Project Part 2 Writeup

Sasha Yousefi

December 7, 2021

1 Introduction

In this project, we developed a program to solve the steady state heat equation in C++. We consider the scenario where we are transferring hot fluid within a pipe (temp T_h) and using an exterior series of cold jets (temp T_c) to keep the outer portion of the pipe cool. We then use our implemented algorithm to find the mean temperature within the pipe walls. The solution domain for our system is discretized into a series of grid points and boundary conditions at exposed surfaces. Each interior position is replaced with a finite difference approximation, which are used to construct linear equations for each point. Using this mathematical interpretation, we are able to set up and solve a system of linear equations (-A)u = b using the iterative Conjugate Gradient (CG) method. The CG. Since CG only requires matrix-vector products, it is quite an efficient iterative method to solve linear systems when the matrix "A" is sparse.[1][2]

2 CG Solver Implementation

To implement our CG Solver, we utilize object oriented programming in C++. First, we create a SparseMatrix class, which defines a sparse matrix object that contains the appropriate methods and variables to build a system of linear equations. We then define a HeatEquation2d class, which sets up and solves our linear system. To set up our SparseMatrix object, A, we treat each row of the matrix as an equation for a particular point on the discretized grid. We build our matrix by calling methods in the SparseMatrix class, which implement our matrix in COO format. We simultaneous build our solution vector (the right hand side "b" of our linear system) while building the matrix, This vector contains a zero element at the interior node positions and a heat boundary value, T_c or T_h, at the exterior boundary positions. In order to handle the edge case that there is only one row, we add to our solution vector additively so that both boundary positions are added to the right hand side of the equation. Lastly, we set our initial guesses for "u" as all zeros.

After converting the matrix to CSR format by calling the appropriate function within our SparseMatrix class, we are able to solve the system of linear equations using the CG method outlined below. This method takes in a SparseMatrix object, a solution vector, an initial guess vector, and a tolerance, and solves our linear system by iteratively approximating the solution. The CGSolver displays OOP by taking in an object and calling the object's method in order to calculate matrix-vector products. Additionally, the CGSolver computes vector operations (such as finding the L2 norm, vector-scalar multiplication, vector addition, and vector-vector multiplication) using a helper file, MatVecOps.

To avoid extraneous copying, we pass our SparseMatrix object and solution vector by reference into the CGSolver algorithm. [2]

Algorithm 1: CG Algorithm

```
initialize $u_0$;
$r_0 = b$ âĂŞ $Au_0$;
$L2norm0 = L2norm0(r_0)$;
$p_0 = r_0$;
$niter = 0$;
while $niter < nitermax$ do
    $niter = niter + 1$;
    $alpha_n = (r_n^T r_n) / (p_n^T A p_n)$;
    $u_{n+1} = u_n + alpha_n p_n$;
    $r_{n+1} = r_n - alpha_n A p_n$;
    $L2normr = L2norm(r_{n+1})$;
    if $L2normr/L2normr0 < threshold$ then
        break;
    end
    $beta_n = (r_{n+1}^T r_{n+1}) / (r_n^T r_n)$;
    $p_{n+1} = r_{n+1} + beta_n p_n$;
end
```

3 User Guide

1. Begin by running the *make* command on the command line to compile the most recent version of the program.
2. Run *./main [input file] [solution prefix]* to solve the steady state system with parameters give in the input file. Parameters given are the following: system length, system width, system grid spacing, T_h at the hot boundary and T_c at the cold boundary.
3. Run *python3 postprocess.py [input file] [solution file]* to do the following: plot a psuedocolor plot of the temperature distribution within the pipe wall, compute and overlay the mean temperature isoline, and report the mean temperature in the pipe wall.
4. Download created images which will be named solution***plot.png from rice and visualize the distributions. The *** will be replaced by numerical values corresponding to the iteration number.

4 Example Visualizations from Post Processing

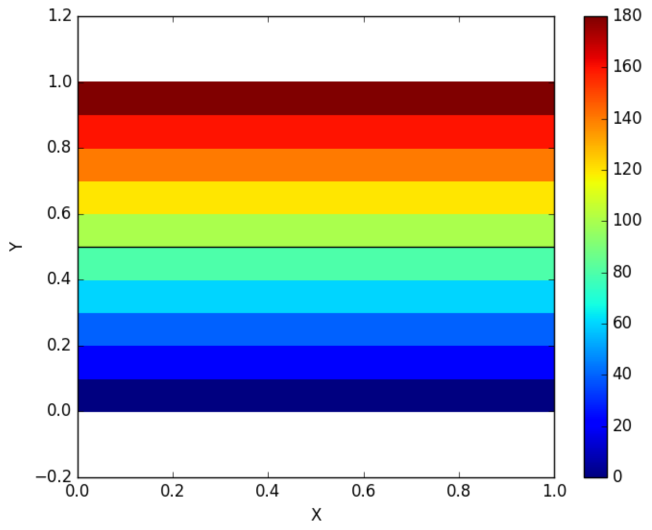

Figure 1: Plot for Input 0

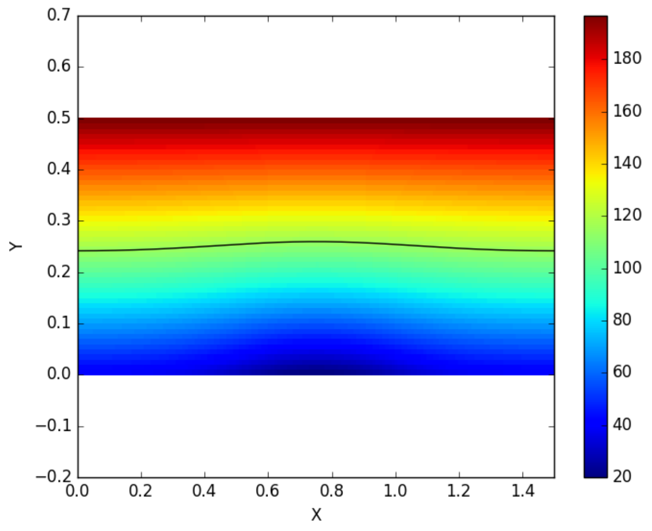

Figure 2: Plot for Input 1

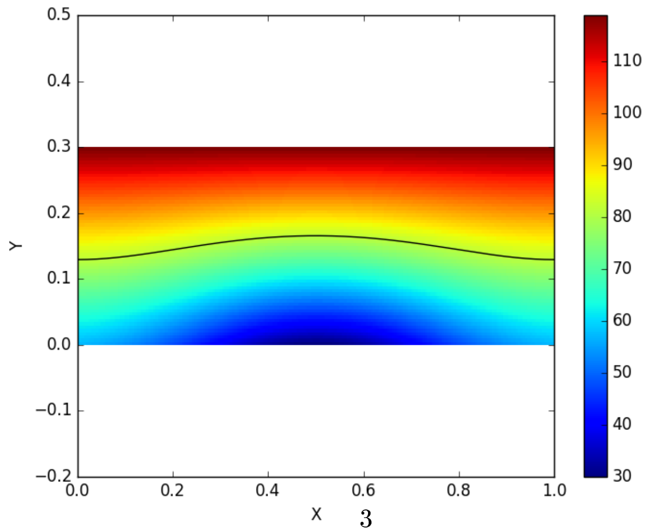

Figure 3: Plot for Input 2

References

[1] CME211. Final project part 1. http://coursework.stanford.edu, November 15 2021.

[2] CME211. Final project part 2. http://coursework.stanford.edu, December 7 2021.