Project 1: 2-D Thermal Analysis

<Jin Gu>
<jingu@andrew.cmu.edu>

Abstract

In this project, given the boundary temperature of a 2D space and the heat source, we need to find out the temperature in this space. Here we can use a 2D PDE equation to show the relationship between each point in the space, which is that

Density — Laplace operator

$$\rho \cdot C_p \cdot \frac{\partial T(x,y,t)}{\partial t} = \kappa \cdot \nabla^2 T(x,y,t) + f(x,y,t)$$

Thermal capacity — Thermal conductivity — Heat source

To solve this equation, there are two methods we can use. One is Gaussian elimination and another is Cholesky factorization. Therefore, this project mainly contains the content of how to solve this 2D PDE equation with two methods mention above in Matlab.

1. Mathematical Formulation

In the system of linear equations, we can use Ax = b to represent the system.

With 2D PDE equation, we can get the relationship between each point in the space, which is

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

Then we can change it to

$$\frac{f(i,j)}{k} = 2\frac{\Delta x^2 + \Delta y^2}{\Delta x^2 \Delta y^2} T_{i,j} - \frac{T_{i+1,j}}{\Delta x^2} - \frac{T_{i-1,j}}{\Delta x^2} - \frac{T_{i,j+1}}{\Delta y^2} - \frac{T_{i,j-1}}{\Delta y^2}$$

Because we are given the boundary temperature, when i = 1, $T_{i-1,j}$ = temperature of bottom bound and $T_{i+1,j}$ = temperature of top bound. When j = 1 and M, $T_{i,j-1}$ = temperature of left bound and $T_{i,j+1}$= temperature of right bound. Because we want to get the temperature of each point, we can use $T_{i,j}$ to form our parameters matrix x, which can be

x = $[T_{1,1}\ T_{1,2}\ldots\ldots T_{2,1}\ T_{2,2}\ldots\ldots\ T_{N,M-1} T_{N,M}]^T$

Output matrix b, which can be

b=$[\frac{f(1,1)}{k} + T_{2,1} + T_{0,1}\ \ \frac{f(1,2)}{k} + \ T_{0,1}\ldots\ldots\ \frac{f(N,M)}{k} + T_{N+1,M} + T_{N,M+1}]^T$

Coefficient NM x NM matrix A, which consists of two kind of M x M sub matrix. One is a matrix which all diagonal element is $2\frac{\Delta x^2 + \Delta y^2}{\Delta x^2 \Delta y^2}$ and the two elements beside diagonal element $-\Delta y^2$. Another is a matrix which its diagonal element are $-\Delta x^2$. Suppose we have a N x N matrix K, then we put the first sub matrix into the diagonal position and the second sub matrix into two position beside the diagonal position, finally we put a M x M zero matrix into all left positions. In this way, we get an NM x NM matrix and this matrix is matrix A.

2. Linear System Solver

After we get the matrix representation of system Ax = b, we can use Gaussian elimination and Cholesky factorization algorithm to solve the system of linear equations.

2.1 Gaussian elimination

2.1.1 Change the augmented matrix B to be upper triangle matrix

In Gaussian elimination, we first need to get the augmented matrix B, here B = [A b]. Then we change B to be upper triangle matrix. Here is what I do. First I loop into a column, and I check the elements below the diagonal element to see if they are zero, if not, I use the diagonal element to make them to be zero. I recursively do this column by column. In this way, we finally get an upper triangle matrix.

2.1.2 Solve the upper triangle matrix from top to bottom

After we get the upper triangle matrix, we can use it to solve the parameters. Here is what I do.

```
n = size(b, 1);
X(n) = b(n) / A(n,n);
for i = n - 1 :-1: 1
    X(i) = (b(i) - A(i, i+1:n ) * X(i+1:n)) / A(i,i);
end
```

2.2 Cholesky factorization

2.2.1 Get an lower triangle matrix L

In Cholesky factorization algorithm, we first need to decompose A into L*L^T. Therefore, we should get a lower triangle matrix L. I use two equations to calculate the matrix L, which are that

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k} L_{j,k}^*}.$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}^* \right), \qquad \text{for } i > j.$$

2.2.2 Solve the system of linear equations with matrix L

After we get the matrix L, equations system Ax = b can be change into $L*L^Tx = b$. Let's think $L^T*x = y$ and L*y = b. because L is lower triangle matrix, therefore L^T is upper triangle matrix. In Gaussian elimination, we already know how to solve Ax = b, if A is upper bound matrix. We can solve it in the same way even A is lower triangle matrix. Here is what I do.

```
y(1) = b(1) / L(1, 1);
for i = 2 : n
    y(i) = (b(i) - L(i, 1:i-1 ) * y(1:i-1)) / L(i,i);
end
```

The only difference is that we solve it from bottom to top. In this way, we can solve the equations system.

