HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
THE SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
ICT K67

IT4110E

SCIENTIFIC COMPUTING

APPLYING SOR TO SOLVE THE 2D LAPLACE EQUATION FOR STEADY STATE DIFFUSION

PROJECT REPORT

MEMBER LIST

Nguyễn Đình Dương	20225966	duong.nd225966@sis.hust.edu.vn
Nguyễn Mỹ Duyên	20225967	duyen.nm225967@sis.hust.edu.vn
Nguyễn Lan Nhi	20225991	nhi.nl225991@sis.hust.edu.vn
Nguyễn Phương Anh	20226011	anh.np226011@sis.hust.edu.vn
Nguyễn Trọng Minh Phương	20225992	phuong.ntm5992@sis.hust.edu.vn
Hồ Bảo Thư	20226003	thu.hb226003@sis.hust.edu.vn

Teacher: Vũ Văn Thiệu

Hà Nội — 2024

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

This report explores the numerical solution of the Steady-State Diffusion problem by transforming it into the 2D Laplace equation, which is then solved using the Successive Over-Relaxation (SOR) method. The 2D Laplace equation, a fundamental partial differential equation, finds extensive applications in fields like heat conduction, electrostatics, and fluid dynamics. The SOR method improves upon the Gauss-Seidel iterative technique by introducing a relaxation factor, ω, which accelerates convergence. Our approach involves discretizing the Laplace equation using finite difference methods, resulting in a system of linear equations that we solve iteratively. The implementation in MATLAB covers grid initialization, boundary condition application, and iterative updates until convergence. Key factors influencing the solution's efficiency and accuracy include the choice of relaxation factor and the handling of boundary conditions. The results confirm the SOR method's effectiveness in accurately solving the 2D Laplace equation for steady-state diffusion problems.

1 Approach to Solve the Problem

The general approach to solving the 2D Laplace equation numerically involves the following steps:

1. **Discretization**: Convert the continuous PDE into a discrete system using finite difference methods. This involves creating a grid over the domain and approximating the derivatives using differences between grid points.
2. **Iterative Solution**: Use an iterative method, such as the SOR method, to solve the resulting system of linear equations. The SOR method is particularly useful for large systems where direct methods are computationally expensive.
3. **Boundary Conditions**: Apply the given boundary conditions to the grid points to ensure that the solution conforms to the physical constraints of the problem.

2 Introduction to Problem

2.1 Steady State Diffusion

Diffusion refers to the net movement of a species down a concentration gradient from an area of high concentration to an area of low concentration. Importantly, diffusion can take place in any phase of matter including in solids! Diffusion processes may be divided into two types: (a) steady state and (b) non steady state. Steady state diffusion takes place at a constant rate - that is, once the process starts the number of atoms (or moles) crossing a given interface (the flux) is constant with time. Steady state (time independent) diffusion is described by Fick's first law:

$$J = -D\frac{dC}{dx}$$

Here J is the diffusion flux: the rate at which an amount of a substance passes through a surface area. D is the diffusion coefficient which is sometimes called diffusivity. It depends on the specific circumstances the diffusion is occurring in including what materials are involved and the state of the surrounding environment. Finally here C is concentration ($\frac{dC}{dx}$ is the change in concentration with respect to change in position). Therefore, Fick's first law tells us how concentration change flows over the region between two different concentrations.

