Equilibrium Temperature Distributions using Linear Algebra

Ali Ehab Marwan Basem SalahDin Rezk

Fall 2023

Zewail City - University of Science and Technology

Outline

Introduction

Introduction

- The heat equation and Laplace's equation [1].
- Objective: Study equilibrium temperature distributions.
- Numerical solutions using Linear Algebra.

Problem

Problem Statement

- Given temperature on boundaries of a plate.
- Determine temperature inside the plate.
- Solve steady-state heat equation ($\Delta u = 0$) numerically.

Solution

Solution Overview

- Discretize problem on an $n \times n$ grid.
- Apply mean-value property of Laplace's equation [2].
- Formulate system of linear equations.
- Solve numerically using inverse matrix or Jacobi iteration.

Inverse Matrix Method

- Rewrite system of equations using matrix notation.
- Calculate inverse of $(I - M)$.
- Obtain solution vector $\mathbf{t}$.

Jacobi Iteration Method

- Iteratively update temperature values.
- Convergence criteria: $\|\mathbf{t}_{\text{new}} - \mathbf{t}\| < \text{tol}$.
- Provide initial guess, tolerance, and maximum iterations.

Implementation

Implementation Details

- Code implemented in Python using NumPy [3].
- Version control with Git and hosted on GitHub.
- Modular code structure for clarity.

Code Structure

- `main.py`: Main script for running the program.
- `matrix_solvers.py`: Functions for solving the system of equations.
- `temperature_solver.py`: Functions for solving the heat equation.
- `plotting.py`: Functions for plotting the results.

Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Matrix Solvers

```
def calculate_inverse_solution(A, b):
    A_inverse = np.linalg.inv(A)
    x = np.dot(A_inverse, b)
    return x
```

```python
def jacobi_iteration(A, b, x0=None, tol=1e-6, max_iter=1000):
    n = len(b)
    x = x0 if x0 is not None else np.zeros(n)
    z_iter = np.zeros((max_iter, n))

    for k in range(max_iter):
        x_new = np.zeros_like(x)

        for i in range(n):
            sigma = np.dot(A[i, :i], x[:i]) + np.dot(A[i, i + 1 :],
                x[i + 1 :])
            x_new[i] = (b[i] - sigma) / A[i, i]
            z_iter[k] = x_new

        if np.linalg.norm(x_new - x) < tol:
            return x_new, k + 1, z_iter[:k + 1]

        x = x_new

    raise ValueError(
```

Temperature Solver

```
def generate_coefficient_matrix(size, left_temp, up_temp,
    right_temp, down_temp):
    matrix = np.zeros((size * size, size * size))
    rhs_vector = np.zeros(size * size)
    for row in range(size):
        for col in range(size):
            point_num = size * row + col
            if col - 1 >= 0:
                matrix[point_num][point_num - 1] = 1
            else:
                rhs_vector[point_num] += left_temp
            if row - 1 >= 0:
                matrix[point_num][point_num - size] = 1
```

```python
            else:
                rhs_vector[point_num] += up_temp
            if col + 1 < size:
                matrix[point_num][point_num + 1] = 1
            else:
                rhs_vector[point_num] += right_temp
            if row + 1 < size:
                matrix[point_num][point_num + size] = 1
            else:
                rhs_vector[point_num] += down_temp
    return matrix, rhs_vector
```

```
        else:
            rhs_vector[point_num] += up_temp
        if col + 1 < size:
            matrix[point_num][point_num + 1] = 1
        else:
            rhs_vector[point_num] += right_temp
        if row + 1 < size:
            matrix[point_num][point_num + size] = 1
        else:
            rhs_vector[point_num] += down_temp
    return matrix, rhs_vector
```

```python
def solve_temperature_equation(size, left_temp, up_temp, right_temp, down_temp):
    coefficient_matrix, rhs_vector = generate_coefficient_matrix(
        size, left_temp, up_temp, right_temp, down_temp
    )
    temperature_solution = calculate_inverse_solution(
        4 * np.identity(size * size) - coefficient_matrix, rhs_vector
    )
    return temperature_solution
```

Plotting

```python
def plot_temperature_distribution_2d(size, temperature):
    plt.imshow(
        temperature.reshape(size, size), cmap="RdYlBu_r",
            interpolation="gaussian"
    )
    plt.colorbar()
    plt.contour(temperature.reshape(size, size), cmap="hot")
    plt.title("Temperature Distribution")
    plt.xlabel("Column Index")
    plt.ylabel("Row Index")
    plt.show()
```

```python
def plot_temperature_distribution_3d(size, temperature):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    x = np.arange(0, size, 1)
    y = np.arange(0, size, 1)
    X, Y = np.meshgrid(x, y)
    ax.plot_surface(X, Y, temperature.reshape(size, size),
        cmap="RdYlBu_r", alpha=0.75)
    ax.contour(X, Y, temperature.reshape(size, size), cmap="hot")
    ax.set_title("Temperature Distribution")
    ax.set_xlabel("Column Index")
    ax.set_ylabel("Row Index")
    ax.set_zlabel("Temperature")
    plt.show()
```

Results

Numerical Results

- Tested on various boundary conditions.
- Visualized temperature distributions in 2D and 3D.

