Project 9: Partial Differential Equations: Initial and Boundary Value Problems

Hieu Bui

November 25, 2022

1 Introduction

Real life phenomena often present themselves as functions of many variables. This project aims to elucidate on how to solve such multi-variable problems. For example, heat transfer across a plate depends on 2 spatial variable; we have 4 boundary conditions. Instead of a three point stencil, all the immediate points affect the point of interest. In order to solve the problem, we will need to use Gaussian-elimination, iterative methods, and discrete differentiation altogether.

2 Methods

2.1 Laplacian Equation

Laplacian equation represents the divergence of the gradient, which is in the form of second order partial differential equation whose sum is 0.

$$\nabla^2 f = \frac{\nabla^2 f}{\nabla x^2} + \frac{\nabla^2 f}{\nabla y^2} = 0$$

We can pick any of the two partial terms and treat it as an isolated boundary value problem. We can set up a tridiagonal matrix system by doing so. However, we will need to discretize the equation first; we will use the central difference for that. Let's say we choose to isolate the boundary value problem along the vertical or j direction; the central difference scheme requires us to move along the points in the j direction. Therefore, we will aim to keep any j changes in the left hand side.

$$
\begin{gathered}
\frac{\nabla^2 f}{\nabla x^2} + \frac{\nabla^2 f}{\nabla y^2} = 0 \\
\implies \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{\Delta y^2} = -\frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{\Delta x^2} \\
\implies f_{i,j-1} - 2f_{i,j} + f_{i,j+1} = -\frac{\Delta y^2 f_{i-1,j}}{\Delta x^2} + \frac{2\Delta y^2 f_{i,j}}{\Delta x^2} - \frac{\Delta y^2 f_{i+1,j}}{\Delta x^2} \\
\implies \Delta x^2 (f_{i,j-1} - 2f_{i,j} + f_{i,j+1}) - 2\Delta y^2 f_{i,j} = -\Delta y^2 (f_{i-1,j} + f_{i+1,j}) \\
\implies -\frac{\Delta x^2}{\Delta y^2}(f_{i,j-1} - 2f_{i,j} + f_{i,j+1}) + 2f_{i,j} = f_{i-1,j} + f_{i+1,j} \\
\implies -\frac{\Delta x^2}{\Delta y^2}(f_{i,j-1} + f_{i,j+1}) + \frac{\Delta x^2}{\Delta y^2} 2f_{i,j} + 2f_{i,j} = f_{i-1,j} + f_{i+1,j} \\
\implies -\frac{\Delta x^2}{\Delta y^2}(f_{i,j-1} + f_{i,j+1}) + f_{i,j}\frac{\Delta x^2 2 + \Delta y^2 2}{\Delta y^2} = f_{i-1,j} + f_{i+1,j} \\
\implies -\frac{\Delta x^2}{\Delta y^2}(f_{i,j-1}) + \frac{\Delta x^2 2 + \Delta y^2 2}{\Delta y^2} f_{i,j} - \frac{\Delta x^2}{\Delta y^2}(f_{i,j+1}) = f_{i-1,j} + f_{i+1,j}
\end{gathered}
$$

The final equation gives us the boundary value problem in the vertical direction; in other words, we can solve for all the points from the bottom boundary to the top boundary, which graphically represents a vertical line. If we extract the coefficients on the left hand side, we will have the coefficient matrix for the tridiagonal system matrix.

$$
\begin{bmatrix}
1 & 0 & 0 & ... \\
\beta & \alpha & 0 & ... \\
\alpha & \beta & \alpha & ... \\
0 & 0 & ... & 1
\end{bmatrix}
$$

where $\alpha = -\frac{\Delta x^2}{\Delta y^2}$, $\beta = \frac{\Delta x^2 2 + \Delta y^2 2}{\Delta y^2}$.

With the tridiagonal system set up, we proceed to march across the other direction, which is horizontal or i. We will further improve the results by using Successive-Over-Relaxation method or SOR for short. The SOR is an improved version of Gauss-Seidel, where we weighted the old and updated results. If the weight factor is between 1 and 2, then we have over relaxation. This over relaxation or weighted average scheme will allow us to converge towards the answer faster. In a graphical sense, we solve for a vertical line, which is a single variable boundary value problem. We then move across the other dimension using the most updated information about the vertical lines. Once the march is completed, we continue to iterate with an iterative method (SOR) until we are within a certain tolerance. We can also do the same thing if we choose to solve for the horizontal line instead (left boundary to right boundary). However, we will have a tridiagonal system with respect to x or i instead of the vertical direction. After which, we sweep from bottom to top. The SOR implementation remains the same.

2.2 Inviscid Subsonic Flow about a Thin Symmetric Airfoil

The idea for solving this problem is the same as the normal Laplacian problem; however, the boundary conditions are no longer constants. We will have first derivative as boundary conditions due to fluid mechanics. Therefore, we will need to solve for the boundary conditions before we can proceed to apply the same process we did with Laplacian. In order to solve for the boundary condition, we will discretize the first derivative and use given conditions to solve for a fictitious point. After obtaining a fictitious point, we can substitute it back into the discrete first derivative previously derived. The result will be the boundary condition. After this process, solving is the same as Laplacian with extra constants.

2.3 Bending of Square Plate

The bending of square plate problem is similar to Laplacian in the solving process; however, the problem is in the form of fourth order derivative instead of second order derivative.

3 Programming

3.1 Laplacian Equation

```
Inputs: u, $\alpha$, $\beta$, $\omega$
Outputs: u, processing time, corrections, iterations
while within tolerance $->$ perform iterations
    uold = unew
    solve for isolated boundary value problem (vertical or horizontal line)
        for start boundary:end boundary
        update resultant array $\vec{d}$
        THOMAS3
        weighted value for SOR
    compute corrections and iterations
end SOR loop
u = unew
```

4 Results

Performing SOR from left to right has a smoother correction curve compared to bottom up. With the given conditions in problem one, gradient of divergence of u is most intense in the middle of the bottom boundary; u diverges radially.

5 Discussion

Numerical solution for solving multivariable PDEs involve solving for multiple isolated boundary value problems, We need to discretize the governing equation beforehand, in which we could either have a tridiagonal system or pentadiagonal system. We can solve those matrix systems using Gaussian elimination.

After which, we can march across a direction of choosing. Once the march is completed, we iteratively repeat the process until we converge to an answer. Sometimes, the problem requires extra mathematical manipulation to kick start the process such as a non-constant boundary condition. All in all, once we have the boundary conditions and a matrix system; we can solve for the PDE.

The code took a significant amount of time to compute as I had everything in the for loop; however, the processing time drop after I take a few elements outside the for loop such as u=unew and uold=unew. This way, the program does not have to initialize the matrices repeatedly in the for loop. Another potential avenue of optimization would be transforming the associated matrices in the loops into 1D arrays rather than 2D matrices.