The following is based on the homework assignment requirement:

1. GITHUB

I create a public account with the username of **dchen0215**.

2. CODE

a. The task of assigning values to the variables is put in the top of the program to simplify the changing process.

b. The **physical domain and partitioning** is not fixed upon requirement.

c. Some functions from "nrutil.c" are used in this program to allocate 3D float matrix, 2D float matrix and float 1D vector.

d. **FTCS** is pretty straightforward and nothing nontrivial. I write the result into the output text file every 1000 time steps, which is used for drawing graphics in Matlab. The slice in the middle of the 3d matrix is the most typical part, i.e. the index of z is equal to nz/2+1, so I only plot that for simplicity.

Below is **a series of plots** to represent the temperature evolving process. It is broken into 10 steps. *(Please note that the scale has been changing)*

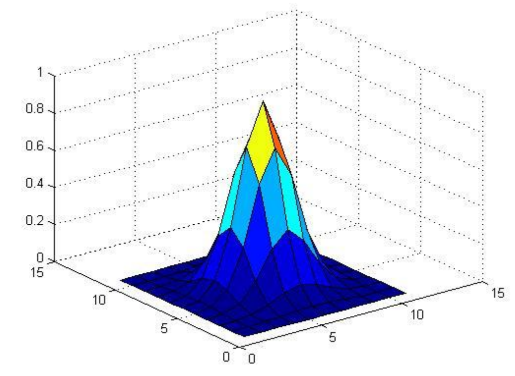


Figure 1

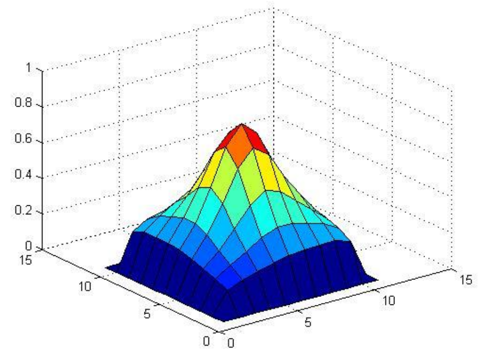


Figure 2

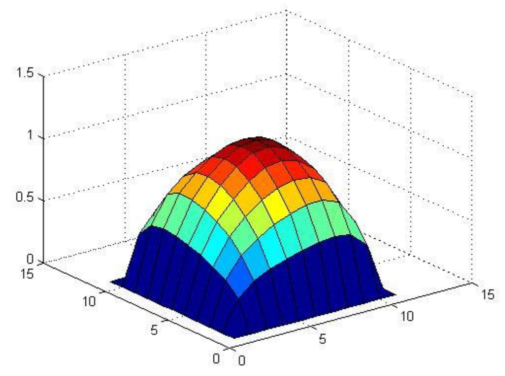


Figure 3

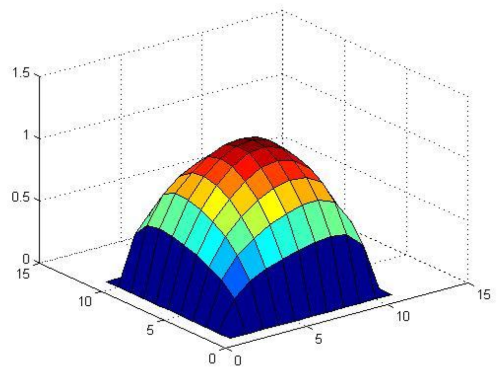


Figure 4

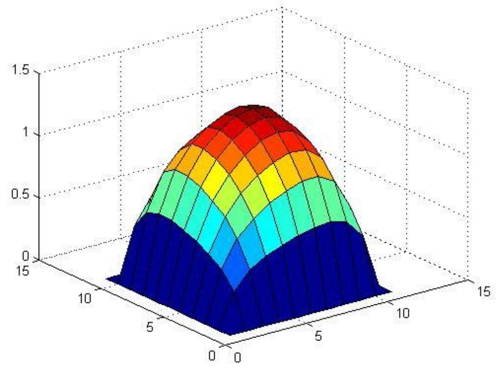


Figure 5

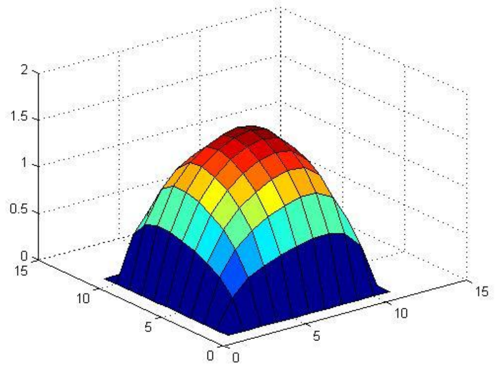


Figure 6

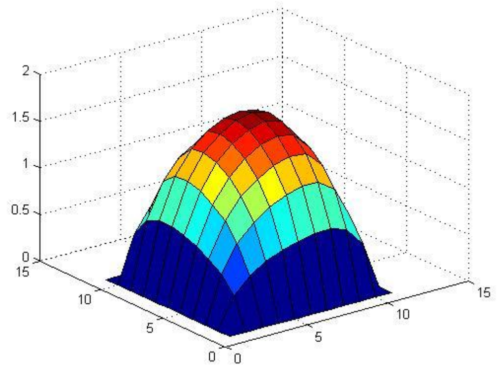


Figure 7

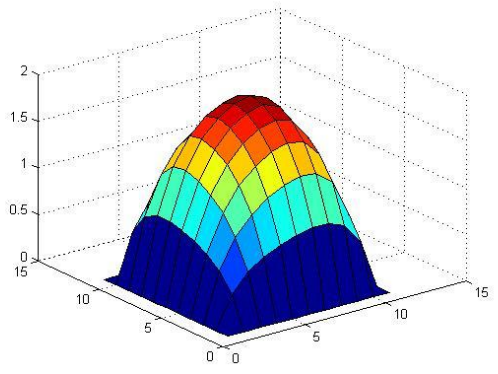


Figure 8

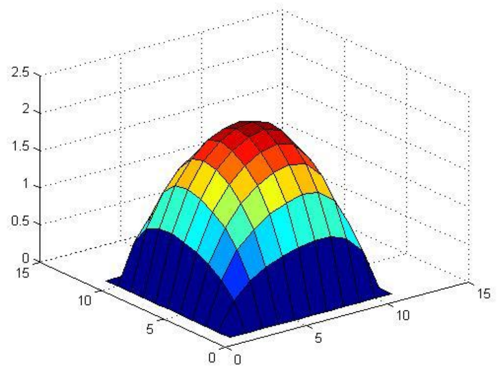


Figure 9

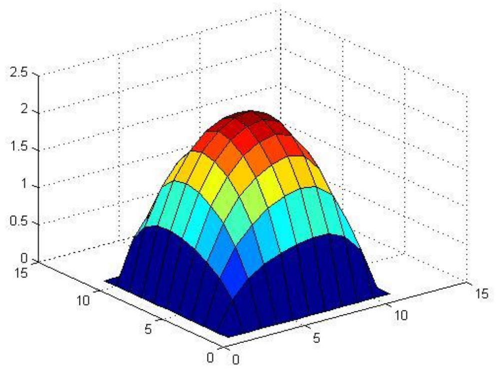
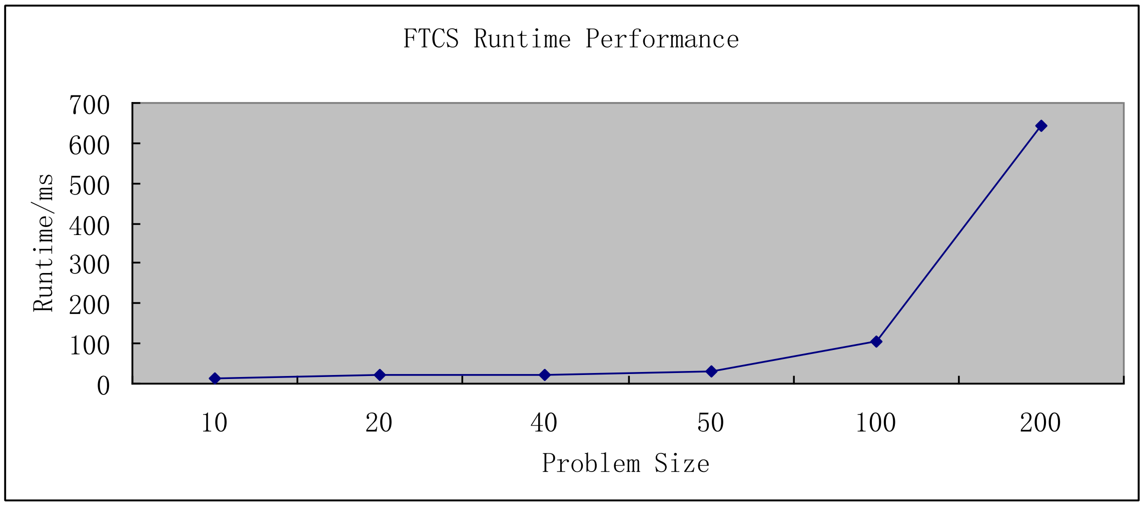


