CME 211: Project Part 2

Due: Monday, December 11, 2017 at 6:30 PM Pacific Time

This project was designed by Patrick LeGresley and modified for the purposes of this course.

Background

In Part 1, you completed the preliminary development of a sparse Conjugate Gradient (CG) solver and are now ready to solve the steady-state heat equation. To accomplish this we will be considering the following scenario. Consider a system where you are transferring some hot fluid, with temperature T_h, within a pipe. To keep the exterior of the 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. We are interested in determining the value of the mean temperature within the pipe walls. To model this problem, we will analyze one periodic section of the pipe wall (i.e. this section will repeat in both directions along the pipe), using a simplified 2D geometry as seen in Figure 1.

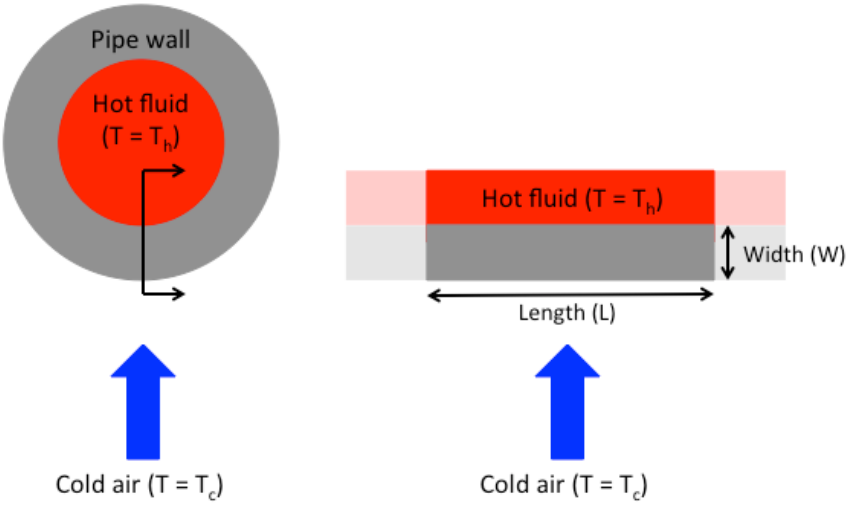

Figure 1: Problem geometry

Since we are only interested in what is going on inside the pipe wall, we can focus on this portion of the geometry by discretizing the pipe wall into an equally spaced Cartesian grid ($\Delta x = \Delta y = h$) and applying appropriate boundary conditions at the exposed surfaces of the pipe wall and the periodic boundaries. A sample discretization of the pipe wall is shown in Figure 2.

Recall the discrete form of the steady state heat equation in 2D defined at every point (i, j) (where i refers to the x axis direction and j refers to the y axis direction) in the domain provided in the Part 1 document:

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

You will need to assemble these equations, using the boundary conditions provided below, into a linear system of equations $Au = b$ and solve it using the CG method.

