Shannon Nguyen
MATH 2605
3/16/14

Steepest Descent

Implementation

In order to accomplish the Steepest Descent assignment, I created four different files: SDAlgorithm.java, ControlPanel.java, TextPanel.java, and SteepestDescent.java. Java was used because of the convenience of using the JAMA package, which is matrix package developed by The MathWorks and the National Institute of Standards and Technology. Further citation and a link to the source can be found in the Citations section.

The actual algorithm computation is in the SDAlgorithm.java file, which utilizes the Matrix class of the JAMA package. There, I have one method that performs the steepest descent calculation named calcSD(), a printMatrix() method that is used to help with matrix output, and 3 other methods that perform the scenarios proposed within the project rules named example1(), example2(), and example3().

calcSD() takes in 4 parameters: double[][] a, double[][] b, and double[][] x, and Matrix aMat. The matrix parameter is used for the case of the third example, where a random 10x10 matrix A is generated. Because we have to perform matrix operations in order to generate the random 10x10 matrix, I decide to pass in a matrix as opposed to a double array. Whenever I am either not using the 2D array or not using the matrix parameters, the respective parameter is set to null, and the method variables are set accordingly. The method returns a String in order to display the text on the applet.

There are numerous method variables.

- Matrix matA sets the matrix A
- Matrix matB sets the b matrix
- Matrix matX is sets the x matrix and is used as the holder for all subsequent x_k matrix calculations
- double magdk holds the value for the $|d_k|$ calculation
- int iterationCount keeps track of how many iterations of the process are performed. The variable is initialized to 1 in order to consider the first calculation of d_{k-1} part of the process.
- Matrix matD holds the matrix calculated for d_{k-1}
- Matrix matD2 holds the dot product of d_{k-1}, or the dot product of matD
- Matrix numer is essentially a temporary storage variable for matD2 that is used as the numerator in the calculation for x_k
- Matrix denom is the denominator matrix calculated for usage in the calculation of x_k
- Matrix fracToAdd is used to give more clarity to the calculation of x_k and contains the numerator times the inverse of the calculated denominator
- double accuracy stores the accuracy to be compared to $|d_k|$ and is always 10^{-5}
- String output is what is returned by the method and lists out each matrix X used and the number of iterations (iterationCount) required by the method to reach the local minimum.

The method begins with an initial calculation for d_{k-1} and $|d_k|$ so that if there is ever a best case Matrix x_o provided, the calculation of x_{k-1} is unnecessary. Otherwise, the program will loop and calculate and recalculate x_k, d_{k-1}, and $|d_k|$ until $|d_k|$ is less than the given accuracy or until the iterationCount cap of 1000000 is reached. The reason for the cap is that, in the situation of the random 10x10 matrix example, the numbers will occasionally be difficult to work and as a result take a long time to calculate and thus stall the entire program. Institution of a cap will at the least ensure that new matrices can be generated in decent time (usually no more than a minute and a little over a minute should the program ever need to iterate as many times as the count) and that resetting the program is unnecessary.

printMatrix() takes in a Matrix m parameter. It simply loops through the rows and columns of the matrix and appends the element in each matrix position to a single String variable named output and then returns output. I use this method as opposed to the printing method provided by JAMA because JAMA's print() method has no return statement and thus cannot be appended to a String.

example1() performs the method calculation for the first matrix example provided in Part 1 of the project guidelines. It returns String output that can display the matrix A, the matrix b, and 20 different randomly generated x_o examples (with values in accordance to the level curve graphs) and their iteration counts.

example2() performs the method calculation for the second matrix example provided in Part 1 of the project guidelines. It returns String output that can display the matrix A, the matrix b, and 20 different randomly generated x_o (with values in accordance to the level curve graphs) examples and their iteration counts.

example3() performs the method calculation for the Part 2 of the project guidelines. It returns String output that can display the 5 different matrices A, the 5 different matrices b, and 5 different x_o examples and their iteration counts for each different matrix A and its according b matrix. For the calculation of the random matrix A, it first generates and 10x10 2D matrix of randomly generated values into the variable double[][] temp. Then, the array is converted into a matrix in the variable Matrix matTemp. Using matTemp, the transpose of matTemp is multiplied by matTemp to get a new matrix stored in Matrix A that will be passed onto the calcSD() method. A 10x1 matrix b and 10x1 matrix x_0 is also randomly generated by looping through 2D arrays.

