18660 Project 1: 2-D Thermal Analysis

Akshay Rajhans
arajhans@ece.cmu.edu

Abstract –

In this project, we numerically solve for the steady-state temperature of a panel using 2D thermal analysis. We discretize the panel into a number of small areas and formulate the heat equations for the temperatures of these areas as a linear system of equations by approximating the Laplacian operator for the temperature evolution into a second-order linear approximation. The positive-definite symmetric diagonally dominant matrix "A" in the system of equations Ax = b is then factorized into a lower triangular matrix "L" and its transpose L' such that L x L' = A (Cholesky decomposition). Once we have the matrix L, the system Ax = b can be solved in two steps by solving equations Ly = b and L'x = y. Since L (and L' respectively) is a lower (upper) triangular matrix, the equation can be solved using forward (backward) substitution.

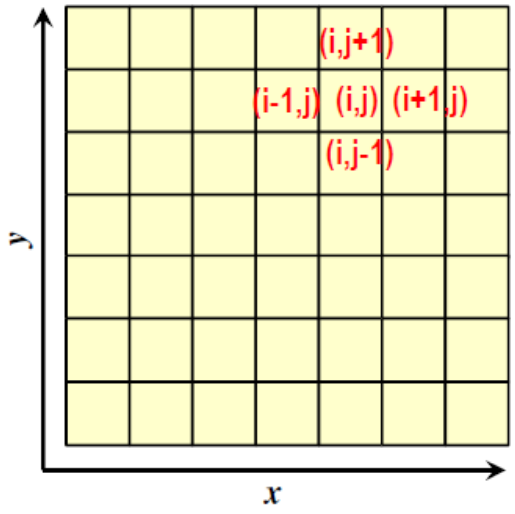
We implement this in Matlab, while trying to minimize the 'for' loops and trying to vectorize the operations as much as possible. As a part of discussion of the accuracy of the results, the results obtained are compared with the results given by standard matlab functions for solving a linear equation, i.e. the backslash command and the Cholesky decomposition function 'chol'.

1. Equation Formulation

The steady-state heat equation is given by

$$\kappa \cdot \nabla^2 T(x,y) + f(x,y) = 0$$

The panel for which we are doing the thermal analysis is discretized into many small rectangles as shown in the given figure. The temperature of each panel is an unknown variable we would like to find the steady-state value of.



For each panel, we have the following equation, where i is the index of the panel in the x direction and j is the index in the y direction.

$$\kappa \cdot \left[\frac{\partial^2 T(i,j)}{\partial x^2} + \frac{\partial^2 T(i,j)}{\partial y^2} \right] = -f(i,j)$$

The partial derivative operations in the Laplacian operator can be approximated by the second order linear approximations

$$\frac{\partial^2 T(i,j)}{\partial x^2} = \frac{T_{i+1,j} + T_{i-1,j} - 2T_{i,j}}{\Delta x^2}$$

$$\frac{\partial^2 T(i,j)}{\partial y^2} = \frac{T_{i,j+1} + T_{i,j-1} - 2T_{i,j}}{\Delta y^2}$$

This gives us the following system of equations, one per panel.

$$\kappa \cdot \left[\frac{T_{i+1,j} + T_{i-1,j} - 2T_{i,j}}{\Delta x^2} + \frac{T_{i,j+1} + T_{i,j-1} - 2T_{i,j}}{\Delta y^2} \right] = -f(i,j)$$

$$1 \le i \le N, 1 \le j \le M$$

Here, we observe that the temperature of each panel contributes to its neighboring panels in the x and y directions. We define new constants $\alpha = \kappa/\Delta x^2$ and $\beta = \kappa/\Delta y^2$.

The size of the panel Δx and Δy is calculated by dividing the x and y dimensions of the panel (given by variables `mediumX` and `meduimY`) by the number of panels in the x and y directions, here N and M respectively (and in the code `nX` and `nY`).

Formulation of the "A" matrix:

Our final goal is to solve the steady-state value of the temperatures using linear equation solving. Therefore, we convert the temperatures in this matrix format, into a vector format. In principle, this is like using the 'reshape' command in Matlab, however, we do not need to store the unknowns. Once we have converted the temperatures from a matrix to a vector, the observation that each temperature contributes to the temperature at +-1 entries in x and y direction needs to be modified. In the vector case now, each temperature contributes to the +/- 1 entries, which are the x-neighbors and +/- N entries, which are now the y-neighbors.

For panels inside boundaries, we fill out the A matrix row-by-row. For ith row, the diagonal (ith) entry is $2(\alpha+\beta)$, while the i+/-1st entry is $-\alpha$ and the i+/-Nth entry is $-\beta$. For panels at boundaries, one (two in case of the four corners) neighbors are actually the boundary conditions. In such cases, the addition of $-\alpha$ or $-\beta$ is simply skipped.

Formulation of the "b" vector:

The b vector is the vector of the inputs to the panels, given by the p matrix. Here, we use the reshape command to turn the p matrix into a vector.