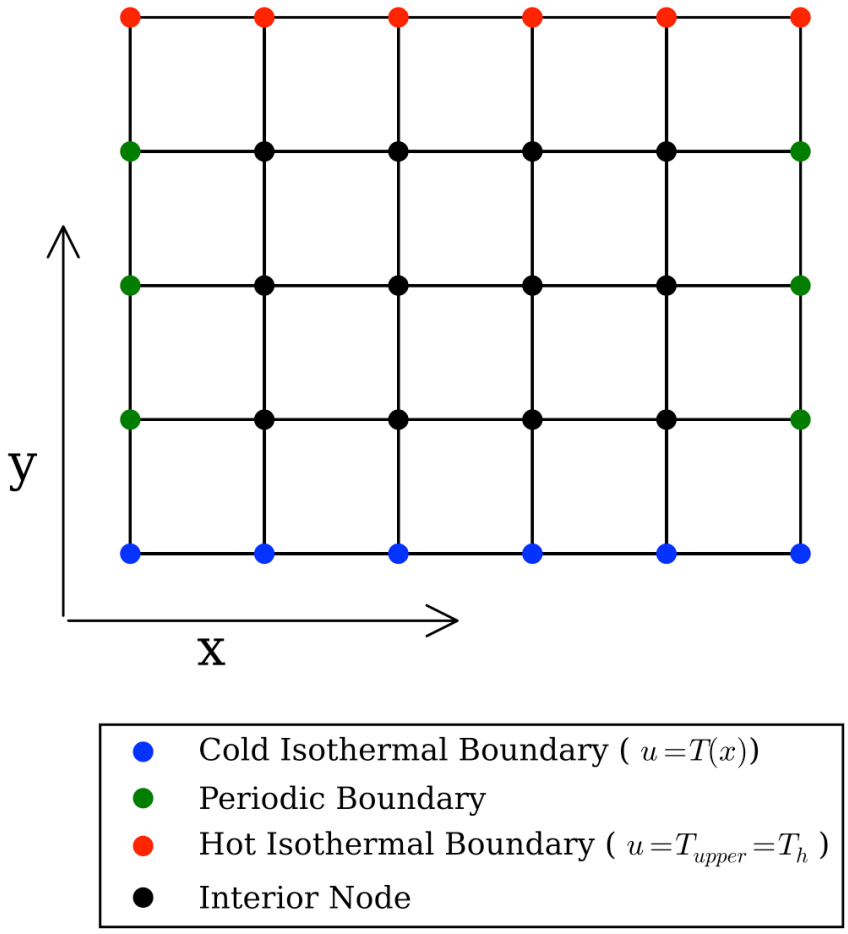

Figure 2: Discretization of the pipe wall

Important Considerations in forming matrix A

1. Using this form of the equations matrix A will be negative definite, so in your implementation, you need to form and solve the system $(-A)u = -b$.
2. The structure of A is dependent on the ordering of the unknowns in your solution vector u. To maintain symmetry in this problem you need to use the natural ordering where the solution vector unknowns are ordered by row:
$$\mathbf{u} = \left[u_{(1,1)},\ u_{(2,1)}, \ldots,\ u_{(nx,1)},\ u_{(1,2)}, \ldots,\ u_{(nx,ny)}\right]^T$$
3. Creation of the matrix A can be simplified by utilizing the fact that the provided COO to CSR conversion function supports accumulation of entries. If in the COO representation there are multiple entries for the same row and column, those entries will automatically be summed when creating the CSR representation.

Boundary Conditions

There are two types of boundaries you will need to handle in this problem: isothermal and periodic.

For an isothermal boundary, the values of u are fixed to a specified temperature. These are known quantities so they should not be included as unknowns in your solution vector, but placed in the right-hand side vector b

when they appear in the equation for a given point $u(i, j)$. An example of a point that includes an isothermal boundary is provided in Figure 3. For our model, the temperature along the hot isothermal boundary is constant and equal to T_h. To model the cool jet, the lower boundary has an inverted Gaussian temperature distribution that is a function of x, defined as:

$$T(x) = -T_c \left(\exp\left(-10 \left(x - L/2 \right)^2 \right) - 2 \right)$$

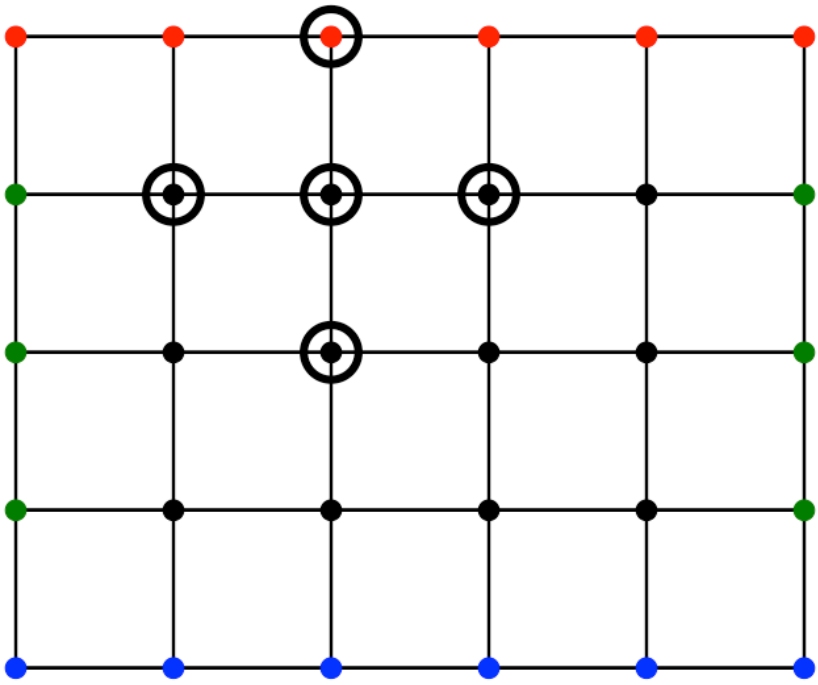

Figure 3: Stencil that includes an isothermal boundary point

For the periodic boundaries, the points on the left and right side denoted by the green dots are considered to be coincident. In this case the stencil for forming the equation for these boundary points will wrap around the domain, using information from the opposite side. An example of how this works can be found in Figure 4. To ensure that the matrix maintains symmetry, you should treat these periodic points as single unknowns in your system.

Part 2

For the completion of your project you will be reorganizing your existing code into an Object Oriented Programming (OOP) design, developing the additional code necessary to solve the heat equation on the described geometry, and developing a post processing code in Python to visualize your solution.