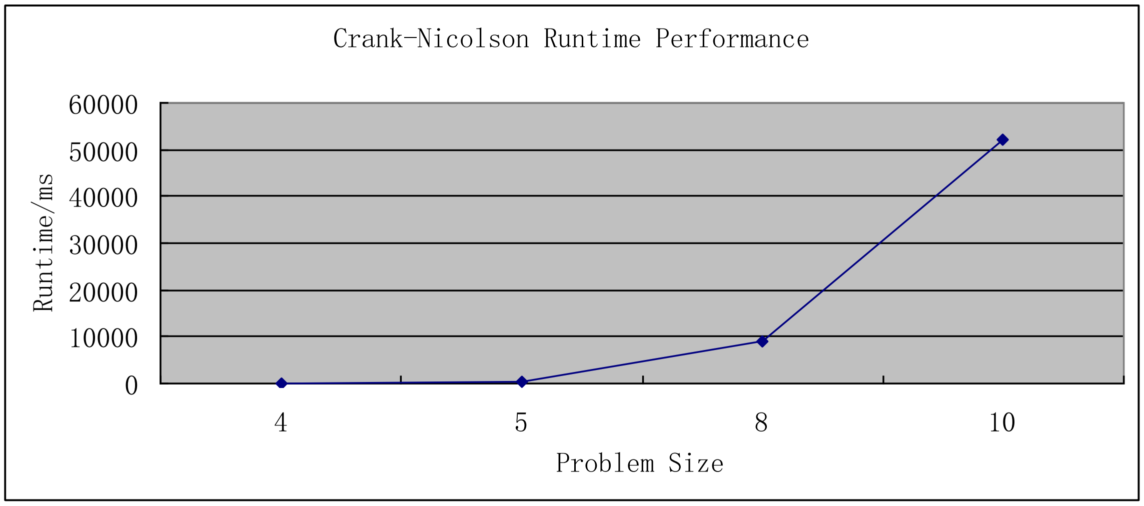
Figure 10

Source term is added to the system. So, generally, the temperature is increasing. However, the change in the appearance of the plot has shown that the heat is diffusing. <u>In the text file, all the matrices are connected together and not separated, because this would be easier for Matlab to read data in.</u>

Crank-Nicolson is implemented in a less specific way, just for educational purpose. The method for solving the linear equation is Gaussian elimination. It is just clearer and

easier to implement, since it is not the most important point in this homework.
e. About the transformation between vector and matrix, I write two functions to make this happen, vectorToMatrix and matrixToVector.
I tried to implement **Alternating Direction Implicit (ADI) Crank-Nicolson** Method based on the reference sent out, but I still got some trouble when running it and there is not enough time for me to debug, because I have a mid-term tomorrow. So, I just leave it there with my total understanding of the algorithm. Hope I can work it out after tomorrow.
f. **Initial condition**: before calculating using FTCS, ADI or CN, **Gaussian noise** is done to the blank 3d matrix to assign the initial condition.
g. **Boundary conditions**: both ways are implemented, but only **Dirichlet** condition is used, because it's more trivial.
h. Below are two plots detailing the runtime performance of the two algorithms. Given the CN here didn't use any fancy method to solve the linear system, it seems pretty slow and not suitable for large problem size.


FTCS Runtime Performance
Runtime/ms
Problem Size


Crank-Nicolson Runtime Performance
Runtime/ms
Problem Size