relaxation methods. Such so-called "relaxation methods" reached a high state of development in the l940s One such result was a modification of the Gauss–Seidel procedure which is now called successive over-relaxation or the SOR method. Taking the recent values of the unknowns. We proceed by calculating the correction which would be given by the Gauss–Seidel iteration (iterating on new unknowns), and then multiply this correction by ω before adding to the previous value. The term over-relaxation then implies that $\omega > 1$.

S.O.R Method (X,P,h,ϵ_r, ω)

This verison of SOR method is optimised for poisson on a uniformly structured rectangular mesh, It takes the initial guess values at all nodes, the stencil and iteratively solves the stencil over all interior nodes unitl the relative tolerance is reached.

Input: $\mathbf{X^{(0)}}$: $N_x \times N_y$ matrix of initial guess values, value of step size h, a matrix containing the initial charge configuration **P**, N an upper bound on the number of iterations,ω the relaxation factor

Output: $\mathbf{X^{(n*)}}$ matrix containing the values of potential on all satisfying the relation 3.3

for *n in* $1, \dots N$ **do**
  for $i = 1, 2, 3, \cdots, N_x - 1$ **do**
    for $j = 0, 1, 2, \cdots, N_y$ **do**
      if $j = 0$ **then**
        $X_{i,j-1}^{(n+1)} = X_{i,j+1}^{(n)}$
      if $j = N_y$ **then**
        $X_{i,j+1}^{(n)} = X_{i,j-1}^{(n)}$
      $X_{i,j}^{(n+1)} = X_{i,j}^{(n)} + \omega\left(\frac{1}{4}\left[X_{i+1,j}^{(n)} + X_{i-1,j}^{(n+1)} + X_{i,j+1}^{(n)} + X_{i,j-1}^{(n+1)} + h^2\frac{P_{i,j}s^2}{\epsilon_0\nu}\right] - X_{i,j}^{(n)}\right)$
  if $\max_{i,j}\{\frac{X^{(n+1)}-X^{(n)}}{X^{(n+1)}}\} \leq \epsilon_r$ **then**
    return Output
    break

Algorithm 3: SOR Method

4 Numerical Analysis

Although one certainly would have some intuition about how the potential distribution should look like if the distribution had lesser number of charge distributions, the intuition tends to be not so reliable in complex systems. Therefore it becomes natural to approach computaional methods for help. The computaional results themselves have to be judged, one way to do so is a brief check that the result vaguely approximates our intuition, another method is by employing different computaional methods to make sure that the method is not biased to a particular solution. For example, if SOR produces a wildly different solution than Jacobi or the former takes more number of iterations than later then certainly either one of the solvers is wrongly implemented. Before we move on to analysing the various results we obtained using the different solving methods, we will first establish the intuitive expectations that we first put forward while framing the problem and before obtaining the final results. Later we will have a short discussion about the role discritization plays in our solution.

4.1 Expectations

To remind ourselves, our problem consists of an interleaved capacitor which is charged to a potential of $\pm 5V$ and then disconnnected. In our problem the first plate A_1 and the last plate A_5 of the capacitor were maintained at $+5V$. We are solving our problem assuming perfect symmetry along the third direction that is along the surface of the plates.(the z-direction).

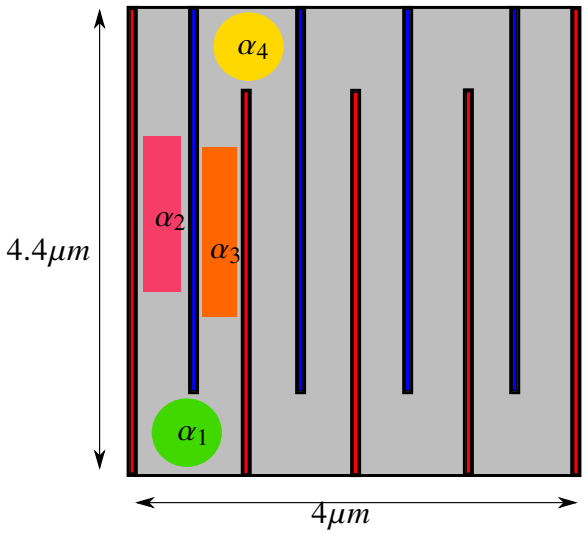


Figure 4: This diagram represents the region Ω being studied, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are the only unique regions in ω that were analysed.

