AMS 326 | FALL 2012 | STONY BROOK UNIVERSITY

Homework 3

Dinia Gepte | 107092681

10/16/2012

Problem 1

Generate a 8 x 8 matrix *A* with random numbers $a_{ij} \in [-1,1]$ as its elements. These random numbers are uniformly distributed in the given interval. Please do the following:

1) Compute the matrix's determinant $\det(A)$.
2) Compute the matrix's inverse A^{-1}.
3) Compute the product of your matrix A and the A^{-1} you generated above.

- The source code of the program is in a file called `Matrix.java` included in the zip file. A copy of the code is provided below:

```
public class Matrix
{
	public static void main(String[] args)
	{
		// CLASS USED TO TAKE USER INPUT IN THE CONSOLE
		Scanner input = new Scanner(System.in);

		// ASK THE USER FOR THE SIZE OF THE N-BY-N MATRIX
		System.out.print("Enter n for an n-by-n matrix A: ");
		int n = input.nextInt();
		double[][] matrixA = generateSquareMatrix(n, -1, 1);

		// THIS IS A LOOP THAT WILL PROVIDE OPTIONS TO THE USER TO GET:
		// (M) PRINT THE MATRIX
		// (D) DETERMINANT OF MATRIX A
		// (I) INVERSE OF MATRIX A
		// (P) PRODUCT OF MATRIX A AND ITS INVERSE
		// (Q) EXIT THE PROGRAM
		String choice = "?";
		while (!choice.toUpperCase().equals("Q"))
		{
			System.out.println("\nEnter an option below:");
			System.out.println("(M) matrix A");
			System.out.println("(D) det(A)");
			System.out.println("(I) inverse(A)");
			System.out.println("(P) product of A and inverse(A)");
			System.out.println("(Q) quit");
			System.out.print("Choice: ");
			choice = input.next();

			// PRINT MATRIX A
			if (choice.toUpperCase().equals("M"))
				show(matrixA);
			// CALCULATE det(A)
			else if (choice.toUpperCase().equals("D"))
				det(matrixA);
			// PRINT THE INVERSE OF A
			else if (choice.toUpperCase().equals("I"))
				show(inv(matrixA));
			// CALCULATE THE PRODUCT OF A AND ITS INVERSE
			else if (choice.toUpperCase().equals("P"))
				show(prod(matrixA, inv(matrixA)));
		}
	}

	private static double[][] prod(double[][] a, double[][] b)
	{
		double[][] c = new double[a.length][a[0].length];
```

```
                for (int i = 0; i < a.length; i++)
                        for (int j = 0; j < b.length; j++)
                                for(int k = 0; k < c.length; k++)
                                        c[i][j] += a[i][k] * b[k][j];
                return c;
        }

        private static double[][] inv(double[][] a)
        {
                // TRANSFORM THE SQUARE MATRIX INTO A RealMatrix OBJECT
                RealMatrix m = new Array2DRowRealMatrix(a);
                // INVERT m USING LU DECOMPOSITION
                RealMatrix mInverse = new LUDecomposition(m).getSolver().getInverse();
                // RETURN THE INVERSE DATA AS A DOUBLE[][]
                return mInverse.getData();
        }

        private static void det(double[][] a)
        {
                System.out.printf("det(a) = %.6f\n", laplaceDet(a));
        }

        private static double laplaceDet(double[][] m)
        {
                double det = 0;
                if (m.length == 2)     // A 2-by-2 MATRIX
                        det = m[0][0] * m[1][1] - m[0][1] * m[1][0];
                else    // AN N-BY-N FOR N > 2
                        for (int j = 1; j <= m.length; j++)
                                det += m[0][j-1] * cofactor(m,1,j);
                return det;
        }

        private static double cofactor(double[][] m, int r, int c)
        {
                return Math.pow(-1, r+c) * laplaceDet(delete(m,r,c));
        }

        private static double[][] delete(double[][] m, int r, int c)
        {
                // MAKE AN n-1-by-n-1 MATRIX
                double[][] n = new double[m.length-1][m[0].length-1];

                int x = 0;      // ROW INDEX OF MATRIX n
                for (int i = 1; i <= m.length; i++)
                {
                        if (i == r)     continue;       // r-TH ROW IS TO BE DELETED SO SKIP IT
                        int y = 0;      // COLUMN INDEX OF MATRIX n
                        for (int j = 1; j <= m[i-1].length; j++)
                        {
                                if (j == c)     continue;       // c-TH COLUMN IS TO BE DELETED SO SKIP IT
```

```
                                n[x][y] = m[i-1][j-1]; // ASSIGN THE VALUE TO THE NEW MATRIX
                                y++;
                        }
                        x++;
                }

                // RETURN THE NEW MATRIX
                return n;
        }

        private static void show(double[][] a)
        {
                for (int i = 0; i < a.length; i++)
                {
                        for (int j = 0; j < a[i].length; j++)
                                System.out.printf("%.6f \t", a[i][j]);
                        System.out.println();
                }
        }

        private static double[][] generateSquareMatrix(int n, double a, double b)
        {
                // DECLARE AN n-by-n MATRIX
                double[][] matrix = new double[n][n];

                // INITIALIZE THE MATRIX WITH RANDOM NUMBERS [a,b]
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                matrix[i][j] = random(a, b);

                // RETURN THE MATRIX
                return matrix;
        }

        private static double random(double x, double y)
        {
                // INITITAL SEEDS
                double x0 = Math.random();
                double x1 = Math.random();

                // GENERATE A RANDOM NUMBER x2 = [0, 1]
                double x2 = (x0 + x1 < 1) ? (x0 + x1) : (x0 + x1 - 1.0);

                // GENERATE A RANDOM NUMBER [x, y]
                return (y - x) * x2 + (x);
        }
}
```

