Project - Heat Equation solver

Jesica Leticia Gonzalez Robles

In this project, a heat equation solver was implemented through the finite differences method and the Conjugate Gradient algorithm.

The physical system of study was a pipe within which hot and cold fluids were transferred with temperatures T_h and T_c, respectively. The goal was to determine the temperature distribution within the pipe walls. A simplified 2D geometry was used to model the pipe walls, defined by an equally spaced cartesian grid.

The project was divided into two main parts:

1. Linear system of equations solver (C++)
2. Heat equation solver with boundary conditions using an Object Oriented Programming (OOP design) and solution post-processing (Python3).

1 Conjugate Gradient Solver

A solver for the system of equations defined as $Ax = b$ was solved in C++ from a given matrix A in CSR format (sparse matrix) using the *Conjugate Gradient (CG)* algorithm (see Algorithm 1 pseudocode).

Algorithm 1 Conjugate Gradient's Algorithm

```
procedure CGSolver($A, \vec{x}, tol$)
    $\vec{x}_0 = 1; \vec{b} = 0$                         ▷ Initialize $x_0$ and b. b does not change.
    $\vec{r}_0 = \vec{b} - A\vec{x}_0$
    $L2normr0 = L2norm(\vec{r}_0)$
    $\vec{p}_0 = \vec{r}_0; niter = 0$
    while $niter < niter_{max}$ do                     ▷ Maximum number of iterations is not reached.
        $niter = niter + 1$
        $\alpha_n = \vec{r_n}^T \vec{r_n} \vec{p_n}^T A \vec{p_n}$
        $x_{n+1} = x_n + \alpha p_n; r_{n+1} = r_n - \alpha A p_n$
        $L2normr = L2norm(\vec{r}_{n+1})$
        if L2normr / L2normr0 < tol then                ▷ Convergence is reached.
            break
        $\beta_n = \vec{r}_{n+1}^T \vec{r}_{n+1} / \vec{r}_n^T \vec{r}_n$
        $p_{n+1} = r_{n+1} - \beta_n p_n$
    return $niter$                  ▷ Returns $niter$ if the algorithm converged or $-1$ if not.
```

The implementation is divided into the following files: *main.cpp*, *CGSolver.cpp*, *COO2CSR.cpp*, and *matvecops.cpp*, with the corresponding HPP files for the last three. In addition, a *makefile* is provided to compile and execute the program.

The *matvecops.cpp* file contains useful functions to perform common matrix and vector operations, such as: substraction of two vectors, addition of two vectors, product of an scalar times a vector, transpose of a vector (expressing it in CSR format) and the product of a matrix in CSR format times a vector. The transpose of a vector was set to return the result in CSR format for it to be useful later on with the first implemented CSR matrix -vector function, since a vector could be seen as a $nx1$ matrix. In addition, an L2norm function is implemented to compute the norm of a vector, given as the square root of the dot product of to vectors ($\sqrt{\vec{r}^T \cdot \vec{r}}$).

The *COO2CSR.cpp* file transforms a matrix A in the usual COO format to CSR format. This is useful to operate an initial given matrix, which contains all the system of equation values (derived from the finite differences method). This file was given by the CME211 teaching team. [1]

The *CGSolver.cpp* file contains the implementation of the *CG* algorithm for the system $A{*}x = b$. It initializes the x solution's vector with ones and the b vector with zeros. Then it computes an initial vector r_0 and its L2norm, which will be updated according to the *CG* algorithm described above. It is important to notice that each matrix-vector or vector-vector product is done separately before performing further operations, for example, for β and α, to avoid problems related to the resulting vector/matrix dimensions (recall that matrix- matrix and matrix-vector product does not commute i.e. $Ax \neq xA$). Finally, the function returns the total number of iterations that are needed to reach convergence for the given tolerance value. Otherwise, the function returns -1 if the algorithm doesn't converge.

The *main.cpp* is executed with two arguments provided by the user in the terminal: `$./main matrix.txt solution.txt$`. This file reads the given matrix file, converts it to CSR format and executes the *CGSolver* function. Finally, it stores the solution vector in the specified file and shows the final number of iterations to reach convergence.