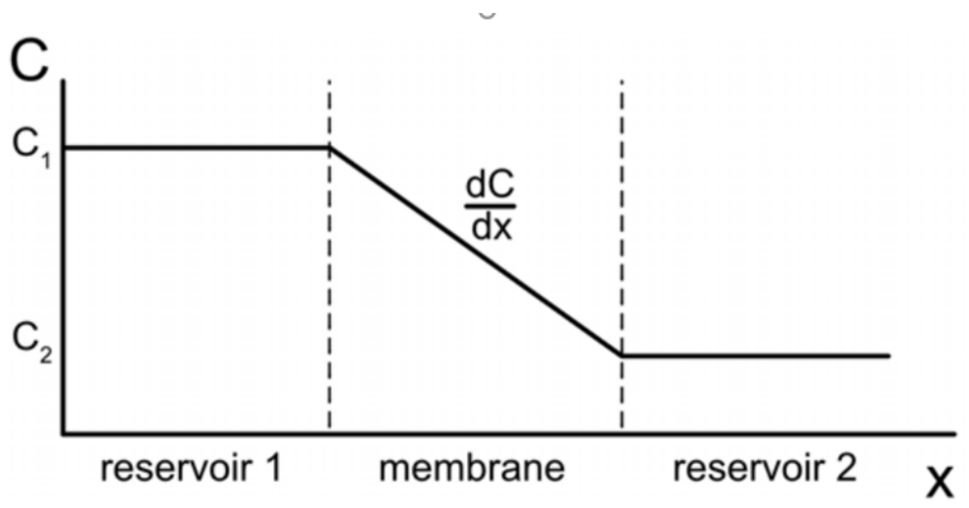

Figure 1: Concentration Profile Across a Membrane Between Two Reservoirs

2.2 Diffusion Coefficient

The diffusion coefficient (D) is a measure of how easily molecules or atoms can move through a medium. It has units of square meters per second (m^2/s) and depends on various factors, including the type of diffusing species, the medium through which diffusion occurs, and the temperature. Mathematically, it is often used in Fick's laws of diffusion to describe the rate at which particles spread.

The energy required for diffusion to occur can be thought of as an activation energy. The diffusion coefficient is:

$$D = D_0 e^{-\frac{E_a}{k_B T}}$$

Here D_0 is the maximum value of diffusivity, E_a is the activation energy, k_B is the Boltzmann constant, and T is temperature. By describing the diffusion coefficient with an Arrhenius relation, recall that we are saying that at a given temperature T, the ratio of the thermal energy given by $k_B T$ to the activation energy E_a sets the value of the diffusion coefficient and it is maximized at infinite temperature.

2.3 Non steady State Diffusion

If the concentration profile varies with respect to time, the steady-state assumption no longer holds and instead Fick's second law is used:

$$\frac{\partial C}{\partial t} = D \frac{\partial^2 C}{\partial x^2}$$

Solutions to Fick's second law are of the form:

$$\frac{C(x,t)-C_0}{C_s-C_0} = 1 - \operatorname{erf}\left(\frac{x}{2\sqrt{Dt}}\right)$$

Here C_s is the concentration of the source, C_0 is the initial concentration, $C(x,t)$ is the expression for concentration as a function of position and time, D is the diffusion coefficient and erf is the error function.

3 Transition to 2D Laplace Equation

3.1 Starting with Fick's Second Law:

In problems with 2 or more dimension, we must use the Laplacian which generate the second derivative in the equation. Hence the Fick's second law in this problem is:

$$\frac{\partial C}{\partial t} = D\nabla^2 C$$

Where:

- C is the concentration of the diffusing substance,
- t is time,
- D is the diffusion coefficient, and
- ∇^2 is the Laplacian operator.

3.2 Steady-State Condition

In steady-state diffusion, the concentration does not change over time so the time derivative term vanishes:

$$\frac{\partial C}{\partial t} = 0$$

Substituting into Fick's Second Law: When we substitute into Fick's second law we get:

$$D\nabla^2 C = 0$$

This equation represents the steady-state diffusion process where the Laplacian of the concentration

$$\nabla^2 C$$

is equal to zero.

3.3 General Form of the Laplace Equation

The resulting equation

$$\nabla^2 C = 0$$

is known as the Laplace equation. It is a second-order partial differential equation that describes systems in which the quantity of interest (in this case concentration) is in a state of equilibrium with no sources or sinks.

3.4 Extending to Two Dimensions

When considering diffusion in two dimensions, we extend the Laplace equation by adding derivatives with respect to the additional spatial dimensions (e.g. x and y). This yields the 2D Laplace equation:

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

This equation describes the steady-state distribution of concentration in a two-dimensional space.

4 Solving The Equation With SOR

The Successive Over-Relaxation (SOR) method is an iterative technique used to solve linear systems of equations, particularly those arising from discretized partial differential equations like the Laplace equation. It is an extension of the Gauss-Seidel method, incorporating a relaxation parameter to accelerate convergence.

4.1 Initialization

Choose an initial guess for the concentration C at each grid point. Set the relaxation parameter ω (typically between 1 and 2) and choose a convergence tolerance. ($\omega = 1.9$ is the best for most cases).

