Robert Spragg
December 9, 2017

1 Introduction

The goal of Part 2 of this project was to solve the 2D steady-state heat equation by implementing a Conjugate Gradient (CG) solver using Object Oriented Programming (OOP) in C++. To do this, 2 classes, *SparseMatrix* and *HeatEquation2D*, were developed. Each of these classes contains methods and data attributes required to solve the system of equations. The CG solver and matvecops files developed in Part 1 of the project were modified accordingly.

After the CG Solver converges, a text file containing the solution vector is written. This file, along with the input file containing the required pipe geometry and temperature profile, are inputs for a python file, which uses matplotlib to create a pseudocolor plot of the domain. The python file also returns the mean temperature of the pipe, and draws an isotherm along the length of the pipe where this temperature occurs.

2 CG Solver Implementation in OOP

Two classes were used to compartmentalize the project's key components. The HeatEquation2D class contains two methods —Setup and Solve. The SparseMatrix class contains four methods —Resize, AddEntry, ConvertToCSR, and MulVec. The SparseMatrix class also contains the data attributes that are used in CG Solver: i_idx, j_idx, and a. Since these attributes are private, they can only be accessed and modified by objects and methods belong to the same class. Therefore, we define a SparseMatrix object 'A' in heat.hpp. This object is used each time we need to access the data attributes, such as in CG Solver.

The *Setup* method reads the input file and generates the necessary values for the sparse matrix, as defined by the discrete form of the heat equation.

$$\frac{1}{h^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}) = 0$$

CGSolver.cpp is called in the *Solve* method of HeatEquation2D. It is called after the components of the COO matrix are entered and the matrix has been converted to CSR format via the *ConvertToCSR* method.

```
int test_output = CGSolver(A,b,x,tol,soln_prefix);
```

Figure 1: Line of code used in heat.cpp to call CGSolver method

The SparseMatrix object A is used each time we need to access the row index, column index, and value associated with the CSR matrix inside of CG Solver. For example, to multiply the CSR matrix by the solution vector u, one must call the MulVec method on the SparseMatrix object, as shown below.

```
std::vector<double> A_u = A.MulVec(u);
```

Figure 2: Line of code used in CGSolver.cpp to call MulVec method

3 CG Solver Pseudocode

The pseudocode for CGSolver remains the same as in Part 1. In the following equations, 'A' refers to the CSR matrix A.

Algorithm 1: Uses Conjugate Gradient method to solve for solution vector u

function Conjugate Gradient (A, b, u);
Input : CSR Matrix A, RHS vector b, initial solution vector u_o
Output: u
initialize u_0
r_o = b - A*u_0
L2normr0 = L2norm(r_0)
$p_0 = r_0$
niter = 0
while *niter* < *nitermax* **do**
  niter = niter + 1
  alpha = $(r_n^T r_n)/(p_n^T\, A\, p_n)$
  $u_{n+1} = u_n + alpha_n * p_n$
  $r_{n+1} = r_n - alpha_n * A * p_n$
  $L2normr = L2norm(r_{n+1})$
  if $L2normr/L2normr0 < threshold$ **then**
    break;
  $beta_n = (r_{n+1}^T r_{n+1})/(r_n^T r_n)$
  $p_{n+1} = r_{n+1} + beta_n * p_n$
end
return Solution Vector u;

CGSolver begins by initializing the residual matrix r_0. To do this, we must multiply our CSR matrix by our initial solution matrix u_0, and then subtract our RHS vector b by $A * u_0$. This is done by initializing a solution vector A_u and calling the MulVec function that is in 'sparse.cpp'. By placing the mathematical operations methods within other file(s), we can make the CGSolver file look more similar to the provided pseduocode, which makes it *easier* for the user to interpret and debug. It also allows the solver to run without letting the data attributes becoming modified by methods external to their respective class.

For Part 2 of the project, not only does CGSolver output the final solution vector as a .txt file, but it also outputs the initial guess and every tenth iteration of the solution vector. These outputs are used in the bonus.py file, which provides an animation of the convergence of the solver.

4 User Guide - Required Files, Command Line Arguments, Postprocessing

The following files are necessary for the C++ makefile to compile correctly.

- main.cpp
- heat (.hpp and .cpp)
- sparse (.hpp and .cpp)
- COO2CSR (.hpp and .cpp)
- CGSolver (.hpp and .cpp)
- matvecops (.hpp and .cpp)
- makefile

C++ Usage Statement:
./main <input geometry file name> <output solution prefix>
The input geometry files contain the pipe length, width, and element size, as well as the isothermal BCs. **Note:** the solution prefix is used when naming the solution outputs from CGSolver. For example, writing 'cat' would create the .txt files cat000.txt, cat010.txt, etc.

