/\*CS2300

\* Assignment 3

\* Brett Ford

\*

\* The purpose of this program is to read a matrix and then perform some calculations on it. It begins by interpreting the matrix as a two by two matrix( a pair of vectors) and a point.

\* it solves for the x which would result in the point when multiplied by the vectors. it determines if the matrix is rank deficient, and returns that also.

\* it then moves on to calculate the eigenvalues, and eigenvectors of the matrix. it resolves by calculating the eigendecomposition of the matrix.

\* lastly, it treats the matrix as three points in space. it calculates the area of the volume of the triangle enclosed inside these three points.

\* It calculates the function of a plane which passes through these points. and then it calculates the distance of the point to the plane

\*

\*/

import java.io.\*;

import java.math.\*;

import java.util.\*;

public class CS2300P3BrettFord {

public static void main(String[] args) throws FileNotFoundException {

//it reads each file in turn and then uses a series of methods to print the results

String location = "test\_input\_1.txt";

Matrix matrix1=new Matrix(location);

findProductMatrix(matrix1);

calculateEigenThings(matrix1,location);

calculateAsPoints(matrix1);

calculate3D(matrix1);

location = "test\_input\_2.txt";

Matrix matrix2=new Matrix(location);

findProductMatrix(matrix2);

calculateEigenThings(matrix2,location);

calculateAsPoints(matrix2);

calculate3D(matrix2);

location = "test\_input\_3.txt";

Matrix matrix3=new Matrix(location);

findProductMatrix(matrix3);

calculateEigenThings(matrix3,location);

calculateAsPoints(matrix3);

calculate3D(matrix3);

}//end of main

//this method solves for x such that x times the twoXtwo matrix equals the point

public static void findProductMatrix(Matrix matrix) {

double vx1,vx2;

double A11= matrix.getElement(0, 0);

double A12= matrix.getElement(0, 1);

double A22= matrix.getElement(1, 1);

double A21= matrix.getElement(1, 0);

double vb1= matrix.getElement(0,2);

double vb2= matrix.getElement(1,2);

System.out.println(A11+" "+A12+"\t\t "+vb1);

System.out.println(A21+" "+A22+" times x = "+vb2);

double determinate = matrix.determinate();

//it only calculates the vlaues if the determinate is not zero

if(determinate !=0) {

System.out.println("the determinate is "+determinate);

double I11= (1/determinate)\*A22;

double I12= -(1/determinate)\*A12;

double I21= -(1/determinate)\*A21;

double I22= (1/determinate)\*A11;

System.out.println("the inverse matrix is:");

System.out.println(I11+"\t"+I12);

System.out.println(I21+"\t"+I22);

vx1=I11\*vb1+I12\*vb2;

vx2=I21\*vb1+I22\*vb2;

System.out.println("the vector x is:\n"+vx1+"\n "+vx2);

}else if( (Math.abs(A11/A21-A12/A22)<.0001) && (Math.abs(A11/A21-vb1/vb2)<.0001) ) {

System.out.println("This system is underdetermined");

}else {

System.out.println("This system is inconsistent.");

}

System.out.println(" ");

}//end of findProductMatrix()

//this calculates the eigenvalues, eigenvectors, and eigendecomposition of the matrix

public static void calculateEigenThings(Matrix matrix,String name ){

double a,b,c,d,E1,E2,A,D,x,ar1,ar2,br1,br2;

a= matrix.getElement(0, 0);

c= matrix.getElement(1, 0);

d= matrix.getElement(1, 1);

b= matrix.getElement(0, 1);

E1=(a+d + Math.sqrt(Math.pow((a+d),2) - 4\*(a\*d-b\*c)))/2;

E2=(a+d - Math.sqrt(Math.pow((a+d),2) - 4\*(a\*d-b\*c)))/2;

System.out.println("the eigenvalues of "+name+" are "+E1+" and "+E2);

A=a-E1;

D=d-E1;

x=-c/A;

Matrix EigenM=new Matrix(a-E1,b,c,D);

Matrix gScaler=new Matrix(1,0,x,1);

Matrix gaussM= gScaler.multiply22(EigenM);

if(gaussM.getElement(1,1)==0) {

ar1 = -b/A;

ar2=1;

A=a-E2;

D=d-E2;

x=-c/A;

br1 = -b/A;

br2=1;

double aRlength=Math.sqrt(Math.pow(ar2,2)+Math.pow(ar1,2));

double bRlength=Math.sqrt(Math.pow(br2,2)+Math.pow(br1,2));

//this prints the eigenvectors

System.out.println("Normalized eigenvectors are");

Matrix eigenvectorMatrix=new Matrix(ar1/aRlength,br1/bRlength,ar2/aRlength,br2/bRlength);

eigenvectorMatrix.print22();

Matrix eTranspose=eigenvectorMatrix.transpose22();

Matrix lambdaMatrix= new Matrix(E1,0,0,E2);

Matrix bigProduct = (eTranspose.multiply22(lambdaMatrix)).multiply22(eigenvectorMatrix);

bigProduct.print22();

//this prints if the eigendecomposition is equal to the origional matrix

if(bigProduct.isSame22(matrix)==true) {

System.out.println("the eigendecomposistion is equal to the origional matrix");

}else {

System.out.println("the eigendecomposistion is not equal to the origional matrix");

}

}

//this prints if no eigenvectors exist for a matrix.

else{

System.out.println("non-trivial eigenvector exists for the matrix"+name);

}

}//end of calculateEigenThings

public static void calculateAsPoints(Matrix matrix) {

double vA1,vA2,vB1,vB2;//stores information from the vectors

vA1=matrix.getElement(0, 1)-matrix.getElement(0, 0);//the vector from point 2 to point 1

vA2=matrix.getElement(1, 1)-matrix.getElement(1, 0);

vB1=matrix.getElement(0, 2)-matrix.getElement(0, 0);//the vector from point 3 to point 1

vB2=matrix.getElement(1, 2)-matrix.getElement(1, 0);

//this calculates and prints the area of the triangle

double T=(vA1\*vB2-vA2\*vB1)/2;

System.out.println("the area of a triangle formed by these points is "+T);

//this prints the equation of a line that passes through these points.