4.2 Iteration

Iterate through each grid point, excluding the boundary points. Update the concentration value at each grid point using the SOR formula:

$$C_{i,j}^{(k+1)} = (1-\omega)C_{i,j}^{(k)} + \frac{\omega}{4}\left(C_{i+1,j}^{(k)} + C_{i-1,j}^{(k+1)} + C_{i,j+1}^{(k)} + C_{i,j-1}^{(k+1)}\right)$$

where:

- $C_{i,j}^{(k)}$ is the concentration at grid point (i, j) in the k-th iteration.
- $C_{i+1,j}^{(k)}$, $C_{i-1,j}^{(k)}$, $C_{i,j+1}^{(k)}$, and $C_{i,j-1}^{(k)}$ are the concentrations at the neighboring grid points.
- ω is the relaxation parameter.

4.3 Convergence Check

Compute the change in concentration values between the current and previous iterations. Check if the maximum change falls below the specified tolerance level. If so, stop iterating; otherwise, continue.

4.4 Solution Extraction

After convergence is achieved, the concentration values at each grid point represent the numerical solution to the Laplace equation.

5 Application

5.1 General Problem

In this project, we explore solutions to the two-dimensional Laplace equation, a fundamental differential equation in physics and engineering. This equation describes the distribution of a scalar field (such as electric potential or temperature) in a homogeneous and steady state. The objective is to determine the stable distribution of this field within a rectangular domain given specific boundary values.

5.1.1 Problem Characteristics

- **Computational Domain**: A two-dimensional grid $L \times L$ with L is entered by the user.
- **Boundary Conditions**: The potential ϕ is set to 100 at the vertical boundaries and 0 at the horizontal boundaries.

5.1.2 Problem Challenges

Solving the two-dimensional Laplace equation is not trivial, especially when it requires rapid and accurate convergence. Specific challenges in this problem include:

- **Computational Efficiency**: Optimizing the number of iterations required to achieve convergence while maintaining necessary accuracy.
- **Algorithm Stability**: Ensuring that the Successive Over-Relaxation (SOR) method operates stably with a relaxation factor $\omega = 1.9$ and does not lead to divergence.
- **Visualization of Results**: Presenting the results visually to analyze the potential distribution across the grid.

5.1.3 MATLAB Code to solve the problem

```matlab
function main()
    % Prompt the user to input the grid size
    L = input('Enter the grid size L: ');

    % Parameters
    omega = 1.9; % Relaxation factor
    tolerance = 1e-6; % Convergence threshold
    max_iter = 10000; % Maximum number of iterations

    % Initialize the grid with zeros
    phi = zeros(L, L);

    % Boundary conditions (example)
    phi(:, 1) = 100; % Left boundary
    phi(:, end) = 100; % Right boundary
    phi(1, :) = 0; % Bottom boundary
    phi(end, :) = 0; % Top boundary

    % Solve the Laplace equation using the SOR method
    [phi, iter] = solveSOR(phi, omega, tolerance, max_iter);

    % Display and save the results
    displayResult(phi, iter, L);
end
```

Listing 1: Main function to solve the Laplace equation

```
function [phi, iter] = solveSOR(phi, omega, tolerance, max_iter)
    % SOR iteration
    for iter = 1:max_iter
        phi_old = phi; % Save the old values to check for convergence

        % Update the internal points using the SOR method
        for i = 2:size(phi, 1)-1
            for j = 2:size(phi, 2)-1
                phi(i, j) = (1-omega) * phi(i, j) + omega/4 * (phi(i+1, j
                    ) + phi(i-1, j) + phi(i, j+1) + phi(i, j-1));
            end
        end

        % Check for convergence
        if max(max(abs(phi - phi_old))) < tolerance
            disp(['Converged after ', num2str(iter), ' iterations.']);
            break;
        end
    end

    % Check if the method did not converge
    if iter == max_iter
        disp(['Did not converge after ', num2str(max_iter), ' iterations.
            ']);
    end
end
```

Listing 2: Function to solve the Laplace equation using SOR method

```
figure;
    subplot(1, 2, 1);
    contourf(phi, 20); % Increase contour levels for better display
    colorbar;
    title(['Contour Plot: Converged after ', num2str(iter), ' iterations'
        ]);
    xlabel('x');
    ylabel('y');

function displayResult(phi, iter, L)
    % Display results
    figure;
    subplot(1, 2, 1);
    contourf(phi, 20); % Increase contour levels for better display
    colorbar;
    title(['Contour Plot: Converged after ', num2str(iter), ' iterations'
        ]);
    xlabel('x');
    ylabel('y');

    subplot(1, 2, 2);
    surf(phi);
    colorbar;
    title('Surface Plot');
    xlabel('x');
    ylabel('y');
    zlabel('\phi');

    % Save the plot as an image
    saveas(gcf, 'laplace_solution_plots.png');
end
```

Listing 3: Function to display and save the results

5.1.4 Results

The program successfully computed and converged after a designated number of iterations, demonstrating the stability and efficiency of the SOR method in solving the Laplace equation. The results were visualized through two types of plots:

- **Contour Plot**: Displays the potential distribution across the computational domain, allowing for the identification of areas with the same potential.
- **Surface Plot**: Provides a three-dimensional view of how the potential varies across the entire domain, offering a deeper understanding of the potential landscape.

