Implementation of a 2D heat solver using a conjugate gradient algorithm

Ren Gibbons
December 14, 2016

1 Overview

This document provides an overview a 2D heat solver program written for the CME 211 course project at Stanford University. The code's application is to determine the distribution of temperature through the shell of a pipe carrying hot fluid. In this scenario, a hot fluid with fixed temperature flows through a pipe, and a cooling agent surrounds the pipe's exterior. The boundary solved is a rectangular slice along the length of the pipe where the top and bottom boundaries have a fixed temperature. The right and left boundary conditions are periodic, meaning that we model the nodes on the left boundary and the right boundary as coincident locations. Periodic boundary conditions ensure symmetry of the linear system and allow us to repeat the boundary over any number of periods.

The heat solver uses the finite difference method satisfying the discretized Laplace partial differential equation

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

to assemble a linear system of equations [2]. The system $Ax = b$ is solved using the conjugate gradient algorithm. The conjugate gradient algorithm is an efficient iterative approach to solving sparse linear systems. The implementation of the CG method was required for the first submission of the project [1]. This document contains sections describing the conjugate gradient solver, use of the code, and basic results.

2 Conjugate gradient solver

The code is structured such that the `Heat` class contains a method `Setup()` that assembles a linear system matrix A and a right hand side vector b. Matrix A is a sparse matrix whose data attributes are contained in an object of the `SparseMatrix` class. `SparseMatrix` contains the following private variables: number of rows (`int`), number of columns (`int`), row data (`std::vector<int>`), column data (`std::vector<int>`), and element entry data (`std::vector<double>`). Once the system is assembled, the `Heat` method `Solve()` calls a function `CGSolver()` to find the solution to the linear system. The conjugate gradient algorithm is presented in the psuedocode on the following page.

The conjugate gradient algorithm uses several common matrix operations which are contained in the source file `matvecops.cpp` to reduce clutter in the primary solver code. The methods contained in `matvecops.cpp` are:

- `addVecs()` : Add the elements of two similar-sized vectors. Return a vector.
- `L2norm()` : Compute the L^2-norm of a vector. Return a scalar.

- `multMatVec()` : Multiply a square matrix by a dimensionally homogeneous column vector. Return a vector.
- `multScalar()` : Multiply the elements of a vector by a scalar. Return a vector.
- `multVecMatVec()` : Multiply the transpose of column vector by a square matrix by the same column vector. Return a scalar.
- `multVecs()` : Perform the dot product of two similar-sized vectors. Return a scalar.
- `subVecs()` : Subtract the elements of two similar-sized vectors. Return a vector.

Algorithm 1 Conjugate gradient solver

initialize $\mathbf{u}_\circ = \mathbf{1}$
$\mathbf{r}_\circ = \mathbf{b} - \mathbf{A}\mathbf{u}_\circ$
$|\mathbf{r}_\circ|_{L^2} = \sqrt{\sum \mathbf{r}_\circ}$
$\mathbf{p}_\circ = \mathbf{r}_\circ$
$n_{iter} = 0$
while $n_{iter} < n_{max}$ **do**
  $n_{iter} = n_{iter} + 1$
  $\alpha_n = (\mathbf{r}_n^T \mathbf{r}_n)/(\mathbf{p}_n^T \mathbf{A}\mathbf{p}_n)$
  $\mathbf{u}_{n+1} = \mathbf{u}_n + \alpha_n \mathbf{p}_n$
  $\mathbf{r}_{n+1} = \mathbf{r}_n - \alpha_n \mathbf{A}\mathbf{p}_n$
  $|\mathbf{r}_{n+1}|_{L^2} = \sqrt{\sum \mathbf{r}_{n+1}}$
  if $|\mathbf{r}_{n+1}|_{L^2}/|\mathbf{r}_\circ|_{L^2} < 0$ **then**
    break
  end if
  $\beta_n = (\mathbf{r}_{n+1}^T \mathbf{r}_{n+1})/(\mathbf{r}_n^T \mathbf{r}_n)$
  $\mathbf{p}_{n+1} = \mathbf{r}_{n+1} + \beta_n \mathbf{p}_n$
end while

3 Using the code

Using the 2D solver requires the following files:

- `makefile`
- `main.cpp`
- `postprocess.py`
- `GCSolver.cpp`
- `CGSolver.hpp`
- `COO2CSR.cpp`
- `COO2CSR.hpp`
- `heat.cpp`
- `heat.hpp`
- `matvecops.cpp`
- `matvecops.hpp`
- `sparse.cpp`

- `sparse.hpp`

The data defining the boundary size, discretization, and top and bottom temperatures are read from a .txt file with the format:

```
length width h
Tc Th
```

If the code has not yet been compiled, type in the terminal:
`$make`

To remove all intermediate object files, type:
`$make clean`

To solve the boundary value problem, type:
`$./main <input file> <solution prefix>`

The program writes the solution to a .txt file for the first and last iteration as well as for every tenth iteration of the form `solution000.txt, solution010.txt,` etc. If the problem converges, the console will output the following:
`SUCCESS: CG Solver converged in xxx iterations.`

Next, to post-process the data, use the Python script `postprocess.py` as follows:
`$python3 postprocess.py <input file> <solution file>`

Post-processing generates a temperature pseudocolor plot with an isoline of the mean temperature in the domain. The terminal output is:

```
Input file processed:  <input file>
Mean temperature:  xxx.xxxxx
```

Note: The solution file contains the name of the input file that was read to generate the solution. This is compared to the command line input file, and if the two names do not match, an exception is thrown. Therefore, there is no chance of crashing the program or encountering unexpected behavior if incompatible input and solution files are provided.