The OOP design requires the development of 2 classes: `SparseMatrix` and `HeatEquation2D`. The `SparseMatrix` class will define a sparse matrix object, which contains the vectors defining the matrix as data attributes and methods to perform necessary tasks (add entries, conversion from COO to CSR format, etc.) The `HeatEquation2D` class will contain data and methods required to form the system of equations, solve the system using the solver you have implemented, and output the results to files.

The output will be a series of solution files (`solution000.txt`, `solution010.txt`, etc.) containing the solution as it converges during your CG solve. You should output the solution after every 10 iterations,

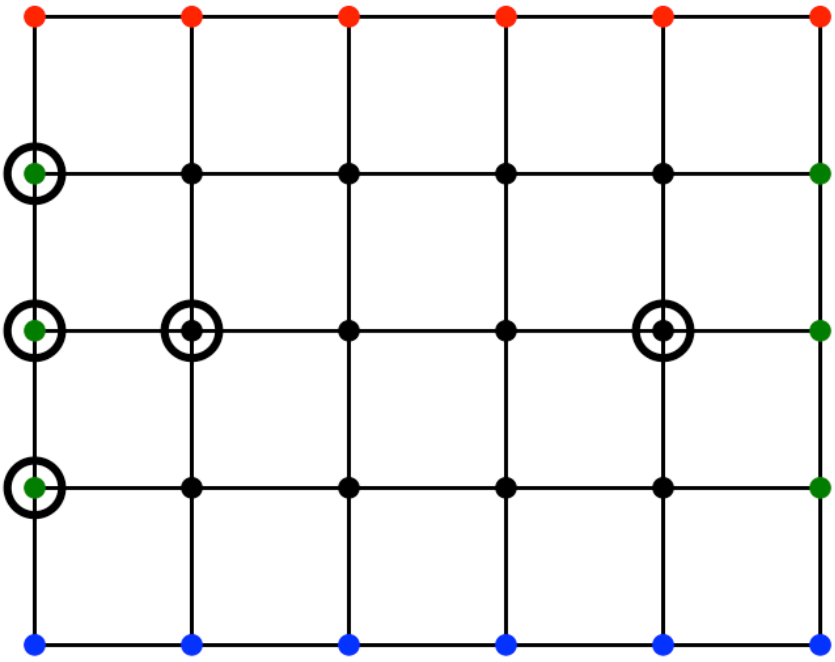

Figure 4: Stencil that includes a periodic boundary point

including the first and last iteration. You are free to pick the format for your output files but obviously your Python post processing code should be compatible with the format you have selected.

For your post processing code in Python, you will need to read in a solution file and perform the following tasks:

1. Compute and report the mean temperature within the pipe wall.
2. Generate a pseudocolor plot (with colorbar) of the temperature distribution within the pipe wall. Additionally, you should compute and overlay the mean temperature isoline on this plot. You can compute the coordinates of this isoline using 1D linear interpolation along the width of the domain.

Note that the solution files, plots, and associated computations should contain the isothermal boundary points, even though they are not treated as variables in your system of equations.

To assist in your efforts, prototypes for a `SparseMatrix` and `HeatEquation2D` class have been provided for you, along with a sample `main()`. While you are not allowed to change the names of the classes or eliminate either of them you can modify the attributes as you see fit.

Note that your C++ code will no longer read in a matrix from an external file because you will be forming the matrix within your program. Instead, your program will take as input an input file containing information describing the geometry (length, width, and h are listed in the first row. T_c and T_h are listed in the second row. See for example `input1.txt` and `input2.txt`) and the prefix for the name of the solution files:

Sample output of your C++ program

```
$ ./main input1.txt solution
SUCCESS: CG solver converged in xx iterations.
```