For panels close to boundaries, we need to update the boundary conditions. Therefore, we add $\alpha.T_c$ and $\beta.T_c$ (where T_c is the temperature of the boundary condition converted to Kelvin) to the corresponding values in the p matrix. This forms the updated "b" vector.

2. Cholesky Factorization

Cholesky factorization works by turning a positive definite symmetric "A" matrix into a product of a lower triangular matrix L and its transpose L'.

Unlike the technique shown in class, which focused on solving all the entries of the L matrix one by one, a significant amount of speedup can be achieved by computing the values of one column at a time. This is one of the highlights of this project.

To compute the entries of the L matrix one column at a time, we iterate over the index of the columns, `nCol`, starting with the first column. The diagonal entry of the first column is given by:

```
d = sqrt(A(nCol,nCol));
```

whereas, the lower entries in the first column are given by:

```
L(nCol+1:end,nCol) = A(nCol+1:end,nCol)/d;
```

This is because the first column of `L` gets multiplied only by the first entry l_{11} to form the first column of `A`.

The second column onwards, the entries of L start depending on the columns to the left of the column under consideration. In this case, the diagonal entry still has a special structure, and is given by:

```
d = sqrt(A(nCol,nCol)-(L(nCol,1:nCol-
1)*L(nCol,1:nCol-1)'));
```

This captures the said effect of the dependence on the columns `1:nCol`.

The lower entries are given by:

```
L(nCol+1:end,nCol) = (A(nCol+1:end,nCol)-
(L(nCol+1:end,1:nCol-1)*L(nCol,1:nCol-
1)'))/d;
```

This captures the dependence on the columns `1:nCol`.

Forward and backward substitution:

Once we obtain the L (and in turn L') matrix, we can use it to solve for the temperature vector. This can be done in two steps. We substitute A = L.L' in the equation Ax = b. This gives us the equation L.L'x = b. We can substitute another vector y such that L'x = y and then we get L.y=b. Since L and L' are lower and upper triangular matrices respectively, these two equations can be solved relatively easily.

Forward substitution to get y from the equation L.y=b:

In case of the first row, there is just one element in the matrix L. Therefore, the first entry of y can be given by:

```
y(nRow) = b(nRow)/L(nRow,nRow);
```

For the rest of the entries in the y vector, we iterate over the row index from 2 to the size of y. For these entries, we can subtract the contribution due to the elements to the left of the diagonal entry in each row and divide it by the diagonal entry of L. This gives the values of the y vector 2 through size of y. The vectorized command for this operation is:

```
y(nRow) = (b(nRow) - (L(nRow,1:nRow-
1)*y(1:nRow-1)))/L(nRow,nRow);
```

Backward substitution to get x from the equation L'x = y:

Here, the idea is similar to that of the forward substitution, however, we need to do it in the reverse order – the last entry is the easiest to solve. For this last entry, since there is only one element in the last row of L', we can get the last value in the x vector as follows:

```
x(nRow) = y(nRow)/L(nRow,nRow);
```

In case of the remaining entries, i.e. 1 through size of b – 1, we subtract the contribution of all the entries to the right of the diagonal entry in L' and divide by the diagonal entry of L'. The contribution by the entries to the right of the diagonal one is that of the entries in x vector below the one we are trying to solve. The vectorized command for this operation is given by:

```
x(nRow) = (y(nRow) -
(L(nRow+1:N,nRow)'*x(nRow+1:N)))/L(nRow,nRo
w);
```

The x vector we get at the end of the backward substitution is the temperature vector. We reshape this into a matrix to get it into the original format by using the 'reshape' command. Further, we convert it from Kelvin to degree Celsius. This forms the matrix 'temperature' in the requested format, of the same size of 'p', ready to be plotted using 'thermalplot'.

3. Experimental Results

Following are the results returned by the program. The three plots depict the temperatures.

For some reason, I seem to be getting wrong values of temperature from the program. I compared the results I get with the backslash command of Matlab, and they seem to match. There is possibly some error in the problem formulation.

The plots I get are attached below.

