Final Project: Part 2
CME 211

Gabriel Buchsbaum

December 14, 2018

1 Introduction/Summary

The goal of this project was to create a program that could solve for the thermal gradient in the wall of a pipe carrying hot fluid and cooled by external jets. This program reads a file with setup information, and uses that to generate a system of equations represented in matrix form that can be solved to find the temperature at each location in the pipe. This system of equations is then solved, and temporary and final results are saved to a file. Finally, a second program was written to plot the temperature results, as well as a line tracking the location of the average temperature.

2 CG solver implementation

The CG solver relies on two objects plus two additional files: one containing the CG solver function, and one containing additional functions needed for manipulating vectors and matrices. The first object, the `SparseMatrix`, uses three vectors to represent a sparse matrix (i.e. one in which most of the data is zero and does not need to be specifically stored). With the equations represented in matrix form as $Ax = b$, this class stores the matrix A (holding the coefficients in the equations). This class includes functions to add entries, convert to the CSR form for easier calculations, and multiply the matrix by a vector (a function that effectively enters the matrix information into the already-existing function in `matvecops.cpp`).

The system of equations is set up by a `HeatEquation2D` object, which contains a `SparseMatrix` representing A and the vectors x and b. The `HeatEquation2D` reads data from a file to determine the necessary sizes of the matrices and vectors and the temperatures along the borders, then sets up the `SparseMatrix` to correctly link every data point. It then is used to input the equations into the `CGSolver` function. Since the `HeatEquation2D` contains information on the area dimensions and the known temperature data, and has more direct access to the desired file name, the function for saving the solution is part of the `HeatEquation2D` class.

The `CGSolver` function takes in the `SparseMatrix` object and the b and x vectors, and uses them to solve the system of equations. Since a reference to x is passed directly to the function, it can manipulate x directly despite x being a component of the `HeatEquation2D` class. The function also takes in a reference to the `HeatEquation2D` so that it can save the data. The `CGSolver` uses the multiplication function from `SparseMatrix`, as well as several functions from `matvecops.cpp`, to aid in the calculations.

See Algorithm 1 for the pseudocode of this algorithm.

3 User guide

The file `makefile` can perform can compile the program easily, just by using the `make` command.

```
$ make
g++ -c -o main.o main.cpp -std=c++11 -O3 -Wall -Wextra -Wconversion -Wpedantic
g++ -c -o CGSolver.o CGSolver.cpp -std=c++11 -O3 -Wall -Wextra -Wconversion -Wpedantic
g++ -c -o COO2CSR.o COO2CSR.cpp -std=c++11 -O3 -Wall -Wextra -Wconversion -Wpedantic
```

Algorithm 1 CG Solver Pseudocode

$u_0 \Leftarrow x$
$r_0 \Leftarrow b - Au_0$
$L2normr_0 \Leftarrow \|r_0\|_2$
$p_0 \Leftarrow r_0$
$n_{iter} \Leftarrow 0$
while $n_{iter} < n_{iter,max}$ **do**
  if $n_(iter)$ is divisible by 10 **then**
    Save results
  end if
  $n_{iter} \Leftarrow n_{iter} + 1$
  $\alpha_n \Leftarrow (r_n^\intercal r_n)\,(p_n^\intercal A p_n)$
  $u_{n+1} \Leftarrow u_n + \alpha_n p_n$
  $r_{n+1} \Leftarrow r_n - \alpha_n A p_n$
  $L2normr \Leftarrow \|r_{n+1}\|_2$
  if $\frac{L2normr}{L2normr_0} < \epsilon$ **then**
    break
  end if
  $\beta_n \Leftarrow \frac{r_{n+1}^\intercal r_{n+1}}{r_n^\intercal r_n}$
  $p_{n+1} \Leftarrow r_{n+1} + \beta_n p_n$
end while
if Desired tolerance not achieved **then**
  Report error and exit
end if
$x \Leftarrow u_{n+1}$
return n_{iter}

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

The makefile can also remove the object and editor files, as well as main:

```
$ make clean
rm -f *.o *~ main
```

To use the program, it must be provided with a correctly-formatted input file. The first row of the file contains the length and width of the zone being solved for, then the separation of points in the solution. The second row contains the temperature of the hot boundary and the temperature of the cold jet. For example:

```
$ cat input2.txt
1.0 0.3 0.005
30 120
```

The user needs to provide main with this file, as well as a prefix to use for the solution files:

```
$ ./main input2.txt solution
SUCCESS: CG solver converged in 157 iterations.
$ ls solution*.txt
solution000.txt  solution050.txt  solution100.txt  solution150.txt
solution010.txt  solution060.txt  solution110.txt  solution157.txt
solution020.txt  solution070.txt  solution120.txt
solution030.txt  solution080.txt  solution130.txt
solution040.txt  solution090.txt  solution140.txt
```

The user can then run the post processing script by giving it the original input file and the final solution file[1]:

```
$ python3 postprocess.py input2.txt solution157.txt
Input file processed: input2.txt
Mean Temperature: 81.80566
```

`postprocess.py` will also save the plot as an image with the same name as the solution file, albeit with the `.png` extension, i.e. `solution157.png`.