The outcomes of this problem not only confirm the capabilities of the SOR method but also provide significant insights for practical applications such as modeling electric fields or heat distribution.

5.2 Specific Problem Statement

Given a rectangular metal plate with dimensions $L_x \times L_y$, we aim to determine the steady-state temperature distribution across the plate using the Successive Over-Relaxation (SOR) method. The problem involves setting specific boundary conditions, handling random holes, incorporating singular temperature points, and adding dynamic heat sources. The main goal is to achieve a stable temperature distribution across the metal plate.

5.2.1 Computational Domain

- **Rectangular Metal Plate:** The metal plate has dimensions $L_x \times L_y$, where L_x is the length in the x-direction and L_y is the length in the y-direction. The values for L_x and L_y are provided by the user.
- **The computational grid is initialized with zero temperatures.**

5.2.2 Boundary Conditions

- The temperature at the boundaries of the plate is specified by the user. These boundary conditions are defined as follows:
  - **Left Boundary:** The temperature varies linearly from $T_{\text{left, bottom}}$ at the bottom to $T_{\text{left, top}}$ at the top.
  - **Right Boundary:** The temperature varies linearly from $T_{\text{right, bottom}}$ at the bottom to $T_{\text{right, top}}$ at the top.
  - **Bottom Boundary:** The temperature varies linearly from $T_{\text{bottom, left}}$ at the left to $T_{\text{bottom, right}}$ at the right.
  - **Top Boundary:** The temperature varies linearly from $T_{\text{top, left}}$ at the left to $T_{\text{top, right}}$ at the right.

5.2.3 Singular Temperature Points

- The user can specify certain points within the plate where the temperature is fixed at specific values. These are known as singular temperature points and are defined by their coordinates and temperature values.

5.2.4 Objectives

- **Determine the Steady-State Temperature Distribution:** Calculate the temperature at each point on the grid until the system reaches a steady state.
- **Analyze Convergence:** Ensure that the SOR method converges efficiently to the solution, achieving the required tolerance.
- **Visualize the Temperature Distribution:** Generate contour and surface plots to visually represent the temperature distribution across the plate.

5.2.5 Challenges

- **Handling Boundary Conditions:** Accurately implementing the user-defined boundary conditions to ensure correct temperature distribution at the edges of the plate.
- **Stability of SOR Method:** Ensuring that the SOR method with relaxation factor $\omega = 1.9$ remains stable and convergent, and does not lead to divergence.

5.2.6 Results

The program calculates the stable temperature distribution over the metal plate and visualizes the results. The results include:

- **Contour Plot:** Displays the potential distribution across the computational domain, allowing for the identification of areas with the same potential.
- **Surface Plot:** Provides a three-dimensional view of how the potential varies across the entire domain, offering a deeper understanding of the potential landscape.

5.2.7 MATLAB Code

```matlab
function main()
    % Prompt the user to input grid sizes
    Lx = input('Enter the grid size in x-direction (Lx): ');
    Ly = input('Enter the grid size in y-direction (Ly): ');

    % Parameters
    omega = 1.9; % Relaxation factor
    tolerance = 1e-6; % Convergence threshold
    max_iter = 10000; % Maximum number of iterations

    % Initialize temperature grid with zeros
    T = zeros(Lx, Ly);

    % Input boundary temperature values from the user
    T = inputBoundaryTemperatures(T, Lx, Ly);

    % Input multiple singular temperature points from the user
    [T, singular_points] = inputSingularTemperaturePoints(T, Lx, Ly);

    % Display initial temperature grid
    displayInitialGrid(T, Lx, Ly);

    % Solve the Laplace equation using SOR method
    [T, iter] = solveSOR(T, omega, tolerance, max_iter);

    % Display and save the results
    displayResult(T, iter, Lx, Ly, singular_points);
end
```