4 Results

Figures 1 and 2 were generated using the heat solver program. Figure 1 shows the same domain size and temperatures as the provided file `input1.txt`. The left plot shows a coarse discretization of $h = 0.1$ and the right plot shows a finer discretization of $h = 0.01$. Similarly, Figure 2 is a simulation of the same problem as the provided `input2.txt`, but the left plot has a discretization of $h = 0.05$ and the right of $h = 0.005$. The mean temperature of the four simulations are provided in the table below.

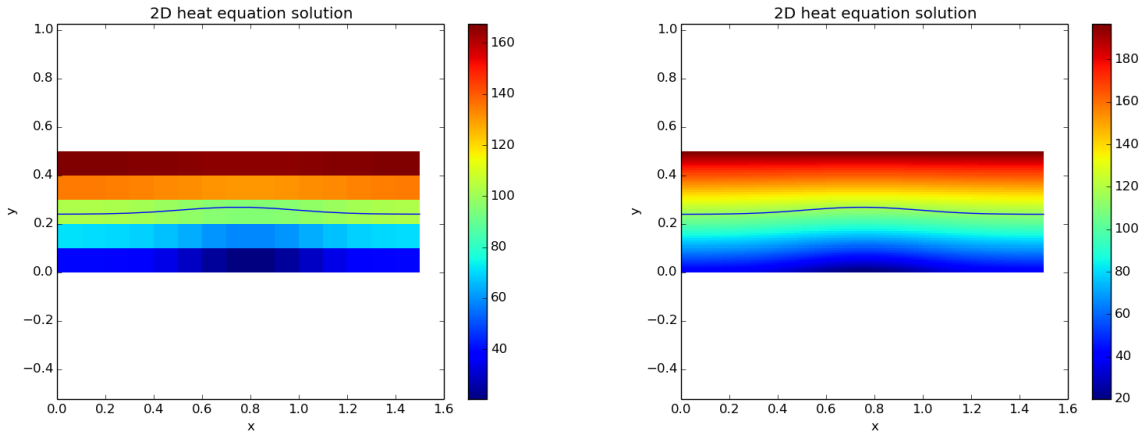

Figure 1: $x = 1.5$, $y = 1.0$, $T_h = 200$, $T_c = 20$
$h_{left} = 0.1$, $h_{right} = 0.01$

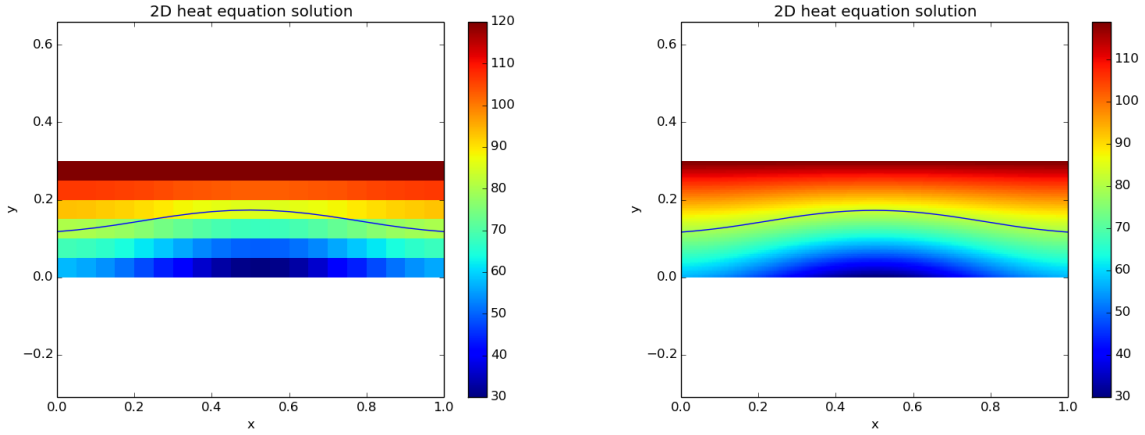

Figure 2: $x = 1.0$, $y = 0.3$, $T_h = 120$, $T_c = 30$
$h_{left} = 0.05$, $h_{right} = 0.005$

	coarse mesh	fine mesh
input1.txt	116.466	116.287
input2.txt	82.092	81.832

Table 1: Mean temperatures for several simulations

References

[1] Henderson, Nick. *CME 211: Project Part 1*, 2016.

[2] Henderson, Nick. *CME 211: Project Part 2*, 2016.