CME 211 Project

Zherui Lin

December 15th 2022

1 Introduction

The project aims at solving the 2D steady-state heat equation on a simple geometry using a sparse matrix solver written in C++. The physical system of interest is a pipe transferring hot fluid, with temperature T_h. To keep the exterior of pipe cool, a series of cold air jets, with temperature T_c are equally distributed along the pipe and continuously impinge on the pipe surface. The simplified model of the system is one periodic section of the pipe wall, where the cool jet exactly points at the midpoint along the length of the periodic section. The objective is to determine the value of the mean temperature and display the temperature distribution within the pipe wall of the periodic section at steady-state. The problem is solved by discrete steady-state heat equations defined in each point on an equally spaced Cartesian grid of the pipe wall and applying appropriate boundary conditions at the exposed surfaces of the pipe wall and the periodic boundaries [1].

2 CG Solver

The key process in the project is to solve a system of discrete steady-state heat equations, which are linear. Thus we have to implement a method to solve the linear system, especially for sparse matrix, which is the feature of this system. The implementation in this project is a sparse Conjugate Gradient (CG) solver. The pseudocode for the CG algorithm to solve $\mathbf{Ax} = \mathbf{b}$ is as below, where the initial guess of solution $\mathbf{x}_0$ and the converge tolerance t should be provided as input as well [2].

```
function CG($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, t$)
    $n_{max} = dim(\mathbf{x}_0)$
    $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
    $\mathbf{p}_0 = \mathbf{r}_0$
    $n = 0$
    **while** $n < n_{max}$ **do**
        $n = n + 1$
        $\alpha_n = (\mathbf{r}_n^{\mathrm{T}}\mathbf{r}_n)/(\mathbf{p}_n^{\mathrm{T}}\mathbf{A}\mathbf{p}_n)$
        $\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n\mathbf{p}_n$
        $\mathbf{r}_{n+1} = \mathbf{r}_n - \alpha_n\mathbf{A}\mathbf{p}_n$
        **if** $\|\mathbf{r}_{n+1}\|/\|\mathbf{r}_0\| < t$ **then**
            **return** $\mathbf{x}_{n+1}$
        **end if**
        $\beta_n = (\mathbf{r}_{n+1}^{\mathrm{T}}\mathbf{r}_{n+1})/(\mathbf{r}_n^{\mathrm{T}}\mathbf{r}_n)$
        $\mathbf{p}_{n+1} = \mathbf{r}_{n+1} + \beta_n\mathbf{p}_n$
    **end while**
**end function**
```

In the implementation written in C++, the matrix **A** is wrapped up as a custom-defined SparseMatrix object, which we can easily interacted with. The CGSolver() method itself will be called by a custom-defined HeatEquation2D object, within which a specific setup can be solved. In summary, a specific physical setup will initialize a HeatEquation2D object, within which a SparseMatrix will be built up interactively, and it will be passed into GCSolver() method to solve this specific steady-state heat system.

3 User Guide

The project directory contains a bunch of .hpp and .cpp files. To compile C++ codes, **makefile** has wrapped up compile commands to compile and assemble all source files and link them together to output an executable main program. The detailed compile commands are as below.

```
g++ -c -o CGSolver.o CGSolver.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -c -o COO2CSR.o COO2CSR.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -c -o heat.o heat.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -c -o main.o main.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -c -o matvecops.o matvecops.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -c -o sparse.o sparse.cpp -O3 -std=c++11 -Wall -Wconversion -Wextra
g++ -o main CGSolver.o COO2CSR.o heat.o main.o matvecops.o sparse.o
```

To run the **main** program, the command usage is as below, where <input file> is the file that contains physical setups of the pipe system, and <soln prefix> is the prefix of the name of the solution files. Each solution file will contain the sequence of temperature data distribution in the 2D grid after specific times of iteration.

```
./main <input file> <soln prefix>
```

