ECSE 543: Assignment 1

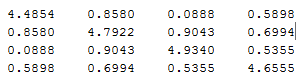
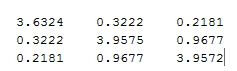
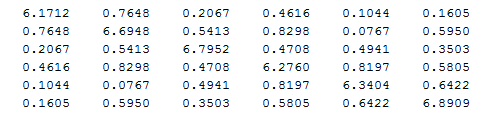
Razi Murshed

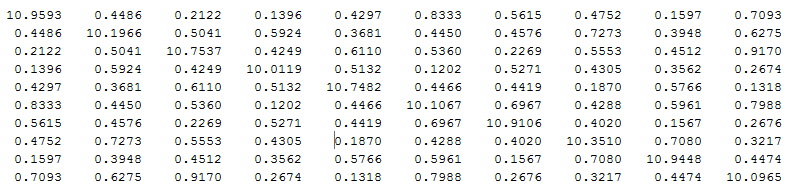
260516333

# Question 1

1. Code provided in Appendix as “CholeskyDecomposition.java”.
2. The following random SPD Matrices were generated using the Matlab script “generateSPDmatrix.m”. The following steps were followed to construct these matrices –

* Randomly generate an n by n matrix.
* Construct a symmetric matrix A AT, using the formula –
* Since by construction and a symmetric diagonally dominant matrix is symmetric positive definite, which can be ensured by adding .

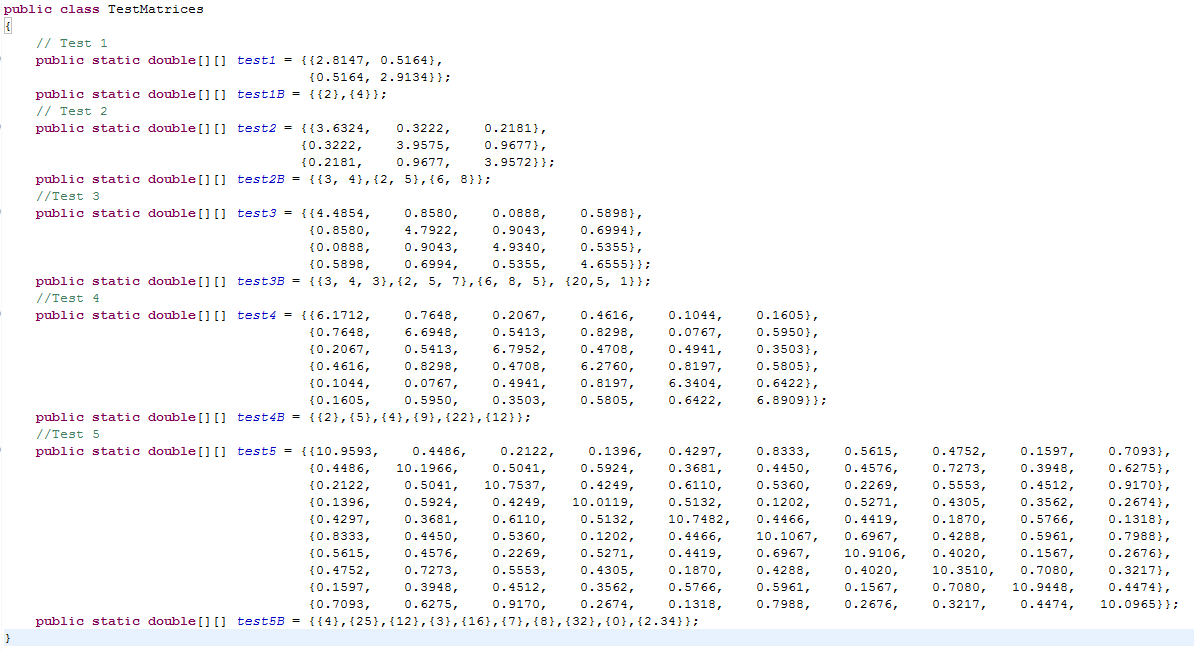
  



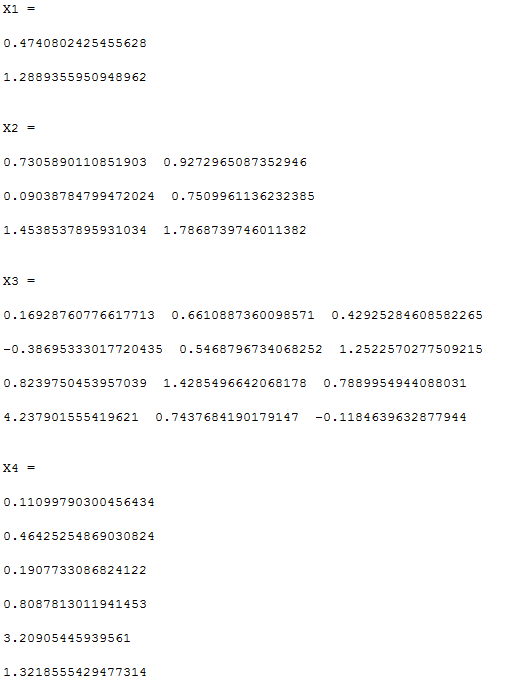
1. The program was tested with the matrices given by “TestMatrices.java” in the Appendix

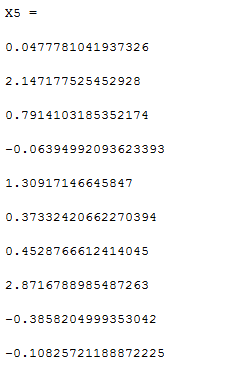
## 1c) “TestMatrices.java”