The other three java files are used to compile all the code in an applet for better presentation. The ControlPanel class provides a control panel that allows users to select which type of example provided in the project they want to run. The ControlPanel takes in a TextPanel and SDAlgorithm so that it can connect the two by access the appropriate information from SDAlgorithm and displaying it with TextPanel. The TextPanel class displays the iterations by using the String returns provided by the methods in the SDAlgorithm class. The class SteepestDescent is what is run by the user and brings both the ControlPanel and TextPanel together into a single window.

Project Discussion

Part 1, First Example:

$$F(x) = \tfrac{1}{2}(x \cdot Ax) - b \cdot x$$

$$A = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \qquad Ax = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5x+2y \\ 2x+y \end{bmatrix}$$

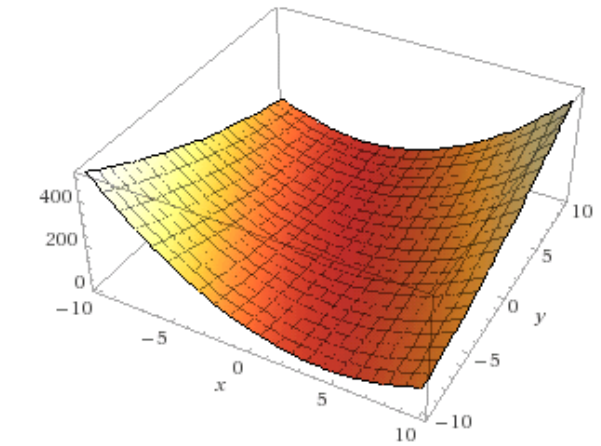
$$x \cdot Ax = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 5x+2y \\ 2x+y \end{bmatrix} = 5x^2+2xy+2xy+y^2 = 5x^2+4xy+y^2$$

$$b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

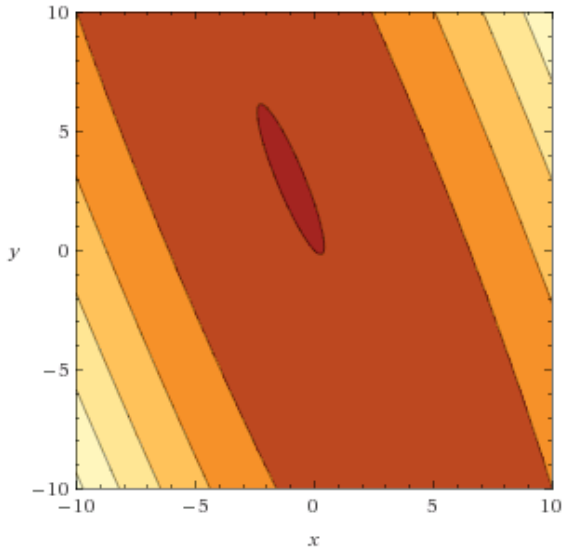
$$b \cdot x = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = x+y$$

$$F(x, y) = \tfrac{1}{2}(5x^2+4xy+y^2) - x - y$$

3D plot:



Contour plot:



Using the formula $F(x, y) = 1/2\ (x_o \cdot Ax_o) - b \cdot x_o$, I graphed the level curves (using the 2-variable graphing from Wolfram|Alpha) of the system's corresponding function. From this, I used randomly generated x_o coefficient ranging from -10 to 10 as demonstrated on the graph to test the various regions of the level curve. Below is a chart of various x_o points and their respective number of steps required to fulfill the requirements of the algorithm.