- **Symmetry:** Now since our arrangement is symmetrical, the resulting solution obtained should also be symmetrical about the line $x = 2$.
- High Potential near the positively charged plates (i.e. the plates denoted by A in the figure1) of capacitor and low potential near the negatively charged plates but we are considerig them in two dimension so they will behave as the "line charge distributions. Since we are ensuring the value at the boundary reamins constant at $+5volts$ so the distribution should contain a positive spike in value of potential at the location of A_i having positive potential at the beginning and negative spike at places of negative potential
- In region α_2 potential is fixed at $+5volts$ at $x = 0$ and stretches to $x = 0.5$ where B_1 was initially charged with $-5volts$. We expect to see a decline in potential in region α_2 due to initial conditions across x direction.

- The B_1 plate extends for $y \in (0.4, 4.4)$ so we expect to see concentration of negative charges hence negative potential for higher values of y where initially there was presence of negatively charged potential plate B_1 and a comparitively higher potential for α_1 or lower values of y because it is surrounded by two positively charged plates.
- For region α_3 we expect to see a rise in potential from $x = 0.5$ to $x = 1$ because it is surrounded by negatively charged potential plate at $x = 0.5$ and postively charged plate at $x = 1$.
- The plate A_2 ends at the boundary of α_4 and it is surrounded by two negatively charged plates on both sides so we expect to see a concentration of negatively charged potential in region α_3.
- For region $x \in (1, 1.5)$ the trend should be similar to region α_2 and for region $x \in (1.5, 2)$ we expect trend to be similar to α_4 .
- Now due to symmetry we expect to see the potential distribution after $x = 2$ same as before it with the line $x = 2$ as mirror

4.2 Results

Now that we have converted the problem into a system of linear equations by applying the relation 3.3 to each node on the mesh and analysed the conversion, we move on to analyse the computational results obtained using three different iterative methods namely Jacobi, Gauss-Seidel and S.O.R to solve the system of linear equations.

