Finite Difference Solution of a Boundary Value Problem

Introduction

In this project the aim is to solve a thermodynamic system where a hot fluid is being transferred within a pipe, with a constant temperature T_h. A series of cold air jets with equal distances from each other are distributed to keep the exterior of the pipe cool, each with a temperature of T_c along the pipe and continuously cool the pipe surface. To find the mean temperature within the pipe walls, one periodic section of the pipe wall is modelled. In Figure 1, the 2D geometry of the model is illustrated.

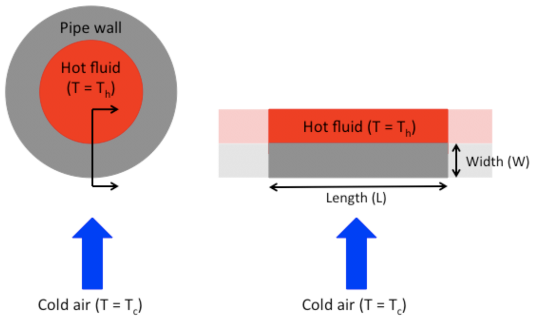


Figure 1: Overall geometry of the system (2)

The pipe wall is discretized into an equally spaced Cartesian grid. Then, appropriate boundary conditions at the exposed surfaces of the pipe wall and the periodic boundaries are applied to get the discretization in figure 2a.

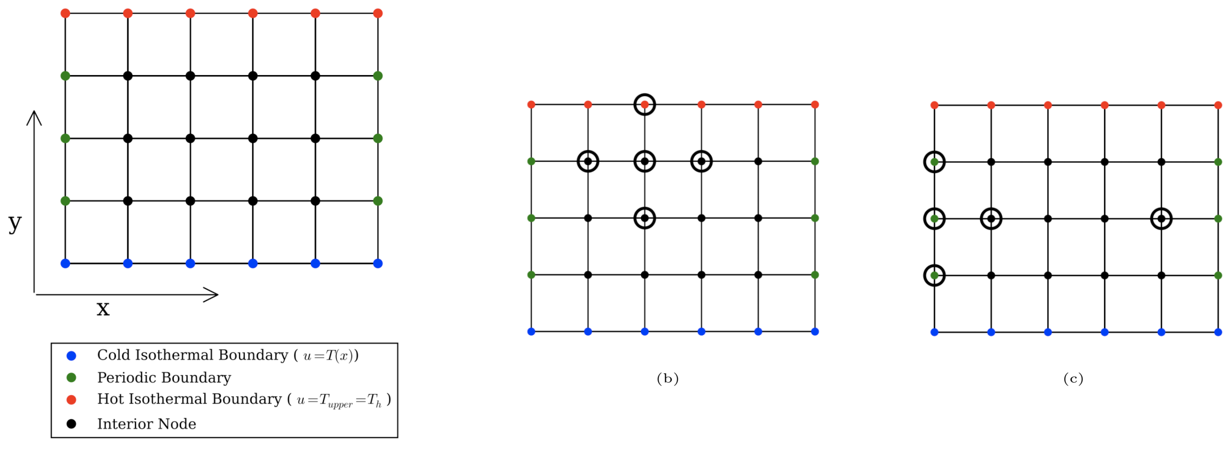

y
x
● Cold Isothermal Boundary (u = T(x))
● Periodic Boundary
● Hot Isothermal Boundary (u = T_upper = T_h)
● Interior Node
(a)
(b)
(c)

Figure 2: a) Discretization of the geometry b) Stencils of an interior node c) Stencils of an edge node (2)

Edges of the model are periodic boundary conditions, and cold isothermal boundary is described by the function:

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

Using the second order finite difference approximation of the steady state heat equation (below) for the discretized geometry, a linear system of equations is achieved $Au = b$.

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

heat.cpp and sparse.cpp:

The two main classes used in the project are heat.cpp and sparse.cpp. In sparse.cpp, a SparseMatrix class is defined in which three vectors define the row, column and values of nonzero elements of a matrix. The methods used in this class are

- Resize, used for updating size of the matrix,
- AddEntry, method for adding a nonzero element to matrix,
- ConvertToCSR, method to change format of the matrix from COO to CSR.

In heat.cpp, HeatEquation2D class is defined and using SparseMatrix class, two methods handle the 2D equation of heat problem:

- Setup: Here, matrix A is formed row by row while right hand side b is formed at the same time. A helper matrix M is also built here which contains node numbers. Node numbering is held in column order, starting from the bottom left non-boundary node. One side note to point out is, since matrix A is negative definite and we need a positive definite matrix for guaranteed convergence for conjugate gradient in n steps for an $n \times n$ matrix, we form $-A$ which is positive definite and solve for $-Au = -b$.
- Solve: In this method, first, the format of matrix A is converted to CSR and an initial guess vector x with 1.0's for all of its elements is created, and CGSolver() is called to update it in place with a tolerance of $1e-05$.

