Pràctica Matrix

Informe de la pràctica de matrius

Introducció

Donat dos arxius, un per fer els tests I l'altre per programar cada funció hem de fer que el programa executi correctament tots els tests de l'arxiu MatrixTest, una vega dit això.

En matemàtiques, una matriu és una taula rectangular de nombres o, més generalment, d'elements d'una estructura algebraica de forma d'anell. Els valors per les matrius són números reals a menys que es digui el contrari.

Definirem una matriu en java com un array bidimensional de doubles. Exemple:
double[][] mat = { {1,2,3}, {4,5,6}, {7,8,9} }
Podeu imprimir una matriu de la següent manera:

```
static void printMat(double[][] mat) {
	for (int i = 0; i < mat.length; i++) {
		for (int j = 0; j < mat[i].length; j++) {
			System.out.printf("%06.2f ", mat[i][j]);
		}
		System.out.println();
	}
}
```

Les operacions amb matrius aniran totes dins una classe anomenada "Matrix":

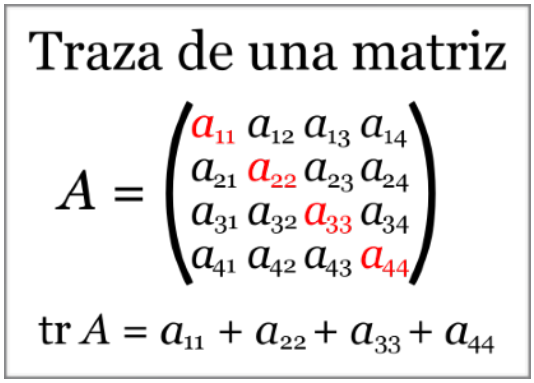
```
class Matrix {
	static double[][] add(double[][] mat1, double[][] mat2) { ...
	}

	static double[][] mult(double[][] mat1, double[][] mat2) {
	…
	}
	etc...
}
```

Els procediments mai modificaran les matrius (arrays) que passem per paràmetre. Igual que els Strings, considerarem les matrius com a IMMUTABLES. Per tant, sempre que heu de tornar un resultat en forma d'array, genereu un objecte nou:
double[][] result = new double[v][h]

Heu d'implementar les operacions següents:

- Suma/resta de matrius
- Traça d'una matriu quadrada (suma de la diagonal principal)
- Multiplicació de dues matrius compatibles
- Multiplicació d'una matriu per un número
- Potència d'una matriu
- Determinant d'una matriu
- Càlcul d'una sub-matriu
- Càlcul de la transposada d'una matriu
- Càlcul de la inversa d'una matriu
- "Divisió" de matrius (multiplicació per la inversa)
- Test d'ortogonalitat d'una matriu
- Resolució de sistemes d'equacions de múltiples incògnites emprant la regla de cramer

Funció TRACE



Per començar farem el traç de la matriu donada, per això el que hem de fer es crear la variable auxiliar dintre de la funció que ens permeti emmagatzemar el resultat per a quan retornem el resultat.
El traç de la matriu es la seva diagonal, per això el que hem de fer es calcular-la amb un bucle que la recorri de manera diagonal:

```
for(int i = 0; i < mat.length; i++){
    Trazo += mat[i][i];
//Treiem el valor final que ens indica la traça de la matriu que retornarem
}
```

Com podem veure al troç de codi anterior, el que tenim es la diagonal de la matriu, ara el que hem de fer es retornar-la per a que el programa la pugui executar amb un:

```
return Trazo;
```

La formula lineal per trobar el traç de la matriu es la següent:

$$\operatorname{tr}(A + B) = \operatorname{tr}(A) + \operatorname{tr}(B)$$

Funció ADD

La funció que segueix es la de la suma. Ens donen dues matrius que s'han de sumar, per fer-ho crearem una variable auxiliar que ens permeti emmagatzemar el valor final, per començar donarem per als seus valors la següent informació:

```
double [][] suma = new double[mat1.length][mat1[0].length];
```