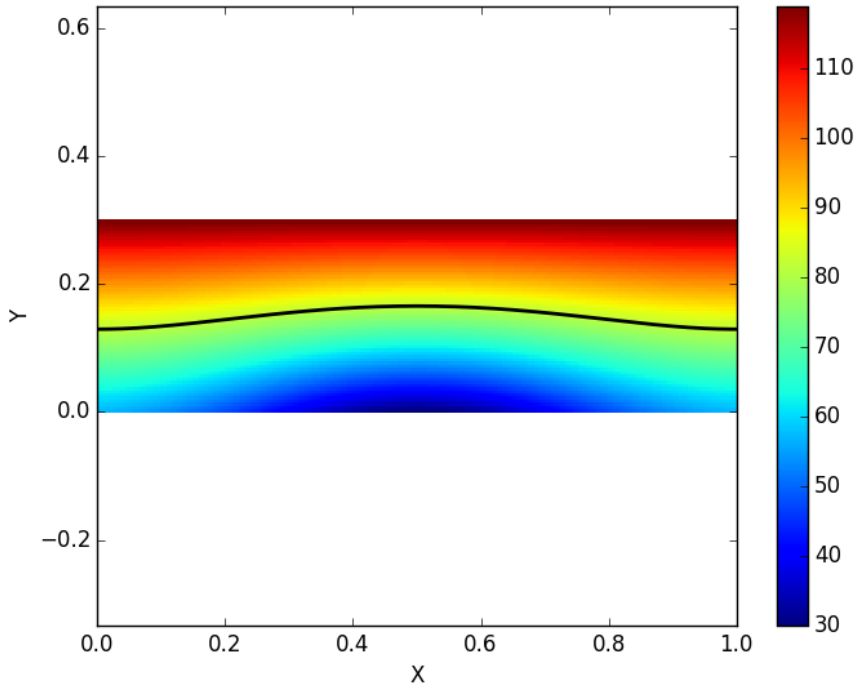

Figure 5: Example of pseudocolor plot with mean temperature isoline for `input2.txt`

Sample output of your Python code

```
$ python postprocess.py input1.txt solution<#>.txt
Input file processed: input1.txt
Mean Temperature: xx.xxxxx
```

For ease of compilation you *must* provide a `makefile` that by default compiles your code with `-O3` optimizations. You should also have a clean target that removes any `*~` or `*.o` files, as well as the executable created by the default target.

Finally, you will need to update and complete your writeup. See the summary of requirements for details.

Bonus

Generate an animation of the temperature distribution development during the CG solve using the animation module of matplotlib. An example of how to do this can be found at: http://matplotlib.org/1.3.0/examples/animation/basic_example.html. The frames of this animation will be pseudocolor plots, normalized using T_h and T_c. This code should be separately implemented in `bonus.py`.

Note that we will not answer questions related to implementing the bonus. The point of the bonus is for you to demonstrate that you know how to use the documentation and learn how to utilize new functionality on your own.

Summary of requirements

1. Complete the implementation of the `SparseMatrix` and `HeatEquation2D` classes:
    a. Start from the provided prototype, modifying attributes as you see fit.

   b. Make the required changes to `CGSolver.cpp` and `CGSolver.hpp` to utilize an instance of the `SparseMatrix` class and write the solution files.

2. Implement a post processing script in the file `postprocess.py` that:
   a. Plots a pseudocolor plot (with colorbar) of the temperature distribution within the pipe wall and overlays the mean temperature isoline.
   b. Reports the mean temperature in the pipe wall. Be sure to include the values from boundary points in this calculation.
3. Writeup written in LaTeX that includes the following sections / topics:
   a. Short introduction / summary of the overall project.
   b. Description of the CG solver implementation that includes background on the OOP design for the matrix and solver classes and includes the pseudocode for the CG algorithm.
   c. "Users guide" that demonstrates how to compile and run the code, including postprocessing and visualization of the results. If you implement the bonus please also include instructions for using that code.
   d. One or more example figures showing the visualizations from the postprocessing.
   e. References section that provides citations for the Part 1 and Part 2 handouts.
   f. Please limit your overall writeup to no more than 4 pages total, including figures.
4. Provide a `makefile` that
   a. `$ make`: must compile all source code and produce `main` with `-O3` optimization
   b. `$ make clean`: must remove all object (`*.o`) and editor files (`~*`) files and remove `main`
5. Finally, remember that the project is in lieu of a final exam and is meant to demonstrate your understanding of software development. You need to follow good programming practices, etc. The feedback for Part 1 is not an endorsement that your code to that point was well written. You should review all of your code for appropriate design and style considerations.

Project submission

Please be very careful with directory and filenames, as points will be deducted if you do not follow the directions. To be clear you should have a minimum of these files in the `project` directory in your GitHub repository:

- `CGSolver.cpp`
- `CGSolver.hpp`
- `COO2CSR.cpp`
- `COO2CSR.hpp`
- `makefile`
- `heat.cpp`
- `heat.hpp`
- `main.cpp`
- `matvecops.cpp`
- `matvecops.hpp`
- `postprocess.py`
- `sparse.cpp`
- `sparse.hpp`
- `writeup.tex`
- `writeup.pdf`

Do not commit temporary files produced by your text editor. We will deduct points if these instructions are not followed. When working in teams and professional environments, it is important to keep your repositories free of extraneous files.

Whatever files you have in your GitHub repository at the deadline will be considered your final submission.

We cannot access the working copy in your account on the `corn.stanford.edu` machines, so if you fail to commit and push your files back to GitHub your assignment will not be considered submitted. Emailed submissions will not be accepted. Do not wait until the last minute to make sure you understand `git` usage.