Listing 4: Main function to solve the Laplace equation

```matlab

function T = inputBoundaryTemperatures(T, Lx, Ly)
    disp('Enter boundary temperature values:');
    T_left_bottom = input('Left boundary (bottom): ');
    T_left_top = input('Left boundary (top): ');
    T_right_bottom = input('Right boundary (bottom): ');
    T_right_top = input('Right boundary (top): ');
    T_bottom_left = input('Bottom boundary (left): ');
    T_bottom_right = input('Bottom boundary (right): ');
    T_top_left = input('Top boundary (left): ');
    T_top_right = input('Top boundary (right): ');

    T(:, 1) = linspace(T_left_bottom, T_left_top, Lx)'; % Left boundary
    T(:, end) = linspace(T_right_bottom, T_right_top, Lx)'; % Right
        boundary
    T(1, :) = linspace(T_bottom_left, T_bottom_right, Ly); % Bottom
        boundary
    T(end, :) = linspace(T_top_left, T_top_right, Ly); % Top boundary
    % Display the temperature grid after applying boundary conditions
    disp('Temperature grid after applying boundary conditions:');
    disp(T);
end
```

Listing 5: Function to input boundary temperature values

```
function [T, singular_points] = inputSingularTemperaturePoints(T, Lx, Ly)
    singular_points = [];
    disp('Enter the number of singular temperature points:');
    num_temp_points = input('Number of points: ');
    for k = 1:num_temp_points
        disp(['Enter coordinates and temperature value for singular point
            ', num2str(k), ':']);
        x_temp = input('x-coordinate: ');
        y_temp = input('y-coordinate: ');
        temp = input('Temperature value: ');
        if x_temp >= 1 && x_temp <= Lx && y_temp >= 1 && y_temp <= Ly
            T(y_temp, x_temp) = temp; % Correct the order of indexing
            singular_points = [singular_points; x_temp, y_temp, temp];
        else
            disp('Error: Invalid temperature coordinates.');
        end
    end
end
```

Listing 6: Function to input singular temperature points

```
function displayInitialGrid(T, Lx, Ly)
    figure;
    contourf(T, 20, 'LineColor', 'none'); % Increase contour levels for
        better visualization
    colormap jet; % Use jet colormap for a visually appealing color
        scheme
    colorbar;
    title('Initial Temperature Grid');
    xlabel('x');
    ylabel('y');
    axis equal; % Ensure equal scaling of x and y axes
end
```

Listing 7: Function to display the initial temperature grid

```
function [T, iter] = solveSOR(T, omega, tolerance, max_iter)
    for iter = 1:max_iter
        T_old = T; % Save old temperature values to check convergence
        % Update temperature using SOR method
        for i = 2:size(T, 1)-1
            for j = 2:size(T, 2)-1
                T(i, j) = (1-omega) * T(i, j) + omega/4 * (T(i+1, j) + T(i-1, j) + T(i, j+1) + T(i, j-1));
            end
        end

        % Display updated temperature grid every 100 iterations for checking
        if mod(iter, 100) == 0
            disp(['Temperature grid after ', num2str(iter), ' iterations:']);
            disp(T);
        end

        % Check convergence
        if max(max(abs(T - T_old))) < tolerance
            disp(['Converged after ', num2str(iter), ' iterations.']);
            break;
        end
    end
    if iter == max_iter
        disp(['Did not converge after ', num2str(max_iter), ' iterations.']);
    end
end
```

Listing 8: Function to solve the Laplace equation using SOR method

```
function displayResult(T, iter, Lx, Ly, singular_points)
    % Display the results
    figure;
    % Temperature contour plot
    contourf(T, 20, 'LineColor', 'none'); % Increase contour levels for better visualization
    colormap jet; % Use jet colormap for a visually appealing color scheme
    colorbar;
    title(['Temperature Contour Plot: Converged after ', num2str(iter), ' iterations']);
    xlabel('x');
    ylabel('y');
    axis equal; % Ensure equal scaling of x and y axes

    % Highlight singular temperature points
    hold on;
    scatter(singular_points(:, 1), singular_points(:, 2), 30, 'k', 'filled'); % Smaller black circles
    for i = 1:size(singular_points, 1)
        text(singular_points(i, 1), singular_points(i, 2), sprintf(' %.1f', singular_points(i, 3)), ...
            'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right'
```

```
                   , 'Color', 'black');
    end
    hold off;

    % Save the plot as an image
    saveas(gcf, 'heat_solution_plot.png');
end
```

Listing 9: Function to display the results

5.2.8 Example with User Input Data

For a grid size of 20 × 20, boundary temperatures varying linearly, and a singular temperature point at coordinates (10, 10) with a temperature value of 400, the MATLAB code will calculate the steady-state temperature distribution and display the results. The initial temperature grid and the final temperature contour plot will be visualized as shown in the figures below.