2 Finite Differences & Post Processing

The goal of the second part of this work was to assemble the system of equations derived from the discretization of the 2D steady state heat equation for a given geometry. The discretization derived in one equation per grid point, assembled in the A matrix. The boundary conditions used were: cold isothermal boundary for the lower part of the grid (last row), periodic boundary conditions for the sides of the grid, and hot isothermal boundary conditions for the top of the grid (first row).

The OOP implementation is divided into the following files:
main.cpp, *CGSolver.cpp*, *COO2CSR.cpp*,*matvecops.cpp*, *heat.cpp*, and *sparse.cpp*, with the corresponding HPP files for the last three. In addition, post-processing script *postprocess.py* is provided to read an input and a solution file, plot the solution vector, and the temperature distribution within the pipe. Finally, a *makefile* is provided to compile and execute the program.

Two classes were created: `SparseMatrix` and `HeatEquation2D`. The `SparseMatrix` class defines a sparse matrix object, which data attributes correspond to the vectors defining the matrix. The methods contained in this class perform the necessary tasks: setsize, resize, convert COO to CSR format, add entries, print matrix, and compute the product of a matrix times a vector. The `HeatEquation2D` class creates a SparseMatrix object, contains the data needed to form the system of equations from a given file, solves the system using the *CGSolver* and returns the solution vector in a .txt file every 10 iterations.

Note: The `Setup` method in `HeatEquation2D` creates a matrix in COO format, excluding the points at the upper and lower boundaries, as well as the right side boundary points since these last are set equal to the ones at the left boundary. Also, the matrix A and solution vector x are ordered by row, meaning that per each column, the unknowns are added to the matrix and vector in ascending order of the rows (top to buttom). A *map* structure was used to relate the position indexes on the grid to the position of the unknowns in the solution vector. Values/variables such as the vector b, the number of equations, rows, columns, order, etc., were passed by reference as `const` to avoid accidental modification of its values.

3 User Guide

3.1 Compiling code

In the terminal, go to your working directory and type:

```
$ make clean // Removes all object file , editor files and main
$ make // Compile all source code and produce main
```

3.2 Running Heat Equation Solver

Input

```
$ ./main input#.txt solution
```

Expected output:

```
SUCCESS: CG solver converged in xx iterations.
```

3.3 Post processing

Input

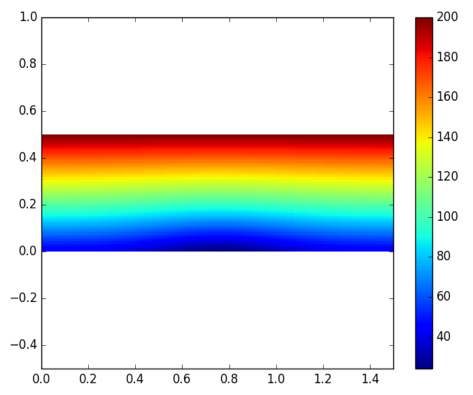
```
$ python3 postprocess.py input1.txt solution<#>.txt
```

Expected output:

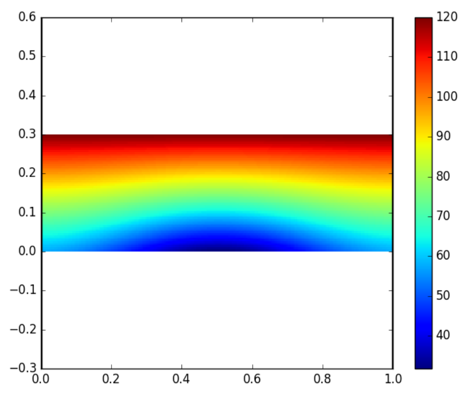
```
    Input file processed: input1.txt
    Mean Temperature: xx.xxxxx
```

4 Visualization

The *postprocess.py* file generates a plot of the temperature distribution within the pipeline and saves it as a PNG file with the name *"Temperature_ Distribution.png"*. The name can be modified at the end of the `Plot_Solution` function according to your needs.



(a) Temperature distribution for input1.txt (size = 150 x 50). Convergence reached in 129 iterations.



(b) Temperature distribution for input2.txt. (size = 200 x 60) Convergence reached in 154 iterations.

Figure 1: Example outputs for *postprocess.py*.

References

1. LeGresley, P. *CME211: Project Part 1*. Modified for CME211 Fall 2022 by the teaching staff.

1. LeGresley, P. *CME211: Project Part 2*. Modified for CME211 Fall 2022 by the teaching staff.