- A sample run of the program is below:

```
Enter n for an n-by-n matrix A: 8

Enter an option below:
(M) matrix A
(D) det(A)
(I) inverse(A)
(P) product of A and inverse(A)
(Q) quit
Choice: m
-0.065778     -0.688974     -0.493951     0.429385      0.335880      -0.326377     -0.493153     0.703773
-0.786402     -0.303677     -0.903412     0.362649      0.611045      0.145978      0.691994      -0.866345
0.433686      -0.009034     0.583879      -0.382663     -0.639604     -0.754766     -0.570553     -0.653356
-0.157775     0.981529      -0.691691     0.117181      -0.608549     -0.033050     0.045350      0.541331
-0.635961     0.373932      0.503562      0.779388      -0.561370     0.019798      -0.450021     -0.599191
0.995199      -0.412001     0.397822      -0.671275     -0.403520     0.284172      -0.971998     0.820823
-0.224685     0.380135      0.393086      0.439732      0.331026      0.098941      -0.893139     -0.033959
-0.662050     -0.705865     0.322618      0.207059      -0.930747     -0.623691     -0.496718     -0.345540

Enter an option below:
(M) matrix A
(D) det(A)
(I) inverse(A)
(P) product of A and inverse(A)
(Q) quit
Choice: d
det(a) = 2.060652

Enter an option below:
(M) matrix A
(D) det(A)
(I) inverse(A)
(P) product of A and inverse(A)
(Q) quit
Choice: i
0.866178      0.031479      0.768993      -0.226228     1.238732      0.333920      -1.187046     -1.361365
-0.297399     -0.168261     0.153098      0.423066      -0.234160     -0.296686     0.485709      -0.157006
-0.327064     -0.837250     -0.353508     -0.548074     -0.076547     -0.472604     0.150366      0.238125
1.158260      -0.227024     0.213042      -0.295983     1.722635      -0.108743     -1.119016     -1.073767
-0.001632     -0.055121     -0.010861     -0.263985     -0.697088     -0.395797     0.769987      -0.065230
-0.439198     0.511994      -0.584274     -0.074169     0.591717      0.862176      -0.142319     -0.153657
0.029081      -0.497064     -0.225676     -0.259770     0.227513      -0.511805     -0.834560     -0.203054
0.091971      -0.695464     -0.580255     0.098503      -0.451672     -0.294496     0.134536      0.358901
```

```
Enter an option below:
(M) matrix A
(D) det(A)
(I) inverse(A)
(P) product of A and inverse(A)
(Q) quit
Choice: p
1.000000      -0.000000     0.000000      -0.000000     0.000000      0.000000      0.000000      0.000000
-0.000000     1.000000      -0.000000     -0.000000     0.000000      0.000000      -0.000000     -0.000000
-0.000000     0.000000      1.000000      0.000000      -0.000000     0.000000      -0.000000     0.000000
-0.000000     -0.000000     -0.000000     1.000000      -0.000000     -0.000000     0.000000      -0.000000
0.000000      0.000000      -0.000000     0.000000      1.000000      -0.000000     0.000000      -0.000000
-0.000000     0.000000      -0.000000     0.000000      0.000000      1.000000      -0.000000     0.000000
0.000000      -0.000000     -0.000000     0.000000      0.000000      -0.000000     1.000000      -0.000000
0.000000      -0.000000     -0.000000     -0.000000     0.000000      0.000000      0.000000      1.000000

Enter an option below:
(M) matrix A
(D) det(A)
(I) inverse(A)
(P) product of A and inverse(A)
(Q) quit
Choice: q
```

