Joon Ki Hong
MATH 2605 Project 2

Project 2:

For the steepest descent project, I decided on using Java as my primary language. I created 4 classes (SteepestDescentMain, SteepestDescentImpl, Vector, and Function). I also utilized the Matrix class from the JAMA package to my advantage. The SteepestDescentMain class basically withholds the main method that prints all of the results to the console. The SteepestDescentImpl class handles the calculation of the steepest descent algorithm. The Vector class represents a simple vector and handles all calculations regarding vectors such as dot products, multiplication, and addition. The Function class represents the multivariable function F(x) =.5* $x \cdot Ax - b \cdot x$. In the main method, all of these classes come together to print out solutions to the equation Ax=b.

The two equations in problem 1 were approached iteratively. To achieve values of x naught for the level curve of each respective function, I first generated the level curves at z=0 through an external applet found here: http://www.slu.edu/classes/maymk/banchoff/LevelCurve.html. The respective graphs generated are found in the folder provided or at the bottom of this document. I used these graphs to get a general idea of which x naught values I should use. I also used an alternative method used in my code. I manually, by pencil, worked out the function defined above and in the documentation to give me this equation: $2z = ax^2+bxy+cxy+dy^2-2wx-2vy$. The (a,b,c,d) represent individual values in a matrix A, where as the w and v represent the vector b, where w is the x component and v is the y component. Using the value 0 for z, I computed the level curve for each function. I then iterated through every possible combination of x and y with a domain of $-50<x<50$ and $-50<y<50$. I incremented the x and y values by 0.25. While incrementing through these values, I checked whether or not these x and y values satisfied the equation I manually solve for. If the equation was satisfied then it meant that the x and y pair was on the level curve. The program prints these pairs in the console.

I then solved matrix using the steepest descent algorithm method found in the SteepestDescentImpl class and the found x naught values. I also kept track of how many steps the calculation took. I then printed the solution calculated. The 10x10 matrices were generated using a random number generator. I used an arbitrary x naught vector of ten one's. I then printed the results onto the console.

Problem 1

After calculating the x naught values that satisfied the level curve at z=0 for each function, I inputted those values into the steepest descent algorithm. This is the result:

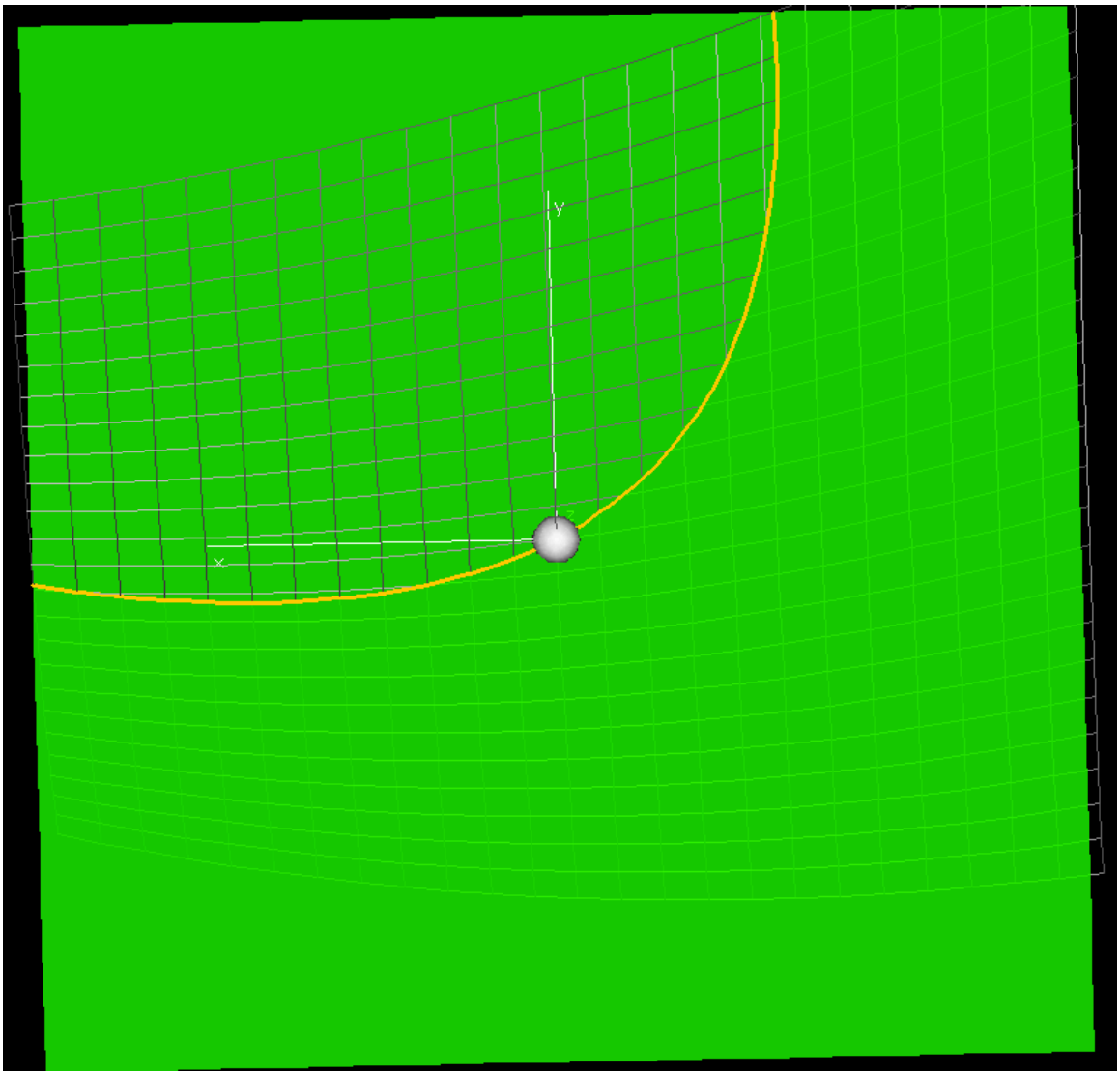
```
--------------------------------------------------------------------------------
Integer Coordinates that mathematically satisfy the Level Curve z=0
(-2.0,4.0)(-2.0,6.0)(0.0,0.0)(0.0,2.0)
The calculations took 13 steps
Matrix:
```