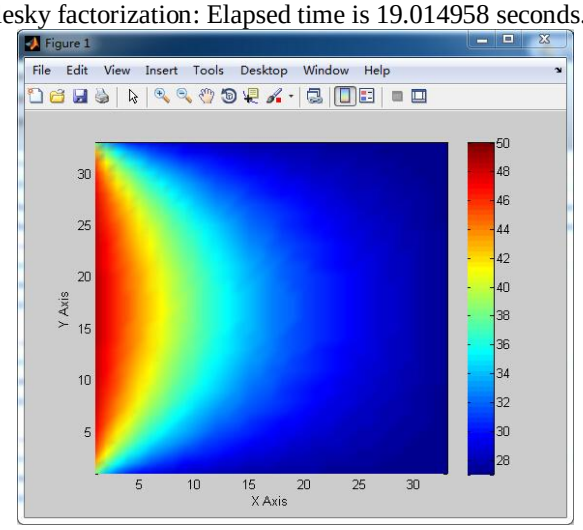
3. Experimental Results

Attach the thermal plots for each test case. Please also show the simulation time of your program for each test case.

Case 1:

Gaussian elimination: Elapsed time is 9.383435 seconds.

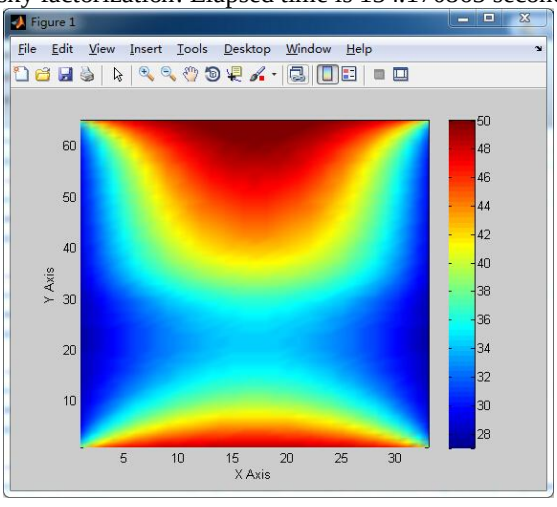
Cholesky factorization: Elapsed time is 19.014958 seconds.



Case 2:

Gaussian elimination: Elapsed time is 40.745665 seconds.

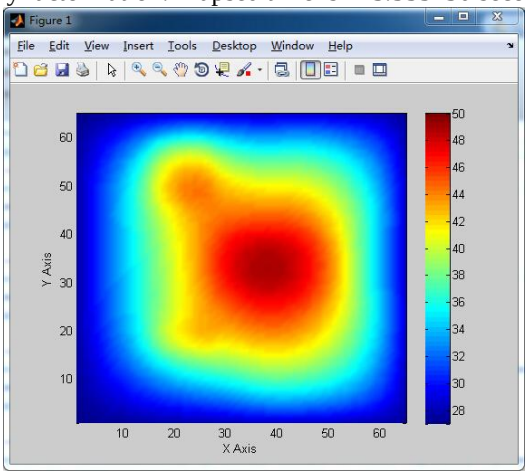
Cholesky factorization: Elapsed time is 154.170865 seconds.



Case 3:

Gaussian elimination: Elapsed time is 236.909110 seconds.

Cholesky factorization: Elapsed time is 725.333250 seconds.



4. Discussion

4.1 Accuracy and efficiency your program can achieve

Error: case 1(compared to golden solution)

Gaussian elimination: 6.2289e-14

Cholesky factorization: 2.8471e-14

case 2

Gaussian elimination: 2.6415e-13

Cholesky factorization: 8.9668e-14

case 3

Gaussian elimination: 1.3026e-13

Cholesky factorization: 1.4207e-13

Efficiency

From the result pictures, it is easy to see that the project get a quite good result we wanted.

4.2 Size of problem your program can handle

In my solver, it can solve problem of any size, but as the size increases, the efficiency will be lower and lower.

4.3 Possible improvements that can be done

In my project, after comparing the result with the method using backslash "\", I found there is not too much space for us to improve the accuracy. Therefore, I think we can improve the efficiency of the project because it only takes several minutes to solve the problem with backslash "\".

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

In this project, I did several things to speed up the procession.

1 use matrix calculation to replace point calculation. From example, when we solve the problem Ax = b if A is upper triangle matrix, we can use matrix calculation to find out the answer. In this way, we can solve it with one loop instead of two.

2 check the rank of matrix to see if we can solve the problem with Gaussian and Cholesky. In this way, we can improve the accuracy

3 calculate the constant term before to avoid calculate them repeatedly. For example

```
deltaX = mediumX / N;
deltaY = mediumY / M;
sX = deltaX ^ 2;
sY = deltaY ^ 2;
sumXY = sX + sY;
mulXY = sX * sY;
four = (2 * sumXY / mulXY) * eye(M, M);
one = (-1 / sX) * eye(M, M);
zero = zeros(M, M);
```

4 pre-allocate the space before to avoid allocate them again and again

```
n = size(A, 1);
L = zeros(size(A));
```

5 use the feature of matrix to speed up the project. For example, in Gaussian elimination, I found that there are many elements which are zero already, therefore put a condition before we change the element to zero, which is that

```
for i = 1 : n - 1  %change the matrix to be a upper triangular matrix
      for j = i + 1 : n
            if B(j, i) ~= 0
            coe = B(j,i) / B(i,i);
            B(j,i:n+1) = B(j,i:n+1) - coe * B(i,i:n+1);
            end
      end
end
```

I found it speeds up my project significantly. If I don't use this optimization, my project needs more than 30 minutes to solve the problem in case 3 with Gaussian elimination.

References

Crouse PPT