- The program asks the user for an integer size of an n x n array. The program then creates a random matrix of this size with random values from [-1, 1] (this range can be changed in the main method). Generating a random number uses the poor man's algorithm that's been discussed in class. I decided upon this because using the random number generator in the Java library only generates values [x, y) (exclusive). After the matrix has been generated, this matrix will be retained throughout runtime. It is used to calculate the various properties above. I have implemented the determinant and taking the product of 2 matrices, while using Apache's Math library (http://commons.apache.org/math/download_math.cgi) that provides linear algebra operations. The .jar file is included in the zip file.
- To find the determinant, I used the Laplace expansion (http://en.wikipedia.org/wiki/Laplace_expansion). I basically implemented the formula provided in the link. I've done some research and, apparently, doing it by this method where recursion is used is slow. It will be faster to find the determinant by using the L-U decomposition; the Apache package uses the L-U to determine the determinant.
- Due to the length of the method descriptions, I excluded them in the code snippet above but in the zip file.

Problem 2

Consider shooting a bullet of mass $\mathrm{m} = 0.1$ kilo-grams to a medium whose resistance is $-\alpha(4 + \mathrm{v}^2)$ where the resistance constant $\alpha = 0.005\ \text{Newton} \times \mathrm{s}^2/\mathrm{m}^2$. Such a unit for α is chosen to eliminate your worry of the unit consistency if Meter-Kilogram-Second (MKS) unit system is adopted. If the initial speed of the bullet is $\mathrm{v}_0 = 100\frac{\mathrm{m}}{\mathrm{s}}, 200\frac{\mathrm{m}}{\mathrm{s}}, 300\frac{\mathrm{m}}{\mathrm{s}}, 400\frac{\mathrm{m}}{\mathrm{s}}$, compute the farthest distances the bullet can travel in the medium. You may assume the medium infinite (for example, the ocean) and you may also neglect the gravity for this problem. Plot a graph to show the relationship of your calculated distances vs. the initial speeds. Fit the four data points to a curve of your choice.

- The source code of the program is in a file called `Bullet.java` included in the zip file. A copy of the code is provided below:

```
public class Bullet
{
        private static final double DELTA = 0.001;   // ARBITRARILY CHOSEN
        private static final double MASS = 0.1; // KG
        private static final double RESISTANCE = 0.005; // N * (s^2 / m^2)

        public static void main(String[] args)
        {
                // GET THE INITIAL VELOCITY FROM THE USER
                Scanner input = new Scanner(System.in);
                System.out.print("Enter an initial velocity (m/s): ");
                double initVelocity = input.nextDouble();

                // THESE ARRAYS WILL HOLD THE VALUES OF THE VELOCITY OF THE
                // BULLET AT ANY GIVEN TIME
                ArrayList<Double> velocity = new ArrayList<Double>();
                ArrayList<Double> time = new ArrayList<Double>();

                // ADD THE INITIAL VALUES TO THE ARRAY
                velocity.add(initVelocity);   // v0 = USER-ENTERED VALUE
```

```
                time.add(0.0); // t0 = 0

                do
                {
                        double lastV = velocity.get(velocity.size()-1);
                        double lastT = time.get(time.size()-1);
                        double nextV = lastV + DELTA / 6.0 * (k1(lastT, lastV) +
                                        2*k2(lastT, lastV) + 2*k3(lastT, lastV) + k4(lastT,
lastV));
                        double nextT = lastT + DELTA;
                        velocity.add(nextV);
                        time.add(nextT);

                } while (velocity.get(velocity.size()-1) > 0);
                // LOOP WILL END WHEN VELOCITY REACHES 0 OR NEGATIVE,
                // THAT IS WHEN THE BULLET STOPS.

                // REMOVE THE LAST ELEMENTS SINCE VELOCITY IS NEGATIVE
                velocity.remove(velocity.size()-1);
                time.remove(time.size()-1);

                // OUTPUT THE DISTANCE TO THE USER
                System.out.printf("The maximum distance the bullet travels is %.4f m.",
                                distance(time, velocity));
        }

        public static double distance(ArrayList<Double> time, ArrayList<Double> velocity)
        {
                double distance = 0;
                for (int i = 1; i < velocity.size(); i++)
                        distance += velocity.get(i) * (time.get(i) - time.get(i-1));
                return distance;
        }

        public static double acceleration(double t, double v)
        {
                return -RESISTANCE * (4 + Math.pow(v, 2)) / MASS;
        }

        public static double k1(double t, double v)
        {
                return acceleration(t, v);
        }

        public static double k2(double t, double v)
        {
                return acceleration(t + 0.5*DELTA, v + 0.5*DELTA*k1(t,v));
        }

        public static double k3(double t, double v)
        {
                return acceleration(t + 0.5*DELTA, v + 0.5*DELTA*k2(t,v));
        }

        public static double k4(double t, double v)
        {
                return acceleration(t + DELTA, v + DELTA*k3(t,v));
        }
}
```