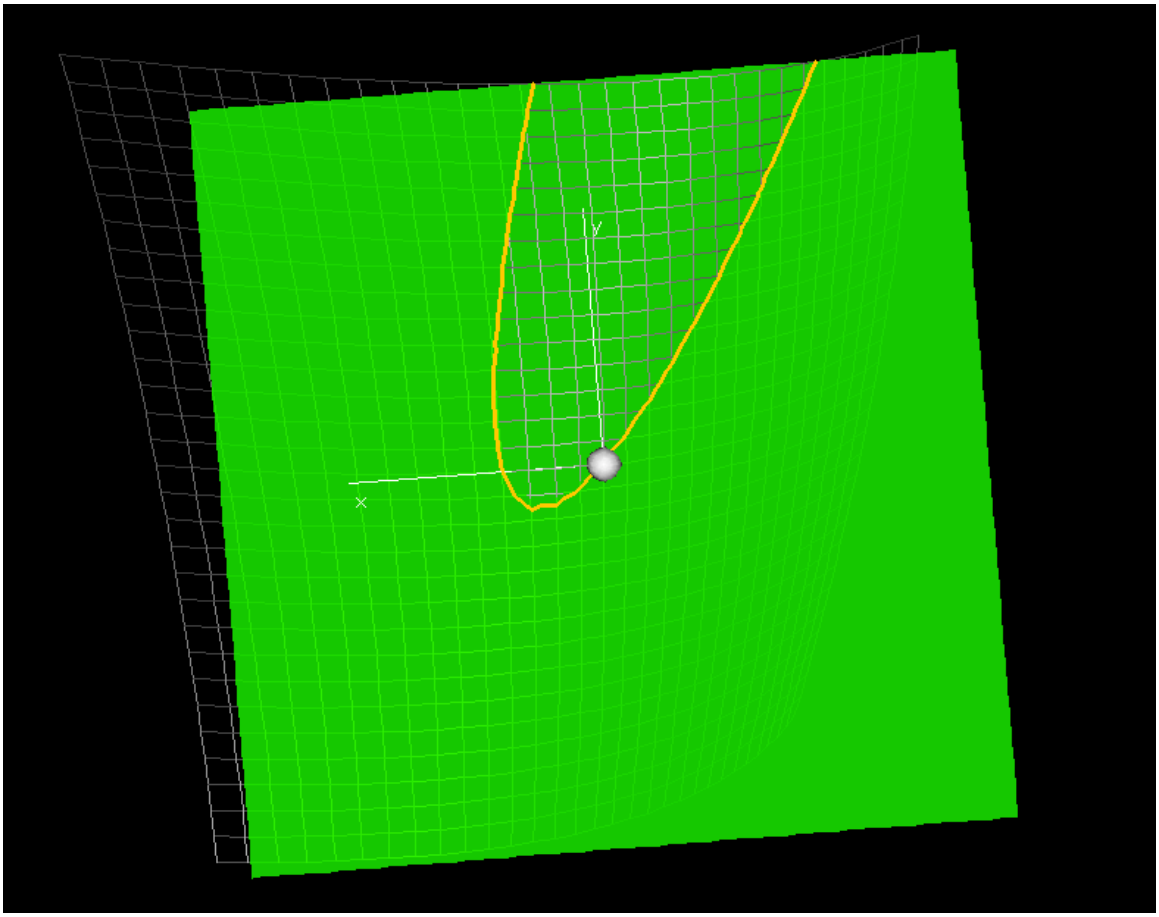
```
 5.000 2.000
 2.000 1.000

Vector x:[-2.0,4.0]
Vector b:[1.0,1.0]
Solution:
[-1.0000033243136934,3.0000080733332553]
------------------------------------------------------------
The calculations took 101 steps
Matrix:

 5.000 2.000
 2.000 1.000

Vector x:[-2.0,6.0]
Vector b:[1.0,1.0]
Solution:
[-1.0000171269723126,3.0000399629353964]
------------------------------------------------------------
The calculations took 101 steps
Matrix:

 5.000 2.000
 2.000 1.000

Vector x:[0.0,0.0]
Vector b:[1.0,1.0]
Solution:
[-0.9999828730276865,2.999960037064602]
------------------------------------------------------------
The calculations took 13 steps
Matrix:

 5.000 2.000
 2.000 1.000

Vector x:[0.0,2.0]
Vector b:[1.0,1.0]
Solution:
[-0.9999966756863069,2.999991926666745]
```

As shown the number of steps taken for each of the vectors are quite similar. The first and last vectors calculated both took 13 steps, whereas the middle two took 101 steps. I think this is the result of the level curve being an ellipsoid and that it reflects its symmetry. The values that took 101 steps reflect the x and y pairs that are farther off from the actual solution where as the values that took 13 steps are relatively closer.

Problem 2:
The randomly generated matrices are calculated and printed in the console after running the program.

Matrix [1.001 -0.999, 1.001 -0.999]





Matrix [5 2, 2 1]