1. The following test matrices for A and B were used to test if the program worked correctly.



The following are the results of these tests proving that the program works correctly.–

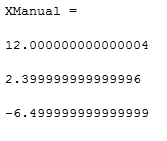




1. Let us assume a SPD matrix A and a vector X defined as below –

If we multiply A and X we get a new matrix B which is –

Now giving A and B to our program we should get X back if it is working correctly. In this care this the output matches –



# Appendix

## 1a) “CholeskyDecomposition.java”

**import** java.util.\*;

/\*

\* Class: CholeskyDecomposition

\* Description: Carries out and tests Cholesky Decomposition on SPD Matrices

\*/

**public** **class** CholeskyDecomposition

{

/\*

\* Name: main()

\* Parameters: String[]

\* Output : void

\* Description: Main function.

\*/

**public** **static** **void** main(String[] args)

{

System.*out*.println("X1 = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*test1*, TestMatrices.*test1B*));

System.*out*.println("X2 = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*test2*, TestMatrices.*test2B*));

System.*out*.println("X3 = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*test3*, TestMatrices.*test3B*));

System.*out*.println("X4 = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*test4*, TestMatrices.*test4B*));

System.*out*.println("X5 = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*test5*, TestMatrices.*test5B*));

System.*out*.println("XManual = \n");

*matrixPrint*(*matrixSolver*(TestMatrices.*testManualMatA*, TestMatrices.*testManualMatB*));

}

/\*

\* Name: choleskyDecompose()

\* Parameters: double[][]

\* Output : double[][]

\* Description: Takes a 2D Matrix inputMatrix and does Cholesky Decomposition on it to

\* produce a lower triangular matrix output.

\*/

**public** **static** **double**[][] choleskyDecompose(**double**[][] inputMatrix)

{

**int** lengthOfMatrix = inputMatrix.length;

**double**[][] outputMatrix = **new** **double**[lengthOfMatrix][lengthOfMatrix];

**for**(**int** i = 0; i < lengthOfMatrix;i++)

{

**for**(**int** k = 0; k < (i+1); k++)

{

**double** sum = 0;

**for**(**int** j = 0; j < k; j++)

{

sum += outputMatrix[i][j]\*outputMatrix[k][j];

}

**if** (i==k)

{

outputMatrix[i][k] = Math.*sqrt*(inputMatrix[i][i] - sum);

}

**else**

{

outputMatrix[i][k]= (1.0 / outputMatrix[k][k] \* (inputMatrix[i][k] - sum));

}

}

}

**return** outputMatrix;

}

/\*

\* Name: matrixSolver()

\* Parameters: double[][], double[][]

\* Output : double[][]

\* Description: Solves the matrix equation AX=B using cholesky decomposition

\*/

**public** **static** **double**[][] matrixSolver(**double**[][] A, **double**[][] B)

{

**double** sum;

**if**(B.length != A.length)

{

System.*out*.println("Matrix row dimensions must agree.");

}

**int** ARowDim = A.length;

**int** BColDim = B[0].length;

**double**[][] X = **new** **double**[ARowDim][BColDim];

**double**[][] U = *choleskyDecompose*(A);

**double**[][] Ut = *matrixTranspose*(U);

**double**[][] Y = **new** **double**[ARowDim][BColDim];

//Solving for U\*y= B

**for**(**int** k =0; k < BColDim; k++)

{

**for**(**int** i = 0; i < ARowDim; i++)

{

sum = 0.0;

**for**(**int** j =0; j < i ; j++)

{

sum += U[i][j]\*Y[j][k];

}

Y[i][k] = (B[i][k]- sum)/U[i][i];

}

//Solving for Ut\*x = Y

**for**(**int** i = ARowDim - 1; i > -1 ; i--)

{

sum = 0.0;

**for** (**int** j = i + 1; j < ARowDim; j++)

{

sum += Ut[i][j]\*X[j][k];

}

X[i][k] = (Y[i][k]- sum)/(Ut[i][i]);

}

}

**return** X;

}

/\*

\* Name: matrixTranspose()

\* Parameters: double[][]

\* Output : double[][]

\* Description: Transposes a matrix

\*/

**public** **static** **double**[][] matrixTranspose(**double**[][] inputMatrix)

{

**int** m = inputMatrix.length;

**int** n = inputMatrix[0].length;

**double**[][] transposedMatrix = **new** **double**[n][m];

**for**(**int** x = 0; x < n; x++)

{

**for**(**int** y = 0; y < m; y++)

{

transposedMatrix[x][y] = inputMatrix[y][x];

}

}

**return** transposedMatrix;

}

/\*

\* Name: matrixPrint()

\* Parameters: double[][]

\* Output : void

\* Description: Prints a given Matrix

\*/

**public** **static** **void** matrixPrint(**double**[][] matrix)

{

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

System.*out*.print(matrix[i][j] + " ");

}

System.*out*.println();

System.*out*.println();

}

System.*out*.println();

}

}

## 1b) “generateSPDMatrix.m”

function A = generateSPDmatrix(n)

% Generate a dense n x n symmetric, positive definite matrix

A = rand(n,n); % generate a random n x n matrix

% construct a symmetric matrix using either

A = 0.5\*(A+A');

%A = A\*A';

% since A(i,j) < 1 by construction and a symmetric diagonally dominant matrix

% is symmetric positive definite, which can be ensured by adding nI

A = A + n\*eye(n);

End

## 1c) “TestMatrices.java”