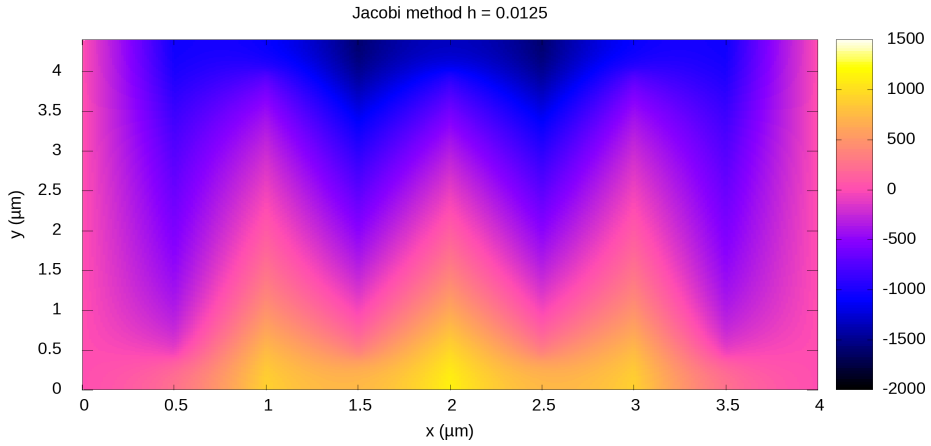


Figure 5: Jacobi

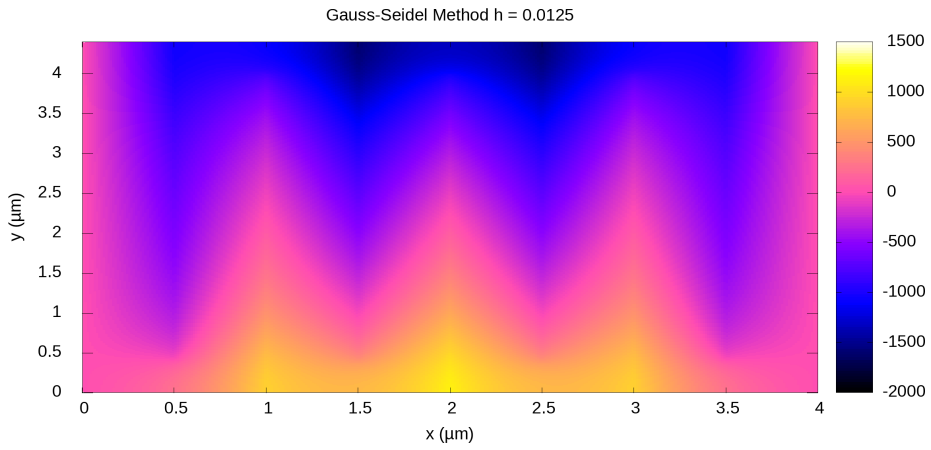


Figure 6: Gauss-Seidel

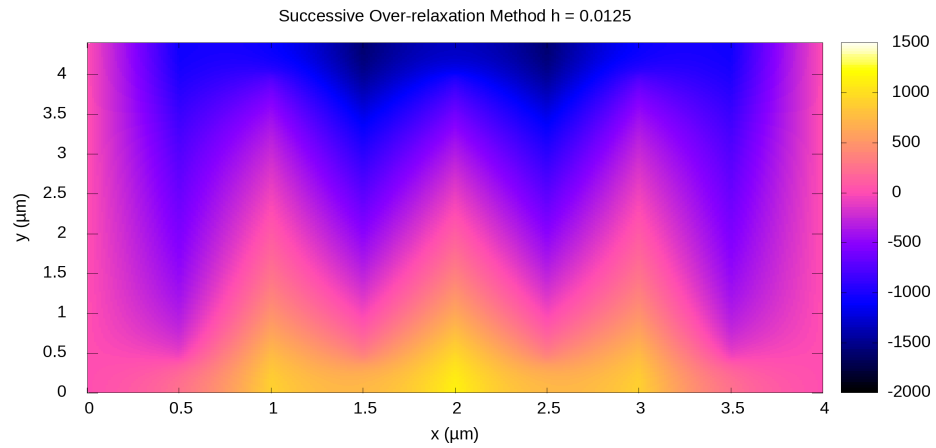


Figure 7: SOR

4.3 Comparison of Methods

Since all the iterative methods we used have given us almost identical results so we will only compare them on the basis of number of iteration required so as to determine which method is more compuationally cheap and viable

meshsize h	No. of iterations		
	Jacobi	Gauss-Seidel	S.O.R (1.9)
0.01	11878	6315	251
0.05	52047	26983	1552
0.025	182367	94927	5821
0.0125	674230	335520	22163

Table 1: Table comparing the number of iterations required for each method for a relative tolerance of 10^{-8}

meshsize h	Approximate computation time ($\pm 0.01s$)		
	Jacobi	Gauss-Seidel	S.O.R (1.9)
0.01	0.41	0.27	0.02
0.05	6.13	3.96	0.28
0.025	90.73	55.46	3.47
0.0125	1380.54	818.68	54.35

Table 2: Table comparing the computation time required for each method for a relative tolerance of 10^{-8}

Comparing all the methods for computation under identical conditions on the basis of number of iteration required for acheiving the required tolerance we can say **SOR** Method is best method among all three iterative schemes used on all grounds.

4.3.1 Successive Over Relaxation (SOR)

In this section we analyse the results obtained for different values of relaxation factor.

Optimum value of Relaxation Factor
In Successive Over Relaxation we have to choose the value of a relaxation factor which is responsible for the rate of convergence of solution. Its value can be chosen anywhere between 1 and 2 i.e. relaxation factor or $\omega \in [1, 2]$. We observed the variation of number of iterations required to reach a given tolerance with change in ω. The following graph represents the graph between number of itereations and value of ω.