The following python files were also generated.

- postprocess.py
- bonus.py (for optional animation section)

Python Usage Statement:
python3 postprocess.py <input geometry file name> <solution<#>.txt >
Postprocess.py reads the provided input file to establish the geometry and calculate the hot and cold boundary temps. Next, it reads the solution file and places the temperatures in a numpy array. Finally, it generates a plot file [i.e. input1.txt makes plot1.png] and calculates the mean temperature of the domain.

5 bonus.py

Usage Statement python3 bonus.py <input file name> <soln prefix>

Bonus.py searches through the current directory and finds all the .txt files that begin with the provided soln prefix. It establishes the temperatures along the hot and cold boundaries, sorts the solution files chronologically, and then iterates through each solution file. For each solution file, it completes the temperature array. Next, it appends a pseudocolor plot of the x and y values and each temperature array to the 'ims' list. It does the same for the isotherm plot as well (places it in 'ims_iso'). Finally, after all solution files have been read, the ArtistAnimation method of matplotlib.animation is called, which generates a dynamic plot that overlays the plot lists. The time interval of each frame, in milliseconds, as well as the repeat delay, can be adjusted as necessary.

A video of my animation can be found here, or by copying and pasting the following link: https://spragg.tinytake.com/sf/MjE4Mzc4M182Nzg3NjU3
I was not able to save a .mp4 file of the animation on either my PC or Corn because neither one has the ffmpeg tool installed. The following image shows the error message received from Corn when trying to save.

```
Animation generation was successful!
/usr/lib/python3/dist-packages/matplotlib/animation.py:696: UserWarning: MovieWriter ffmpeg unavailable
  warnings.warn("MovieWriter %s unavailable" % writer)
Traceback (most recent call last):
  File "bonus.py", line 123, in <module>
    im_ani.save('t_animate.mp4')
  File "/usr/lib/python3/dist-packages/matplotlib/animation.py", line 713, in save
    with writer.saving(self._fig, filename, dpi):
AttributeError: 'str' object has no attribute 'saving'
```

Figure 3: Error message in corn

6 Results and Figures

The provided input files (input0.txt, input1.txt, and input2.txt), converge in 9, 132, and 157 iterations, respectively. The figure below shows the pseudocolor plot for input1 and input2 at convergence.

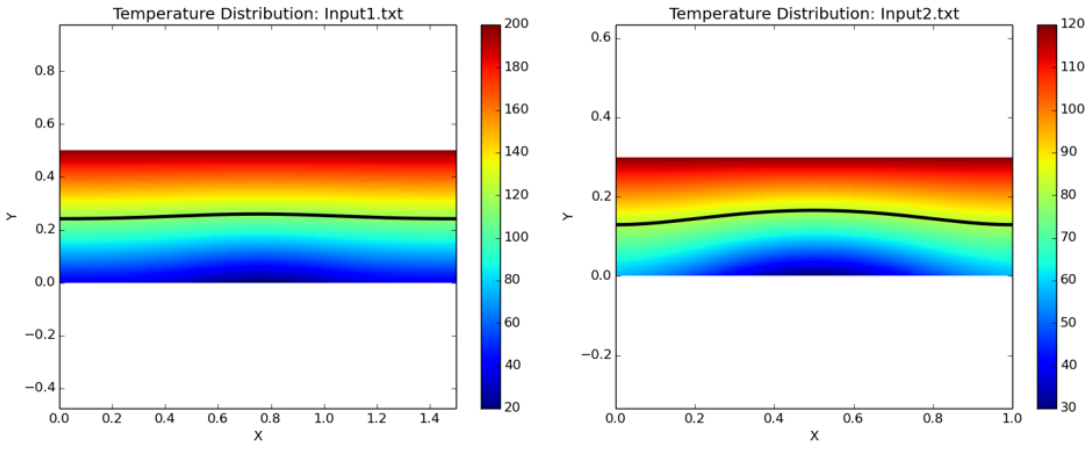

Figure 4: Steady-State solution for inputs 1 and 2, respectively.

7 Discussion

Through this project, I was able to develop an understanding of OOP in C++. Furthermore, by implementing the Conjugate Gradient algorithm, I was reminded of the power of abstraction in facilitating code development on large projects with many methods. Finally, by utilizing both C++ and Python, I gained a better handling of both languages and how they can be used to complement each other.

8 References

LeGresley, Patrick. "CME 211: Project Part 1." Stanford University, Nov. 2017, Class handout.

LeGresley, Patrick. "CME 211: Project Part 2." Stanford University, Nov. 2017, Class handout.