Results: Example 1

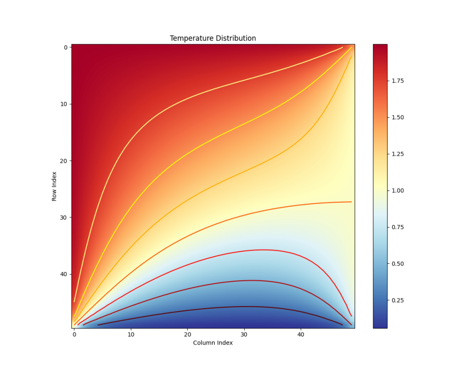

Figure 1: Boundaries $T_{\text{left}} = 2, T_{\text{up}} = 2, T_{\text{right}} = 1, T_{\text{down}} = 0$

Results: Example 1

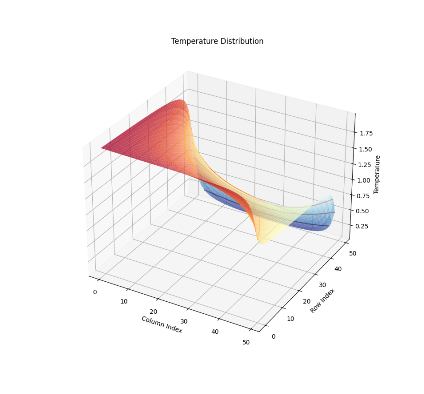

Figure 2: Boundaries $T_{\text{left}} = 2, T_{\text{up}} = 2, T_{\text{right}} = 1, T_{\text{down}} = 0$

Results: Example 2

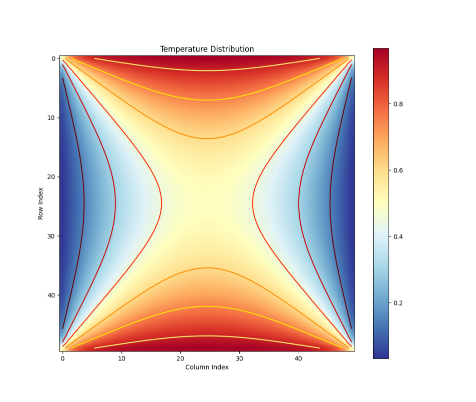

Figure 3: Boundaries $T_{\text{left}} = 0, T_{\text{up}} = 1, T_{\text{right}} = 0, T_{\text{down}} = 1$

Results: Example 2

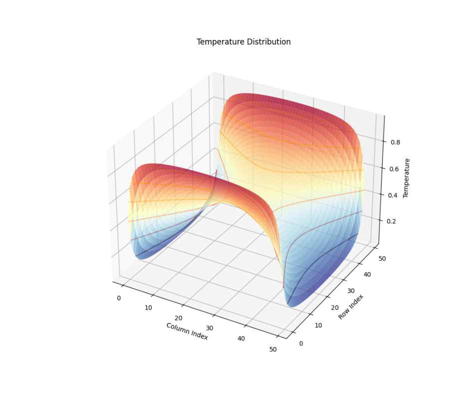

Figure 4: Boundaries $T_{\text{left}} = 0, T_{\text{up}} = 1, T_{\text{right}} = 0, T_{\text{down}} = 1$

Conclusion

Conclusion

- Linear algebra techniques for solving heat distribution problems.
- Insights gained from numerical solutions.
- Future work: Extend to more complex geometries.

Complementary Resources

- Code Repository: `https://github.com/salastro/etd-la`
- Iterations Video: `https://youtu.be/Zn6hnecikcc`

References

[1] E. M. Stein and R. Shakarchi, *Princeton Lectures in Analysis*, ser. 4 Vols. Princeton, NJ: Princeton University Press, 2003, vol. 1.

[2] H. Anton and C. Rorres, *Elementary Linear Algebra with Supplemental Applications*. Johanneshov: TPB, 2011.

[3] A. Gandhi, F. Kalkin, and K. Kainth, *Equilibrium Temperature Project*, Apr. 2022. [Online]. Available: `https://github.com/anshgandhi4/equilibrium_temperature_project`.