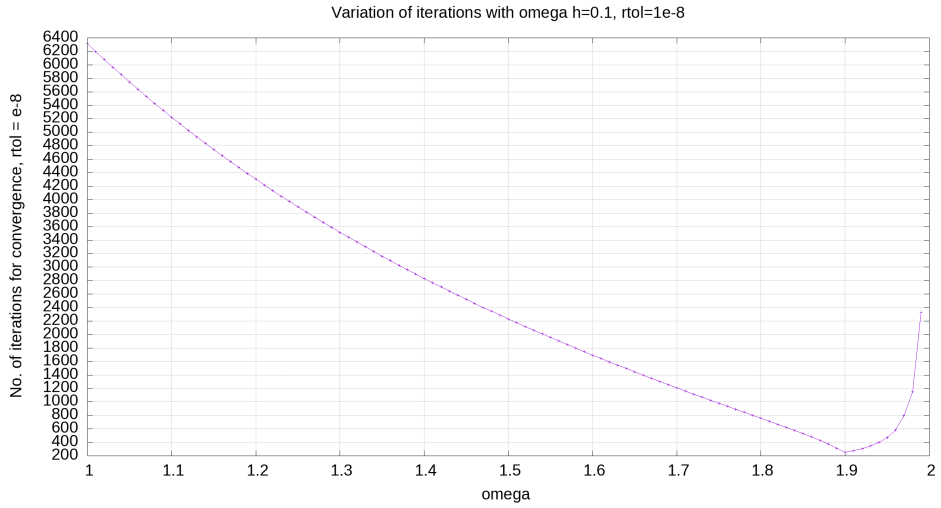


Figure 8: variation of number of iterations for convergence with relative tolerance=10^{-8} and meshsize $h = 0.1$

From the graph we can see that the most optimum value of ω is around 1.90 according to number of iterations required to reach the solution.

5 Conclusion

5.1 Result

We tried to solve the problem of interleaved capacitor using the finite difference method for solving poisson equation. We have solved the problem using three different iterative schemes. After completing this project we can say that the method of SOR is best for solving the system of linear equation after using the finite differences method. Using SOR method we were able to solve mesh of meshsize 0.0125 in just 22163 number of iteration and just $54.35 \pm 0.1sec$ seconds. Also we gained better insight and intuition after solving the problem of interleaved capacitor.

5.2 Experience

We have learnt a lot of new things during this project. We have learnt how to solve a physical problem computationally and the various processes it involves such as non-dimensionalisation, the concept of convergence etc. This project has also boosted our python, latex and gnu skills. It has also increased our fascination with power of computation methods which allow us to solve complex physical problems without going through tedious calculatoins just by following some standard methods. We also spend a good portion of time studying about theoretical aspects of different computational methods and trying to understand the concepts related to convergence, truncation error. This project has been an incredible journey for us beacause it has not only increased our theoretical , physical and compuational knowledge but it has also taught us about the importance of perseverance and patience as there were many topic that we didn't understand easily just by studying about it from one or two places and things that were not easily available in comprehendable manner for us due to advance nature of partial differential equation, sometimes we had to spend a lot of time in just trying to find a good resource. We would like to end this report by saying that we are very grateful that we had chosen such a topic that has taught us so much.

6 Contributions

Contribution of "*Akarsh Shukla*"

- In Formulation of the problem: Conversion of phyiscal problem into computaional
- In Programming: data handling
- In Plotting Graphs: complete plotting
- In Report Writing:
  - Introduction
- In Beamer:
  - Introduction
  - Conclusion

Contribution of "*Brahmanand Mishra*"

- In Formulation of the problem: non -dimensionalisation
- In Programming: sor and Gauss seidel method
- In Report Writing:
  - Abstract
  - Expectation part
  - Result part
  - Conclusion
- In Beamer:
  - Methodology
  - Truncation error
  - Finite difference method
  - Result part

Contribution of "*Shashvat Jain*"

- In Formulation: diagrams using inkscape and tikz
- In Programming: function for mesh and jacobi method
- In Report Writing:
  - Finite difference method
  - Truncation error
  - Algorithm
  - Conversion of PDE into finite difference equation(FDE).

7 References

[1] I. o. A. M. o. D. Dmitri Kuzmin. "Non-dimensionalisation and an overview of discretization techniques." (), [Online]. Available: http://www.mathematik.tu-dortmund.de/~kuzmin/cfdintro/lecture3.pdf.

[2] M. opencourseware. "Computational methods in aerospace engineering." (), [Online]. Available: http://www.mathematik.tu-dortmund.de/~kuzmin/cfdintro/lecture3.pdf.

[3] J. Thomas, *Numerical Partial Differential Equations: Finite Difference Methods*, ser. Texts in Applied Mathematics. Springer New York, 2013, ISBN: 9781489972781. [Online]. Available: https://books.google.co.in/books?id=83v1BwAAQBAJ.

[4] K. Morton and D. Mayers, *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2005, ISBN: 9781139443203. [Online]. Available: https://books.google.co.in/books?id=GW6%5C_AwAAQBAJ.