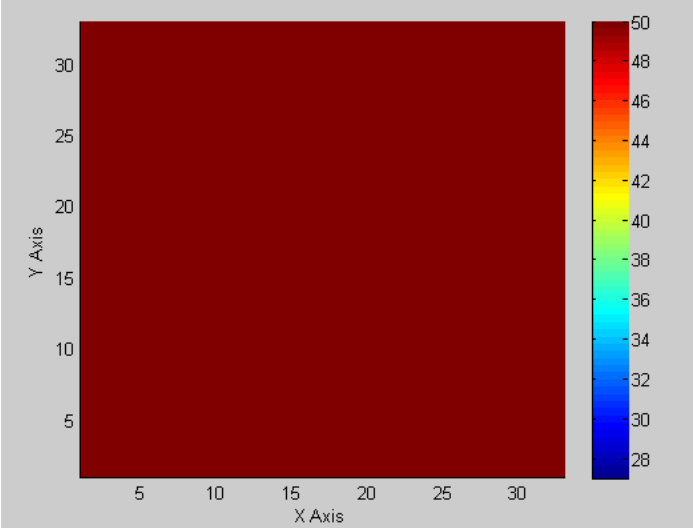


Figure 1: Steady-state temperature plot for case 1

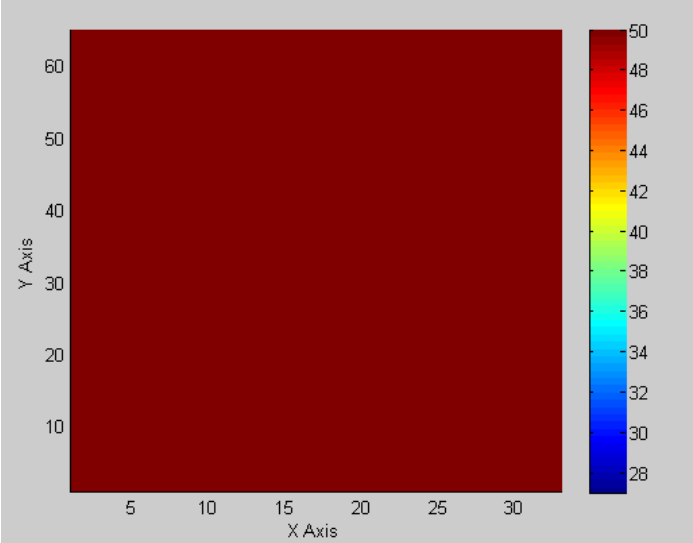


Figure 2: Steady-state temperature plot for case 2

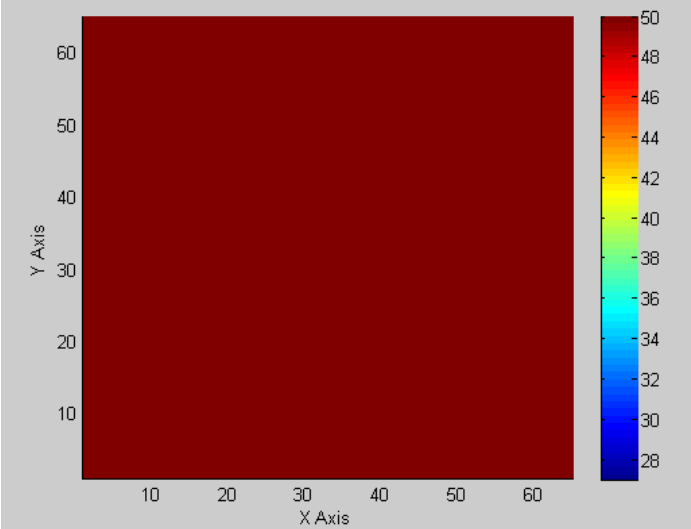


Figure 3: Steady-state temperature plot for case 3

The following table summarizes the computational time required by the program for the three cases. These were evaluated in Matlab using the tic and toc commands.

Computational times:

Case 1	2.9082 seconds
Case 2	23.293 seconds
Case 3	185.02 seconds

4. Discussion

Accuracy and efficiency the program can achieve:

In order to analyze the accuracy of the result returned by the program, we compared the max of the absolute error between our solution and the solution returned by the backslash command. The error seems to be significantly small which implies that the program is performing fairly accurately.

Case 1	1.4552e-011
Case 2	8.3674e-011
Case 3	5.0932e-011

Table 1: Max of absolute error between any entry of the temperature vector returned by the program and by the backslash command

In order to analyze the accuracy of the Cholesky decomposition performed by the program, we again sought the max of the absolute error made by the entries in the L matrix computed by this program compared to the L matrix returned by the 'chol' command in Matlab. The results are summarized in the following table.

Case 1	6.3665e-012
Case 2	1.819e-011
Case 3	4.0927e-011

Table 2: Max of absolute error between any entry of the L matrix returned by the program and by the chol command

As seen in the table, the error made is extremely small. Therefore, the program seems to be doing a good job.

Size of problem your program can handle:

This program can handle all three sizes of the cases given in the problem set. The computation time required seems to grow with the size of the panel, but this is expected. Given a panel of MxN elements, we need to formulate the "A" matrix that has $(MxN)^2$ entries.

Possible improvements that can be done:

1. Bug resolution: There is some mistake in the problem formulation, which is giving me wrong results. I cannot seem to be able to find out where the bug is. But that is the improvement that is needed.
2. Exploiting sparsity: Currently, the code does not exploit the fact that the A matrix is mostly sparse. This will probably provide the most speed-up for the third case, which has the largest variable size to solve.

Anything unique you have done to improve/validate your program's accuracy/efficiency:

The most important unique improvement in this code is that the 'for' loops have been kept at a minimum and the code is vectorized, optimizing its speed and efficiency for the for matrix operations.

References

[1] Class slides – lecture # 4, Xin Li

[2] Project 1 handout slides – Wangyang Zhang