- 4 sample runs of the program is below:

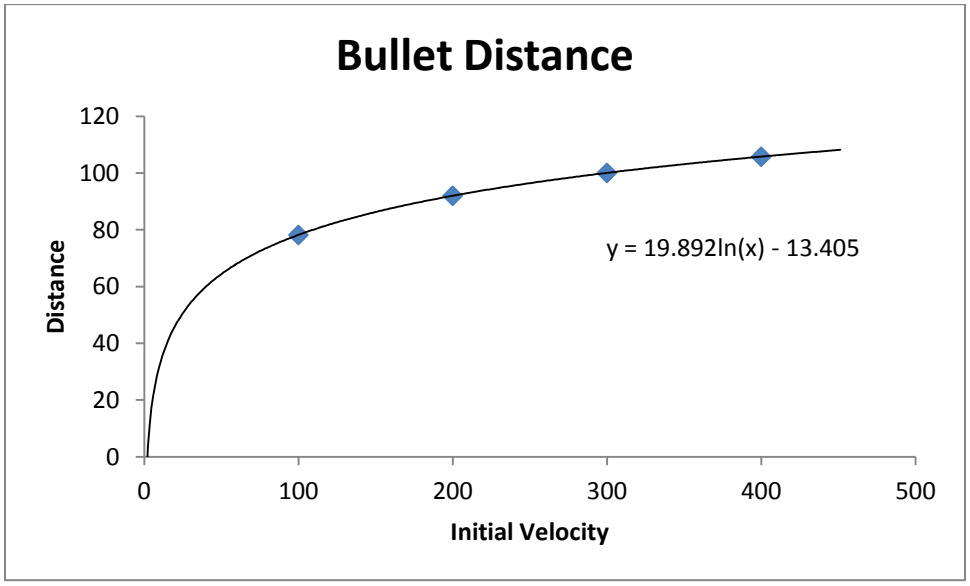
```
Enter an initial velocity (m/s): 100
The maximum distance the bullet travels is 78.1945 m.

Enter an initial velocity (m/s): 200
The maximum distance the bullet travels is 92.0046 m.
```

```
Enter an initial velocity (m/s): 300
The maximum distance the bullet travels is 100.0635 m.

Enter an initial velocity (m/s): 400
The maximum distance the bullet travels is 105.7673 m.
```

- The graph and curve of the relationship between distance traveled and initial velocity of the bullet is below, using MS Excel:



- Using Runge-Kutta's method to calculate IVP, I continuously calculated the velocity until it became zero or negative (a non-positive velocity means the bullet has stopped). After calculating the velocity and time, I used Reimann's sum to calculate the distance traveled by the bullet when it stopped. Another implementation to calculate the velocity would be to use Euler's method but Runge-Kutta provides a more accurate result.
- I used MS Excel to plot the data points of distance vs. initial velocity. I then decided on the curve to be logarithmic because it makes sense that if the initial velocity is 0, the distance traveled by the bullet is 0 as well.
- The method descriptions can be found in the java code in the zip file.