[5] D. Kuzmin. "Introduction to computational fluid dynamics." (), [Online]. Available: http://www.mathematik.tu-dortmund.de/~kuzmin/cfdintro/lecture1.pdf.

8 Appendix

8.1 Non-dimensionalisation

In this section we would like to non-dimensionalise Laplace equation. The general form of Laplace equation is:

$$\vec{\nabla}^2\varphi(\vec{r}) = 0$$

or

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

or

$$u_{xx} + u_{yy} = 0$$

where u_{xx} represents the second order partial derivative of u with respect to x. and u_{yy} represents second order partial derivative of u with respect to y. In our case $u = V$ so we need to non -dimensionalise the following equation:

$$V_{xx} + V_{yy} = 0$$

Now we will replace the dimensional variable with non-dimensional variables. Let

$$\hat{V} = \frac{V}{V_s}, \hat{x} = \frac{x}{x_s}, \hat{y} = \frac{y}{y_s}$$

here the $\hat{V}, \hat{x}, \hat{y}$ are dimensionless variables and V_s, x_s, y_s are the scaling variables having the same unit as their counterparts.

$$\frac{\partial^2 V}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial V}{\partial x}\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{\partial(\hat{V}V_s)}{\partial(\hat{x}x_s)}\right)$$

$$= \frac{V_s}{x_s}\left[\frac{\partial}{\partial(x_s\hat{x})}\left(\frac{\partial\hat{V}}{\partial\hat{x}}\right)\right]$$

$$= \frac{V_s}{x_s^2}\left[\frac{\partial^2\hat{V}}{\partial\hat{x^2}}\right]$$

similarly, we can write for y as :

$$\frac{V_s}{y_s^2}\left[\frac{\partial^2\hat{V}}{\partial\hat{y^2}}\right]$$

Now we can write our equation as:

$$= \frac{V_s}{y_s^2}\left[\frac{\partial^2\hat{V}}{\partial\hat{y^2}}\right] + \frac{V_s}{x_s^2}\left[\frac{\partial^2\hat{V}}{\partial\hat{x^2}}\right] = 0$$

So now if choose same magnitude for our scaling factor then we write our equation as:

$$= \frac{V_s}{x_s^2}\left[\frac{\partial^2\hat{V}}{\partial\hat{y^2}} + \frac{\partial^2\hat{V}}{\partial\hat{x^2}}\right] = 0$$

$$= \frac{\partial^2\hat{V}}{\partial\hat{y^2}} + \frac{\partial^2\hat{V}}{\partial\hat{x^2}} = 0$$

Hence so if we just use the same sacling factors for both x and y variable then we don't need to non-dimensionalise the Laplace equation.

8.1.1 Poisson Equation

Now since the general form of Poisson Equation is as follows:

$$\vec{\nabla}^2\varphi(\vec{r}) = f(\vec{r})$$

or

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -f(r)$$

So for our case we can write it as:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = -\frac{\rho}{\epsilon_0}$$

Now following the same procedure for LHS as done in case of Laplace equation we get:

$$\frac{V_s}{x_s^2}\left[\frac{\partial^2 \hat{V}}{\partial \hat{y^2}} + \frac{\partial^2 \hat{V}}{\partial \hat{x^2}}\right] = \frac{-\rho}{\epsilon_0}$$

or

$$\frac{\partial^2 \hat{V}}{\partial \hat{y^2}} + \frac{\partial^2 \hat{V}}{\partial \hat{x^2}} = \frac{-\rho}{\epsilon_0}\frac{x_s^2}{V_s}$$

Now we will choose the scaling factors such that the value of RHS becomes unity to ease our computation part and it will depend upon the question.