To run the bonus animation, the same input as `postprocess.py` is used. While every intermediate solution file is used, the program can determine these intermediate file names from the last file name.

```
$ python3 bonus.py input2.txt solution157.txt
Input file animated: input2.txt
```

`bonus.py` saves the animation using a similar convention, i.e. `solution157.mp4`. It does require the `ffmpeg` writer to save the file. An example has been uploaded, in case the code does not function in the environment being used.

4 Images

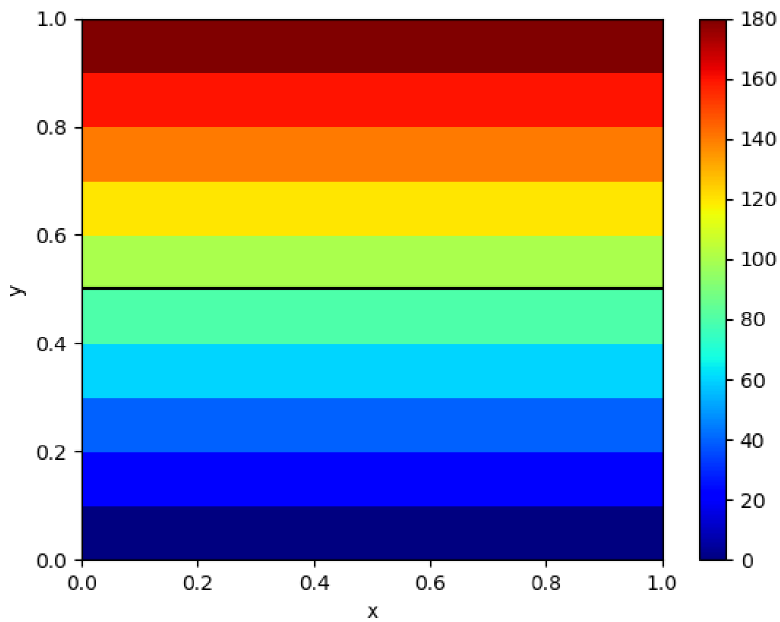

Figure 1: Post processing results for `input0.txt`

[1] This can also work on intermediate solutions, although the results will obviously not be correct

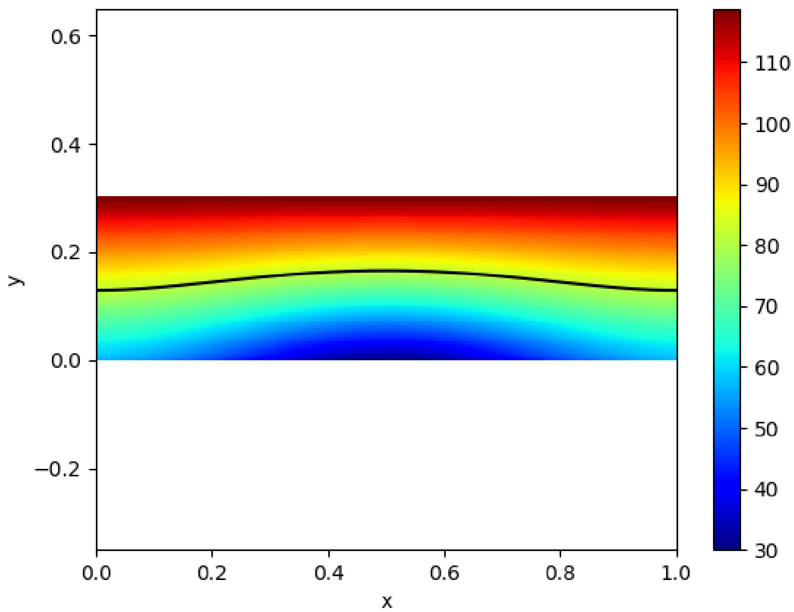

Figure 2: Post processing results for `input2.txt`

References

[1] CME211. Final project: Part 1. http://canvas.stanford.edu, November 9, 2018.

[2] CME211. Final project: Part 2. http://canvas.stanford.edu, November 25, 2018.