System.out.println(" equation of a line through these points is");

System.out.println(matrix.getElement(0, 0)+"+t\*"+vA1);

System.out.println(matrix.getElement(1, 0)+" "+vA2);

double wLength=Math.sqrt(vB1\*vB1+vB2\*vB2);

double vLength=Math.sqrt(vA1\*vA1+vA2\*vA2);

double cosAngle = (vA1\*vB1+vA2\*vB2)/(wLength\*vLength);

double distance = wLength\*(Math.sqrt(1-cosAngle\*cosAngle));

System.out.println("the distance from point 3 to the line is "+distance);

}//end of calculateAsPoints

//this method makes calculation of the matrix as though it represented points in 3D space

public static void calculate3D(Matrix matrix) {

double pA1,pA2,pB1,pB2,pA3,pB3,pC1,pC2,pC3;//stores information from the matrix

pA1=matrix.getElement(0, 0);

pA2=matrix.getElement(1, 0);

pA3=matrix.getElement(2, 0);

pB1=matrix.getElement(0, 1);

pB2=matrix.getElement(1, 1);

pB3=matrix.getElement(2, 1);

pC1=matrix.getElement(0, 2);

pC2=matrix.getElement(1, 2);

pC3=matrix.getElement(2, 2);

//m = (p2 + p1)/2 //the point midway between them

double pM1=(pA1+pB1)/2;

double pM2=(pA2+pB2)/2;

double pM3=(pA3+pB3)/2;

double distance=Math.sqrt((pA1-pB1)\*(pA1-pB1)+(pA2-pB2)\*(pA2-pB2)+(pA3-pB3)\*(pA3-pB3));

System.out.println("the distance is between the points is "+distance);

//n = (p2 - p1)/||p2 - p1|| //The unit length vector normal

double vN1=(pA1-pB1)/distance;

double vN2= (pA2-pB2)/distance;

double vN3=(pA3-pB3)/distance;

//n1x1 + n2x2 + n3x3 - (n1m1 + n2m2 + n3m3)= 0

System.out.println("the equation for the palne is:\n"+vN1+"xX1+"+vN2+"xX2+"+vN3+"xX3+"+(vN1\*pM1+vN2\*pM2+vN3\*pM3)+"=0");

//d = n1p3,1 + n2p3,2 + n3p3,3 - (n1m1 + n2m2 + n3m3)

double distToPlane = vN1\*pC1+vN2\*pC2+vN3\*pC3-(vN1\*pM1+vN2\*pM2+vN3\*pM3);

System.out.println("the distance of point c to the plane is "+distToPlane);

}

}//end of class

//this class is used to store methods and contains methods for them

class Matrix{

double[][] threeThree= new double[3][3];

public Matrix() {

}

public Matrix(double a,double b,double c,double d) {

this.threeThree[0][0]=a;

this.threeThree[0][1]=b;

this.threeThree[1][0]=c;

this.threeThree[1][1]=d;

}

//this constructor generates a matrix from a file

public Matrix(String location) throws FileNotFoundException {

File fileName = new File(location);

Scanner readFile = new Scanner(fileName);

int r=0;

while(readFile.hasNextDouble()) {

for(int c=0;c<3;c++) {

this.threeThree[r][c]=readFile.nextDouble();

}

r++;

}

readFile.close();

}

public double[][] getThreeThree() {

return threeThree;

}

//this getter returns elements from a matrix

public double getElement(int r,int c) {

return threeThree[r][c];

}

public void setElement(int r,int c,double number) {

this.threeThree[r][c] = number;

}

public double determinate() {

double determinate = threeThree[0][0]\*threeThree[1][1]-threeThree[0][1]\*threeThree[1][0];

return determinate;

}

//a method that return the product of the top left four squares of two matrices

public Matrix multiply22(Matrix matrix2) {

double P11,P12,p21,p22;

P11=threeThree[0][0]\*matrix2.getElement(0, 0)+threeThree[0][1]\*matrix2.getElement(1, 0);

P12=threeThree[0][0]\*matrix2.getElement(0, 1)+threeThree[0][1]\*matrix2.getElement(1, 1);

p21=threeThree[1][0]\*matrix2.getElement(0, 0)+threeThree[1][1]\*matrix2.getElement(1, 0);

p22=threeThree[1][0]\*matrix2.getElement(0, 1)+threeThree[1][1]\*matrix2.getElement(1, 1);

Matrix prodMatrix=new Matrix(P11,P12,p21,p22);

return prodMatrix;

}

//a method for printing only 2X2 matrices

public void print22() {

System.out.println(threeThree[0][0]+"\t"+threeThree[0][1]);

System.out.println(threeThree[1][0]+"\t"+threeThree[1][1]);

}

// a method for transposing a only 2X2 matrices

public Matrix transpose22() {

Matrix transpose = new Matrix(threeThree[0][0],threeThree[1][0],threeThree[0][1],threeThree[1][1]);

return transpose;

}

//a method for checking if two matrices are the same.

public boolean isSame22(Matrix matrix2) {

for(int r=0;r<2;r++) {

for(int c=0;c<2;c++) {

if(Math.abs(threeThree[r][c]-matrix2.getElement(r, c))>.0001) {

return false;

}

}

}

return true;

}

}//end of Matrix