8.2 Code

8.2.1 Python Scripts

The file *poisson.py* containing all our important methods and classes.

```
from numba import jit
import numpy as np
import time as t
import os

def convert_gnuplotgrid3d(FILE):
    f = open(FILE,'r')
    prev_x = f.readline().split()[0]
    lst1= ""
    for line in f :
        new_x = line.split(' ')[0]
        if new_x!=prev_x:
            lst1+="\n"+line
            prev_x = new_x
        else:
            lst1+=line
    f.close()
    f2 =open(FILE,"w")
    f2.write(lst1)
    f2.close()

class mesh:
    def __init__(self,x=(0,1),y=(0,1),h=0.1,gtype="2D"):
        self.x_dim,self.y_dim=int((x[1]-x[0])/h +1),int((y[1]-y[0])/h +1)
        self.y_dom=np.linspace(y[0],y[1],self.y_dim)
        self.x_dom=np.linspace(x[0],x[1],self.x_dim)
        self.h = h
```

```
        self.omega = dict()
        self.gtype= gtype
        self.X,self.Y = np.meshgrid(self.x_dom,self.y_dom)
        if self.gtype == "1D":
            self.grid=np.ones((self.x_dim,),dtype="double")
        elif self.gtype == "2D":
            self.grid=np.ones((self.y_dim,self.x_dim),dtype="double")

    def dirichlet(self,x,excluded = "y"):
        if self.gtype=="2D" and len(x)==4:
            if excluded != "y":
                self.grid[0,:],self.grid[-1,:] = x[2] , x[3]
            if excluded !="x":
                self.grid[:,0],self.grid[:,-1] =  x[0] , x[1]
        elif self.gtype == "1D" and len(x)==2:
            self.grid[0],self.grid[-1] = x[0] , x[1]
        else:
            raise ValueError("Expected {0} but recieved {1} Boundary
    conditions.".format(self.gtype[0],len(x)))
    def get(self):
        return(self.grid)
    def set_loc(self,loc):
        self.loc = os.getcwd()+f"/dats/{loc}"
        try:
            os.mkdir(self.loc)
        except:
            pass
    def jacobi_poisson2d(self,p,nu=1,rtol=None):
        solve = jacobi2d(self.grid,self.h,p,rtol)
        self.u = solve[0]*nu
        self.dat = np.array([self.X.flatten('F'),self.Y.flatten('F'),self.u
    .flatten('F')],dtype =float)
        np.savetxt(f"{self.loc}/jacobi_poisson2d_{self.h}.dat",self.dat.T,
    fmt="%.20g")
        convert_gnuplotgrid3d(f"{self.loc}/jacobi_poisson2d_{self.h}.dat")
        return(solve)
    def sor_poisson2d(self,p,w,nu=1,rtol=None):
        solve = sor2dpoisson(self.grid,self.h,w,p,rtol)
        self.u = solve[0]*nu
        self.omega[w] = solve[1]
        self.dat = np.array([self.X.flatten('F'),self.Y.flatten('F'),self.u
    .flatten('F')],dtype =float)
        np.savetxt(f"{self.loc}/sor_poisson2d_{self.h}_{w}_{nu}.dat",self.
    dat.T,fmt="%.20g")
        convert_gnuplotgrid3d(f"{self.loc}/sor_poisson2d_{self.h}_{w}_{nu}.
    dat")
        return(solve)

    def save_omega(self):
        np.savetxt(f"/home/planck/Desktop/secc_project_proposal/dats/{self.
    loc}/omega_variation.dat",
        np.array([list(self.omega.keys()),list(self.omega.values())]).T)

@jit("Tuple((f8[:,:],f8))(f8[:,:],f8,f8,f8[:,:],f8)",nopython=True,cache=
    True)
def sor2dpoisson(x,h,overcf=1.9,p=None,rtol=None):
    k,m = x.shape[0],x.shape[1]
    if p is None :
        p = np.zeros(x.shape)
    if rtol is None :
        rtol = h**2
    for f in np.arange(1,1e+6,1):
        new_x= x.copy()
```

```
        for i in np.arange(0,k,1):
            for j in np.arange(1,m-1,1):
                left = new_x[i][j-1]
                right = new_x[i][j+1]
                ##Neumann 0 wrt y-axis at y_upper and y_lower bounds
                if i == k-1 :
                    up = new_x[i-1][j]
                else :
                    up = new_x[i+1][j]
                if i == 0 :
                    down = new_x[i+1][j]
                else:
                    down = new_x[i-1][j]
                new_x[i][j] = new_x[i][j] + ((up+down+right+left + h**2*p[i
    ][j])/4 - new_x[i][j]) * overcf
        er = np.abs((new_x - x )/ new_x).max()
        if er<=rtol:
            break
        else:
            x = new_x.copy()
    return(new_x,f)

@jit("Tuple((f8[:,:],f8))(f8[:,:],f8,f8[:,:],f8)",nopython=True)
def jacobi2d(x,h,p=None,rtol=None):
    k,m = x.shape[0],x.shape[1]
    if p is None :
        p = np.zeros(x.shape)
    if rtol is None :
        rtol = h**2
    for f in np.arange(1,1e+6,1):
        new_x= x.copy()
        for i in np.arange(0,k,1):
            for j in np.arange(1,m-1,1):
                left = x[i][j-1]
                right = x[i][j+1]
                ##Neumann 0 wrt y-axis at y_upper and y_lower bounds
                if i == k-1 :
                    up = x[i-1][j]
                else :
                    up = x[i+1][j]
                if i == 0 :
                    down = x[i+1][j]
                else:
                    down = x[i-1][j]
                new_x[i][j] = ((up+down+right+left + h**2*p[i][j])/4)
        er = np.abs((new_x - x) / new_x).max()
        if er<=rtol:
            break
        else:
            x = new_x.copy()
    return(new_x,f)

if __name__ == "__main__":
    pass
```