CG Solver

CGSolver.cpp is a function that uses conjugate gradient algorithm (1) to solve a linear system equations $Ax = b$ iteratively where A is a SparseMatrix class object in CSR format. A few notes on the CGSolver function:

- The solver runs a maximum number of iterations equal to the size of the linear system.
- The initial guess is updated in-place as the right hand side and the other inputs are unchanged.
- x is written in a text file after every 10 iterations, including the first and last iterations, with the name prefix followed by iteration number as the name of file.
- Function returns the number of iterations held to converge the solution to the specified tolerance, or -1 if the solver did not converge.

The algorithm of CGSolver can be found in the Appendix.

User's Guide

The software consists of two main programs: main.cpp and postprocess.py. In main.cpp, an input file and a prefix for the output files are required. The usage would be as follows:

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

Input file should contain information describing the geometry (length, width, and h) which are listed in the first row. Tc and Th should be listed in the second row.

An example input file for L=2.0, width=1.0, h=0.1, T_c=30 and T_h=120 would be:

```
2.0 1.0 0.1
30 120
```

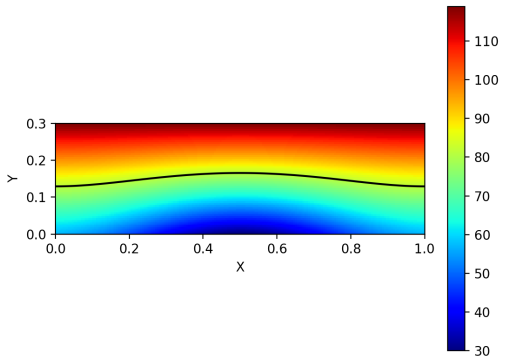
When initiated, `main` reads the given input file and builds a `HeatEquation2D` class with the given parameters, in which a `SparseMatrix` class is initialized. After `Setup` method is called a status check is triggered whether the matrix setup of A and vector b was successful or not. The same status check is performed after `Solve` is held as well. The output after `Solve` is called are the solution of non-boundary nodes in every 10 iterations including first and last with names starting with solution prefix given as input followed by the iteration number with fixed width of 4 digits with leading zeros.

To visualize the outputs of `main.cpp`, `postprocess.py` needs to be called with the following usage:

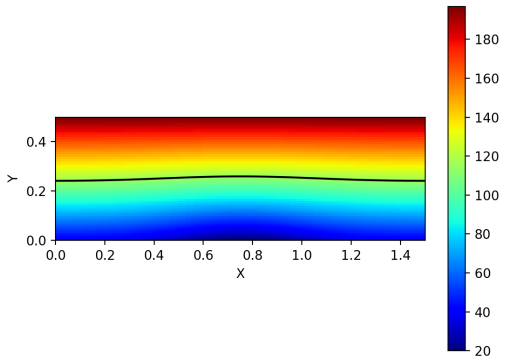
```
$ python3 postprocess.py <input file> <solution file>
```

`postprocess.py` creates a grid with the solution and boundary conditions and creates a pseudo-color plot with a isoline of the mean temperature in the wall section. In the command prompt, it shows the mean temperature and the input file associated with it. If the input file is not associated with the output file or one of the files could not be found, it raises a run-time error.

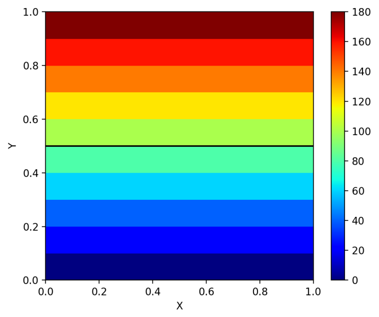
Example Outputs



(a) input02.txt



(b) input01.txt



(c) input0.txt

Figure 3: Output plots of given input files

References

[1] CME 211 Project Part I Handout, Fall 2019

[2] CME 211 Project Part II Handout, Fall 2019

0.1 Appendix

Algorithm 1: Conjugate Gradient

```
Data: A, a SparseMatrix object in CSR format, x, the initial guess and b, the RHS for $Ax = b$
Result: Number of iterations took for updating x s.t $|r| < tol$ for $r = Ax - b$
begin
    $Ax \longleftarrow A \times x$
    $r_0 \longleftarrow Ax - b$
    $p_0 \longleftarrow r_0$
    Initialize $x_{n+1}, r_{n+1}, p_{n+1}$
    niter=0
    nitermax= size of linear system
    while $niter \leq nitermax$ do
        niter++
        $\alpha = \frac{(r_n^T r_n)}{(p_n^T A p_n)}$
        $x_{n+1} = x_n + \alpha_n p_n$
        $r_{n+1} = r_n - \alpha_n A p_n$
        if $|r_{n+1}| < tol$ then
            $x \longleftarrow x_{n+1}$
            break while loop
        else
            $\beta_n = (r_{n+1}^T r_{n+1})/(r_n^T r_n)$
            $p_{n+1} = r_{n+1} + \beta_n p_n$
    return niter;
```