x: 0.0 y: 7.0	20 iterations	x: -2.0 y: 0.0	14 iterations	x: -7.0 y: 7.0	20 iterations	x: 4.0 y: 3.0	8 iterations	x: -2.0 y: 1.0	22 iterations
x: 8.0 y: -9.0	14 iterations	x: 7.0 y: 7.0	8 iterations	x: 2.0 y: -7.0	38 iterations	x: 8.0 y: -3.0	6 iterations	x: -9.0 y: 8.0	18 iterations
x: 5.0 y: -6.0	12 iterations	x: 3.0 y: 9.0	12 iterations	x: -4.0 y: 2.0	14 iterations	x: 7.0 y: 0.0	6 iterations	x: -5.0 y: -4.0	6 iterations
x: 4.0 y: 10.0	12 iterations	x: -7.0 y: -6.0	4 iterations	x: -4.0 y: -6.0	6 iterations	x: 8.0 y: -4.0	8 iterations	x: -2.0 y: 3.0	108 iterations

From observing the number of iterations of those values listed on the table and looking at the 3D plo, those values that are in the lower, un-elevated regions of the graph require more iterations than those that lie in the side regions of the graph. For instance, looking at (-2.0, 3.0) which took 108 iterations, it is located in the lower region of the graph whereas (-7.0, -6.0) rests on the elevated left side of the graph. From these observations, I assumed the same pattern would be seen for the next matrix example in Part 1 of the project guidelines.

Part 1, Second Example:

$$F(x) = \tfrac{1}{2}(x \cdot Ax) - b \cdot x$$

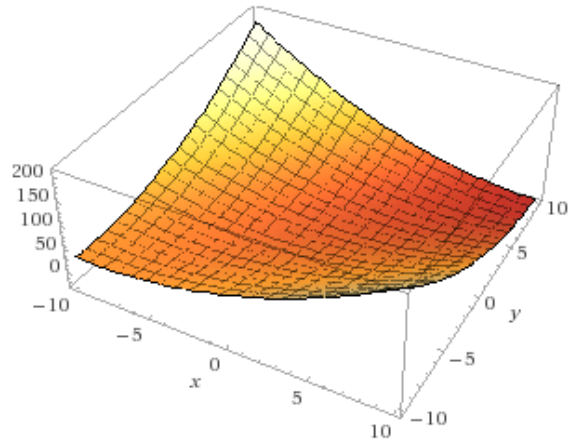
$$A = \begin{bmatrix} 1.001 & -0.999 \\ -0.999 & 1.001 \end{bmatrix} \quad Ax = \begin{bmatrix} 1.001 & -0.999 \\ -0.999 & 1.001 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.001x - 0.999y \\ -0.999x + 1.001y \end{bmatrix}$$

$$x \cdot Ax = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1.001x - 0.999y \\ -0.999x + 1.001y \end{bmatrix} = 1.001x^2 - 0.999xy - 0.999xy + 1.001y^2 = 1.001x^2 - 1.998xy + 1.001y^2$$

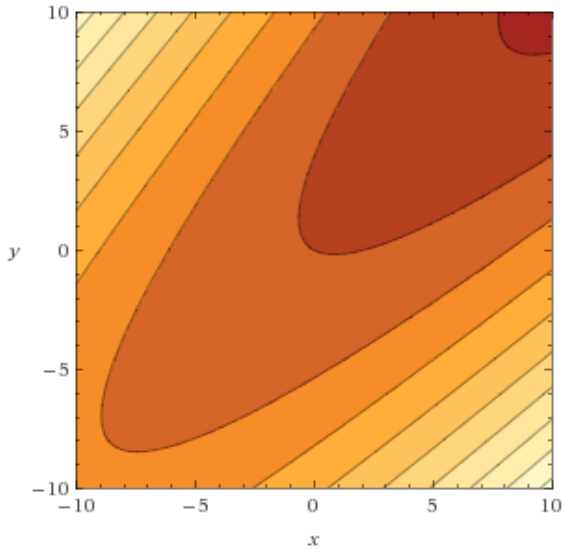
$$b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad b \cdot x = \begin{bmatrix} 1, 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x + 2y$$

$$F(x, y) = \tfrac{1}{2}(1.001x^2 - 1.998xy + 1.001y^2) - x - 2y$$

3D plot:



Contour plot:



The same process was repeated for the process noted above, with the program generating 20 x_o matrices to test the algorithm on.

x: -9.0 y: -7.0	4786 iterations	x: -1.0 y: -1.0	2231 iterations	x: 5.0 y: 5.0	2205 iterations	x: -5.0 y: -6.0	2251 iterations
x: 10.0 y: 7.0	4896 iterations	x: 6.0 y: 3.0	4870 iterations	x: -10.0 y: 10.0	144 iterations	x: -7.0 y: 0.0	914 iterations
x: -5.0 y: 1.0	1186 iterations	x: 9.0 y: 6.0	4890 iterations	x: 2.0 y: 8.0	1208 iterations	x: 0.0 y: 10.0	502 iterations
x: -1.0 y: -3.0	6306 iterations	x: 10.0 y: -7.0	214 iterations	x: -4.0 y: 10.0	274 iterations	x: 8.0 y: 0.0	932 iterations
x: -6.0 y: 0.0	1184 iterations	x: -2.0 y: -4.0	6304 iterations	x: -8.0 y: 8.0	214 iterations	x: 1.0 y: -4.0	2220 iterations

As I predicted, the same pattern continues with this function. Looking at (9.0, 6.0) which has 4890 iterations, it rests in the trough region of the graph, as opposed to (-8.0, 8.0) with 214 iterations, which can be pinpointed near the top, elevated corner on the 3D plot.

Part 2:

With part two, I had the program randomly generate a 10x10 matrix and a 10x1 b matrix to go with it, on top of 5 different 10x1 x_o matrices to test with each system. Because of the large amount of iterations going with each, I will post screenshots of the particular matrices generated for me.

Matrix A:

380.0	146.0	19.0	104.0	-37.0	-136.0	39.0	81.0	31.0
146.0	390.0	-121.0	205.0	-287.0	-183.0	61.0	-53.0	-123.0
19.0	-121.0	398.0	-3.0	64.0	118.0	-44.0	-117.0	139.0
104.0	205.0	-3.0	407.0	-176.0	-33.0	-125.0	-48.0	33.0
-37.0	-287.0	64.0	-176.0	339.0	98.0	33.0	130.0	137.0
-136.0	-183.0	118.0	-33.0	98.0	304.0	42.0	4.0	-12.0
39.0	61.0	-44.0	-125.0	33.0	42.0	283.0	140.0	-206.0
81.0	-53.0	-117.0	-48.0	130.0	4.0	140.0	268.0	-144.0
31.0	-123.0	139.0	33.0	137.0	-12.0	-206.0	-144.0	362.0
-51.0	206.0	-102.0	110.0	-177.0	-75.0	11.0	-129.0	-13.0

Matrix b:

-6.0
-9.0
1.0
3.0
-10.0
-5.0
-9.0
0.0
1.0
3.0

Matrix x:

2.0
2.0
9.0
3.0
2.0
8.0
-3.0
-1.0
-6.0
-5.0

Number of iterations: 22135

Matrix x:

-5.0
-7.0
-5.0
-6.0
6.0
4.0
-5.0
-5.0
-7.0
-7.0

Number of iterations: 21584

Matrix x:

-3.0
-9.0
-7.0
3.0
-9.0
2.0
1.0
5.0
-8.0
6.0

Number of iterations: 23769

Matrix x:

5.0
-4.0
7.0
-8.0
6.0
-10.0
-10.0
0.0
7.0
-10.0

Number of iterations: 22464

Matrix x:

-5.0
-4.0
4.0
-7.0
9.0
9.0
-4.0
-5.0
-4.0
0.0

Number of iterations: 18750

Matrix A:

363.0	-22.0	-74.0	13.0	81.0	-147.0	-34.0	82.0	-88.0
-22.0	336.0	40.0	81.0	-79.0	-42.0	12.0	-55.0	-187.0
-74.0	40.0	455.0	32.0	185.0	142.0	200.0	-327.0	119.0
13.0	81.0	32.0	231.0	40.0	-104.0	-84.0	134.0	-106.0
81.0	-79.0	185.0	40.0	410.0	87.0	90.0	99.0	72.0
-147.0	-42.0	142.0	-104.0	87.0	225.0	49.0	-163.0	165.0
-34.0	12.0	200.0	-84.0	90.0	49.0	386.0	-260.0	-29.0
82.0	-55.0	-327.0	134.0	99.0	-163.0	-260.0	640.0	-141.0
-88.0	-187.0	119.0	-106.0	72.0	165.0	-29.0	-141.0	276.0
58.0	53.0	-124.0	-76.0	-165.0	-10.0	-110.0	-73.0	38.0

Matrix b:

-6.0
10.0
8.0
4.0
5.0
-3.0
-8.0
8.0
7.0
8.0

Matrix x:	Matrix x:	Matrix x:
-9.0	7.0	-5.0
4.0	-3.0	-3.0
-2.0	-1.0	5.0
8.0	-6.0	-6.0
-3.0	9.0	-2.0
-7.0	-1.0	-3.0
-9.0	3.0	-8.0
4.0	-2.0	3.0
10.0	-8.0	5.0
6.0	1.0	-4.0
Number of iterations: 6080	Number of iterations: 6690	Number of iterations: 6049

Matrix x:	Matrix x:
-6.0	9.0
-9.0	-7.0
-7.0	-7.0
10.0	5.0
-9.0	4.0
-5.0	7.0
-10.0	5.0
0.0	1.0
-6.0	0.0
-4.0	10.0
Number of iterations: 5794	Number of iterations: 5569

Matrix A:

257.0	-78.0	-128.0	-184.0	133.0	57.0	40.0	-172.0	-56.0
-78.0	402.0	145.0	-47.0	50.0	-125.0	21.0	114.0	55.0
-128.0	145.0	283.0	-23.0	9.0	-63.0	-44.0	119.0	32.0
-184.0	-47.0	-23.0	578.0	-213.0	-103.0	-133.0	189.0	72.0
133.0	50.0	9.0	-213.0	266.0	-72.0	36.0	-38.0	-63.0
57.0	-125.0	-63.0	-103.0	-72.0	231.0	158.0	59.0	17.0
40.0	21.0	-44.0	-133.0	36.0	158.0	407.0	233.0	96.0
-172.0	114.0	119.0	189.0	-38.0	59.0	233.0	487.0	83.0
-56.0	55.0	32.0	72.0	-63.0	17.0	96.0	83.0	181.0
91.0	-85.0	-221.0	104.0	-41.0	87.0	19.0	12.0	19.0

Matrix b:

7.0
-6.0
-3.0
-9.0
1.0
1.0
2.0
-10.0
0.0
-3.0

Matrix x:	Matrix x:	Matrix x:
-3.0	-2.0	6.0
-9.0	-4.0	3.0
-10.0	10.0	-10.0
-7.0	-3.0	-8.0
7.0	0.0	-7.0
7.0	-9.0	0.0
-4.0	3.0	-3.0
-6.0	1.0	-1.0
-3.0	6.0	-4.0
-4.0	-10.0	4.0

Number of iterations: 2596 Number of iterations: 2950 Number of iterations: 3060

Matrix x:	Matrix x:
-6.0	-7.0
-7.0	-6.0
-10.0	-1.0
-6.0	0.0
-6.0	6.0
2.0	-8.0
3.0	9.0
-10.0	0.0
-3.0	-6.0
-1.0	3.0

Number of iterations: 2096 Number of iterations: 3335

Matrix A:

337.0	-4.0	33.0	-15.0	10.0	-127.0	287.0	-27.0	115.0
-4.0	522.0	49.0	-132.0	259.0	-195.0	66.0	-53.0	-60.0
33.0	49.0	226.0	-119.0	24.0	49.0	57.0	-91.0	11.0
-15.0	-132.0	-119.0	309.0	-135.0	-28.0	-113.0	-115.0	-32.0
10.0	259.0	24.0	-135.0	211.0	-131.0	68.0	95.0	53.0
-127.0	-195.0	49.0	-28.0	-131.0	239.0	-119.0	20.0	17.0
287.0	66.0	57.0	-113.0	68.0	-119.0	376.0	161.0	226.0
-27.0	-53.0	-91.0	-115.0	95.0	20.0	161.0	499.0	355.0
115.0	-60.0	11.0	-32.0	53.0	17.0	226.0	355.0	424.0
124.0	-51.0	42.0	82.0	3.0	33.0	109.0	111.0	314.0

Matrix b:

-6.0
-6.0
2.0
-7.0
-9.0
7.0
4.0
-3.0
3.0
1.0

Matrix x:	Matrix x:	Matrix x:
0.0	-4.0	0.0
7.0	2.0	-8.0
5.0	-10.0	-7.0
-9.0	5.0	-3.0
-10.0	-6.0	-1.0
-3.0	0.0	1.0
-9.0	5.0	-4.0
-5.0	-7.0	-8.0
-3.0	8.0	3.0
7.0	10.0	10.0
Number of iterations: 262996	Number of iterations: 285710	Number of iterations: 283778

Matrix x:	Matrix x:
-10.0	-2.0
0.0	-6.0
3.0	3.0
4.0	-4.0
8.0	0.0
0.0	0.0
-2.0	-2.0
-10.0	0.0
0.0	7.0
6.0	-7.0
Number of iterations: 269488	Number of iterations: 285187

Matrix A:

410.0	-164.0	-8.0	71.0	266.0	65.0	-101.0	-37.0	105.0
-164.0	418.0	-70.0	-136.0	-187.0	47.0	32.0	63.0	-26.0
-8.0	-70.0	409.0	92.0	57.0	-50.0	37.0	37.0	-131.0
71.0	-136.0	92.0	288.0	83.0	-106.0	-6.0	28.0	-237.0
266.0	-187.0	57.0	83.0	344.0	154.0	-32.0	-71.0	98.0
65.0	47.0	-50.0	-106.0	154.0	555.0	158.0	-136.0	99.0
-101.0	32.0	37.0	-6.0	-32.0	158.0	394.0	-152.0	-85.0
-37.0	63.0	37.0	28.0	-71.0	-136.0	-152.0	211.0	-136.0
105.0	-26.0	-131.0	-237.0	98.0	99.0	-85.0	-136.0	479.0
4.0	-159.0	137.0	21.0	75.0	17.0	-108.0	-3.0	112.0

Matrix b:

-2.0
-1.0
2.0
7.0
-5.0
6.0
-4.0
0.0
-6.0
0.0

Matrix x:

-6.0
0.0
10.0
-2.0
-3.0
-4.0
0.0
3.0
-4.0
-6.0

Number of iterations: 380

Matrix x:

-8.0
-9.0
-2.0
8.0
0.0
8.0
-10.0
4.0
7.0
8.0

Number of iterations: 430

Matrix x:

6.0
7.0
-9.0
3.0
-4.0
-6.0
10.0
-1.0
3.0
-2.0

Number of iterations: 394

Matrix x:

4.0
-1.0
9.0
-3.0
0.0
2.0
2.0
-6.0
10.0
3.0

Number of iterations: 374

Matrix x:

-5.0
-1.0
-8.0
9.0
2.0
0.0
-7.0
3.0
0.0
5.0

Number of iterations: 400

Citations

- JAMA: http://math.nist.gov/javanumerics/jama/
    - Specifically for this project, the following files were used:
        - Matrix.java
- Wolfram|Alpha: http://www.wolframalpha.com/examples/PlottingAndGraphics.html