The file *cap.py* utilizing all methods and classes to solve the main problem.

```
from lib.poisson import *
import numpy as np
import time as t

'''Set constants'''
ep = 8.854e-12
us = 1
```

```
xs = 1
'''Set Domain'''
mm = mesh((0,4),(0,4.4),0.1)
h = np.diff(mm.x_dom)[0]
'''Set Dirichlet Boundary value conditions'''
'''
------------upper_y_bound-----------
lower                              upper
_x_                                _x_
bound                              bound
------------lower_y_bound-----------
'''
#lower_bound_y = np.ones((mm.x_dim,))
#upper_bound_y = np.ones((mm.x_dim,))
lower_bound_x = np.ones((mm.y_dim,))*5/us
upper_bound_x = np.ones((mm.y_dim,))*5/us
mm.dirichlet([lower_bound_x,upper_bound_x,None,None])

'''Set charge distributiion'''
distri = np.zeros(mm.grid.shape)
xr,yr = np.round_(mm.X,2),np.round_(mm.Y,2)
bool_A = ((xr==2) | (xr==1) | (xr==3)) & (yr <=4)
bool_B = np.any([xr==0.5,xr==1.5,xr==2.5,xr==3.5],axis=0) & (yr >= 0.4)
distri[bool_A] = 2*1e+5*xs**2/us
distri[bool_B] = -2*1e+5*xs**2/us

'''Create a folder where data gets stored'''
mm.set_loc("t2")

#Solve poisson
t1= t.time()
s1=mm.sor_poisson2d(distri,1.9,nu=us,rtol=1e-8)
print(s1[1])
t2= t.time()
s2=mm.sor_poisson2d(distri,1,nu=us,rtol=1e-8)
t3= t.time()
print(s2[1])
s3=mm.jacobi_poisson2d(distri,us,rtol=1e-8)
t4= t.time()
print(s3[1])

#mm.save_omega()
print(t2-t1)
print(t3-t2)
print(t4-t3)

# Variation of iterations with omega
iter = []
for i in np.arange(1,2,0.01):
    iter.append(mm.sor_poisson2d(distri,i,nu=us,rtol=1e-8)[1])
np.savetxt("omega_variation.dat",np.array([np.arange(1,2,0.01),iter]).T)
```

8.2.2 Gnuplot Scripts

For plotting variation of no of iterations with *omega*,

```
set title "Variation of iterations with omega h=0.1, rtol=1e-8"
set xlabel "omeg "
set ylabel "No. of iterations for convergence, rtol = e-8"
set terminal pngcairo
set terminal png font arial 30 size 2400,1300
set xrange [1:2]
set ytics 200
```

```
set xtics 0.1
set grid xtics,ytics
set output 'omega.png'
plot 'dats/omega_variation.dat' notitle w lp
```

The following script plots the colourmap of the potential scalar field in region Ω,

```
set title "Successive Over-relaxation Method h = 0.0125"
set xlabel "x "
set ylabel "y "
set zlabel "U (Volts)"
set terminal pngcairo
set terminal png font arial 32 size 2400,1300
set xrange [0:4]
set yrange [0:4.4]
set samples 80,88
set isosamples 80,88
set view 50,50
set pm3d at bs; set palette rgbformulae 30,31,32
unset surface
set view map
set output 'sor_map.png'
splot 'dats/dataset4_e-8/sor_poisson2d_0.0125_1_1.dat' notitle
```