# PHY 180 - Computational Physics - Spring 2023

**Project 6: Solving the Laplace Equation**
Due: Friday, March 24th
Trevin Detwiler

## 1 Introduction

The Laplace equation is one of the first partial differential equations introduced to a student. While analytical solutions can be obtained, it can often be faster and easier to simply compute this solution. The potential around a parallel plate capacitor is one example that follows the Laplace equation. We will investigate this problem using three computational methods of solving the Laplace equation.

## 2 Modifications to the Code

To start, the `point-3D-fast.f` program was translated into Fortran 90 and reduced to a 2D problem in the file named `potential2D.f90`. The `continue` and `goto` lines were replaced with proper `do` loops with `exit` conditions. The program is also split into the subroutines `initialize` and `calculate` to handle the appropriate parts of the program.

The `initialize` subroutine prepares the arrays `pot0` and `pot1`, which were also changed to be allocatable arrays. The subroutine reads in the grid size, `grid`, the scale of the grid, `h`, and the max number of iterations, `nmax`. The arrays are allocated symmetrically, that is, running from $-$`grid`/2 to `grid`/2 in both dimensions so that the size of the grid is `grid`$\times$`grid`. The position of the parallel plates are determined with respect to the grid size so that their width is 1/10th of the grid size and their separation is half of their width. `pot0` is initialized to 0 except on the parallel plates, where the potential is set to $+1$ on one plate and $-1$ on the other.

The calculation of the potential goes back and forth between `pot0` and `pot1` in a loop over `niter`. First, `pot1` is updated using `pot0`, and then the reverse of this. Thus, the final number of iterations is actually 2*`niter`. The `calculate` subroutine handles these identically. To start, `error` is initialized to 0 and the relaxation method is applied to each point in the grid. If the point is on the parallel plate, the potential is reset to $\pm 1$ respectively. Finally, the error is calculated at each point and accumulates in the `error` variable. The loop ends when `error` is appropriately small, at which point the program writes the potential data to the output file.

The adaptations for the Gauss-Seidel method and the simultaneous over-relaxation (SOR) method were fairly straightforward. The Gauss-Seidel method in the `potential2D_GS.f90` file allows us to replace the arrays `pot0` and `pot1` with a single array, `pot`. In the `calculate` subroutine, the variable `pot_old` is then used to hold the previous potential value for proper error calculation. The main loop then only needs to call `calculate` once per iteration instead of twice, as we had to in `potential2D.f90`. The SOR method in the `potential2D_SOR.f90` file further adapts the `potential2D_GS.f90` file. The over-relaxation parameter `alpha` is calculated using `grid` in the `initialize` subroutine. The potential is updated using the Gauss-Seidel method first, then the SOR method is applied, making use of `pot_old`.

## 3 Results

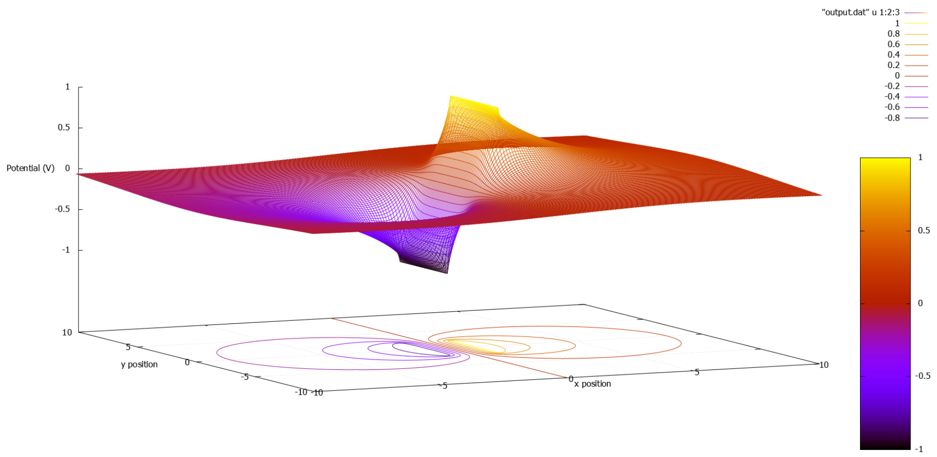

Figure 1: Potential for a parallel plate capacitor on a 400 × 400 grid with a grid scale of `h` $= 0.1$. This plot is restricted to $[-10, 10] \times [-10, 10]$.

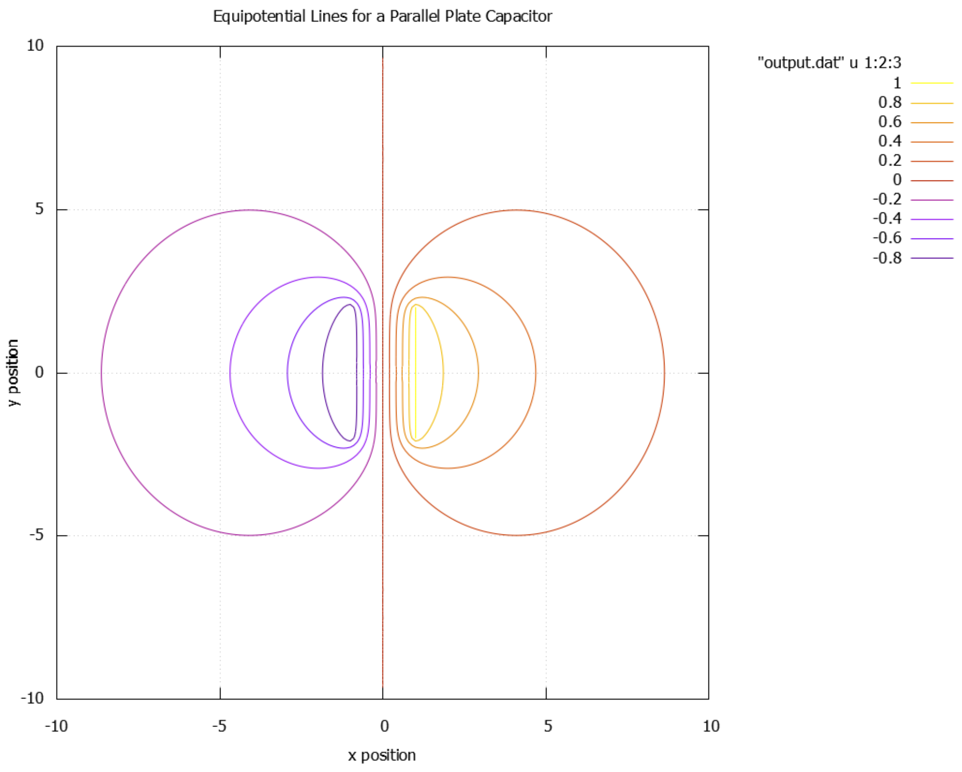

Figure 2: The equipotential lines around the parallel plate capacitor.

| Method | `niter` | Time (s) |
|---|---|---|
| Relaxation | 142996 | 243.297 |
| Gauss-Seidel | 75883 | 119.203 |
| SOR | 1306 | 3.484 |

Table 1: CPU time and iterations for each method

## 4 Conclusions

As can be seen in Figures 1 and 2, our program produces a high resolution plot of the potential with the expected equipotential lines. The Gauss-Seidel and SOR methods also improve the efficiency of the program on the orders described in Giordano and Nakanishi. The number of iterations reduced by half for the Gauss-Seidel method, and is comparable to $\mathcal{O}(L)$, where $L = \texttt{grid}$.