To run the **postprocess.py** program, the command usage is as below, where <input file> is the same as above, and <solution file> is the solution file we want to process. The program will output the mean temperature of the given solution file and generate a image file with the the same name as the solution file and a suffix of .png displaying a pseudocolor plot with the mean temperature isoline.

```
python3 postprocess.py <input file> <solution file>
```

To run the **bonus.py** program, the command usage is as below, where <input file> and <solution prefix> are the same as above. The program will generate an .gif animation with the name of the solution prefix, which displays the development of the temperature distribution of the physical setup given by the input file, generated by a series of valid solution files with the given solution prefix.

```
python3 bonus.py <input file> <solution prefix>
```

4 Example Figures

The output images of converged solutions generated by **python3.py** for **input1.txt** and **input2.txt** are shown in Figure 1 and Figure 2 on the next page.

References

[1] CME211. Project part 1. http://canvas.stanford.edu, December 6, 2022.

[2] CME211. Project part 2. http://canvas.stanford.edu, December 15, 2022.

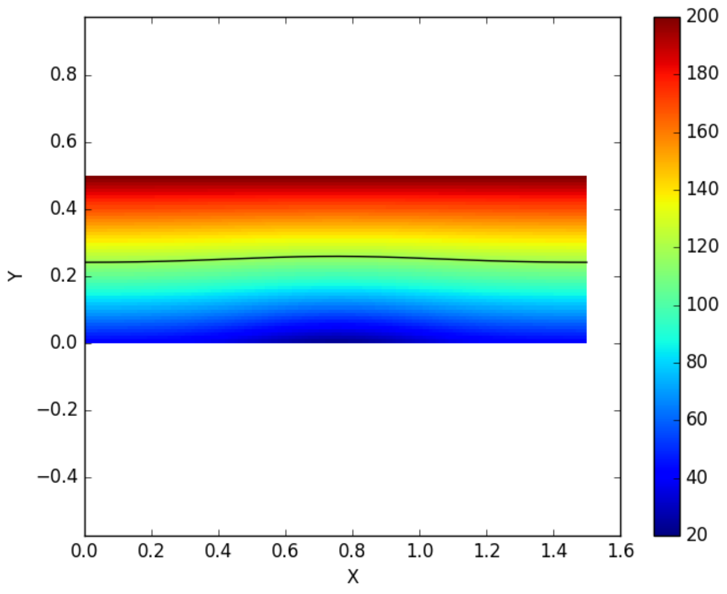

Figure 1: Pseudocolor Plot with Mean Temperature isoline for **input1.txt**

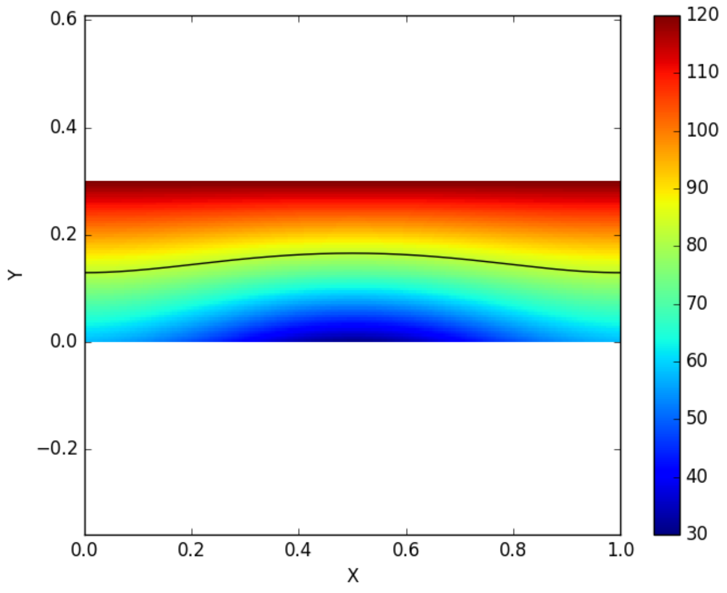

Figure 2: Pseudocolor Plot with Mean Temperature isoline for **input2.txt**