- **Grid size:** 20 × 20
- **Boundary temperatures:**
  - Left boundary (bottom to top): 200 to 400
  - Right boundary (bottom to top): 300 to 500
  - Bottom boundary (left to right): 150 to 300
  - Top boundary (left to right): 350 to 600
- **Singular temperature points:**
  - Coordinate (10, 10) with temperature value: 500

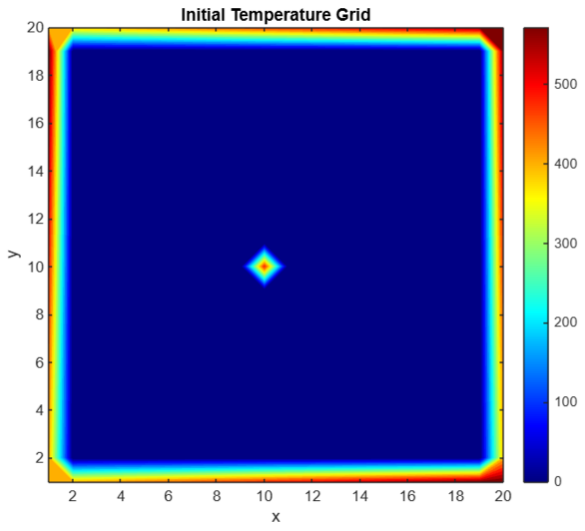

Figure 2: Initial Temperature Grid

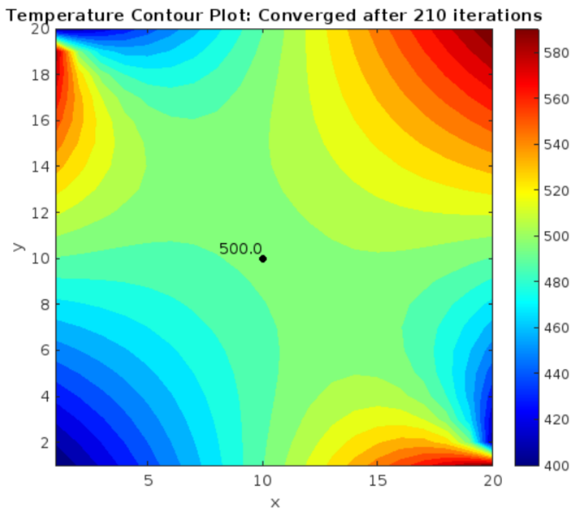

Figure 3: Temperature Contour Plot

6 Conclusion

Through the meticulous application of the SOR method, the Laplace equation governing steady-state diffusion can be effectively solved, offering valuable insights into concentration dynamics within the system. This iterative approach facilitates computational efficiency and enhances the understanding of diffusive phenomena in diverse scientific and engineering contexts.

7 References

- “Laplace’s Equation in 2D” – LibreTexts: https://math.libretexts.org/Bookshelves/Differential_Equations/Introduction_to_Partial_Differential_Equations_(Herman)/02%3A_Second_Order_Partial_Differential_Equations/2.05%3A_Laplaces_Equation_in_2D
- “Two Dimension Laplace Equation” – StudySmarter: https://www.studysmarter.co.uk/explanations/physics/electromagnetism/two-dimensional-laplace-equation/#:~:text=The%20two%2Ddimensional%20Laplace%20equationin%20a%20two%2Ddimensional%20environment.
- “Steady State Diffusion Diffusion Coefficient Fick’s Second Law” – LibreTexts: https://chem.libretexts.org/Bookshelves/Inorganic_Chemistry/Introduction_to_Solid_State_Chemistry/06%3A_Recitations/6.26%3A_Steady_State_Diffusion_Diffusion_Coefficient_Ficks_Second_Law
- “Successive over-relaxation method” – ESE Jupyter Material: https://primer-computational-ma
github.io/book/c_mathematics/numerical_methods/6_Solving_PDEs_SOR.html