Amb aquesta variable ja inicialitzada el que farem serà crear dos bucles que recorrin la primera matriu, un de manera horitzontal i l'altra de manera vertical. El resultat del bucle amb la informació de la suma és el següent:

```
for(int i = 0; i < mat1.length; i++){
    for(int j = 0; j < mat1[0].length; j++){
        suma[i][j] = mat1[i][j] + mat2[i][j];//Sumam les dues matrius
    }
}
```

El que hem fet ens dona que sumant **mat1[i][j] + mat2[i][j]** on i es el valor de la posició horitzontal, i, j és el valor de la posició vertical on ens trobam obtenim que **suma[i][j] conté el resultat de la suma que hem realitzat**. Per tant el que ens queda es que hem trobat el resultat de la suma, així que el que ens queda es retornar-ho.

```
return suma;
```

La formula de la suma es la següent:

$$\boldsymbol{Z}_{2x2} + \boldsymbol{X}_{2x2} = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} + \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} = \begin{pmatrix} z_{11} + x_{11} & z_{12} + x_{12} \\ z_{21} + x_{21} & z_{22} + x_{22} \end{pmatrix}$$

Funcio MULT (amb dos matrius)

Tenim dues funcions de multiplicació al codi font, aquesta és la funció que ens permet multiplicar dues matrius. S'ha de complir un requisit per a poder multiplicar-les, i es que han de ser de la mateixa dimensió, es a dir, que per poder multiplicar-les han de ser 2x2, 3x3, etc.

Abans de començar a operar el que farem serà crear com a cada funció fins ara una variable auxiliar que emplearem per emmagatzemar el resultat. A aquesta variable assignarem el valor de la longitud de la primera matriu a la primera dimensió, i la longitud de la segona matriu a la segona dimensió de l'array.

Quan hem creat la variable el que ens queda es realitzar la multiplicació, primer el que hem de fer es un condicional assignant que la condició ha de ser que la horitzontal de la primera matriu ha de fer la mateixa mida que la segona matriu, el codi per això ens queda de la següent manera:

```
if(mat1[0].length == mat2.length)
```

Continuarem creant 3 bucles per recorrer les dues matrius així que farem els bucles per controlar la longitud de les dues matrius, per a I i J assignarem que I es la longitud de la segona matriu, per a J assignarem la longitud horitzontal de la segona matriu i per a K assignarem la longitud de la primera matriu sencera, el codi ens quedarà així:

```
for(int i = 0; i < mat2.length; i++){
    for(int j = 0; j < mat2[0].length; j++){
        for(int k = 0; k < mat1.length; k++){
            multiplicacioMatrius[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
}
```

Amb aquest bucle ens trobarem amb el resultat de la multiplicació, el que ens queda es retornar-ho amb un return. Així ja tendríem feta la primera funció de multiplicació.

La formula de la funció es la següent:

$$(1 \quad -2 \quad 3) \cdot \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} =$$
$$= (1 \cdot 4 - 2 \cdot 5 + 3 \cdot 6) =$$
$$= (4 - 10 + 18) =$$
$$= (12)$$

Funció POWER

Aquestá funció el que fa es l'exponent d'una matriu, ens donen la matriu amb la que haurem de multiplicar-la per ella mateixa el número de vegades del seu exponent. Per començar comprovarem el valor de l'exponent.

En cas de que l'exponent sigui 0 el que farem serà calcular la longitud i profunditat de la matriu amb dos bucles, un que recorri les files y l'altra que recorri les columnes, una vegada fet això assignarem a la matriu **powerUp[i][j]** el valor de multiplicar **mat[i][j] * 0 i sumar-li 1**. Seguidament comprovarem que I sigui igual a J per assignar un 1 o un 0 com a valors. El codi ens quedaría així:

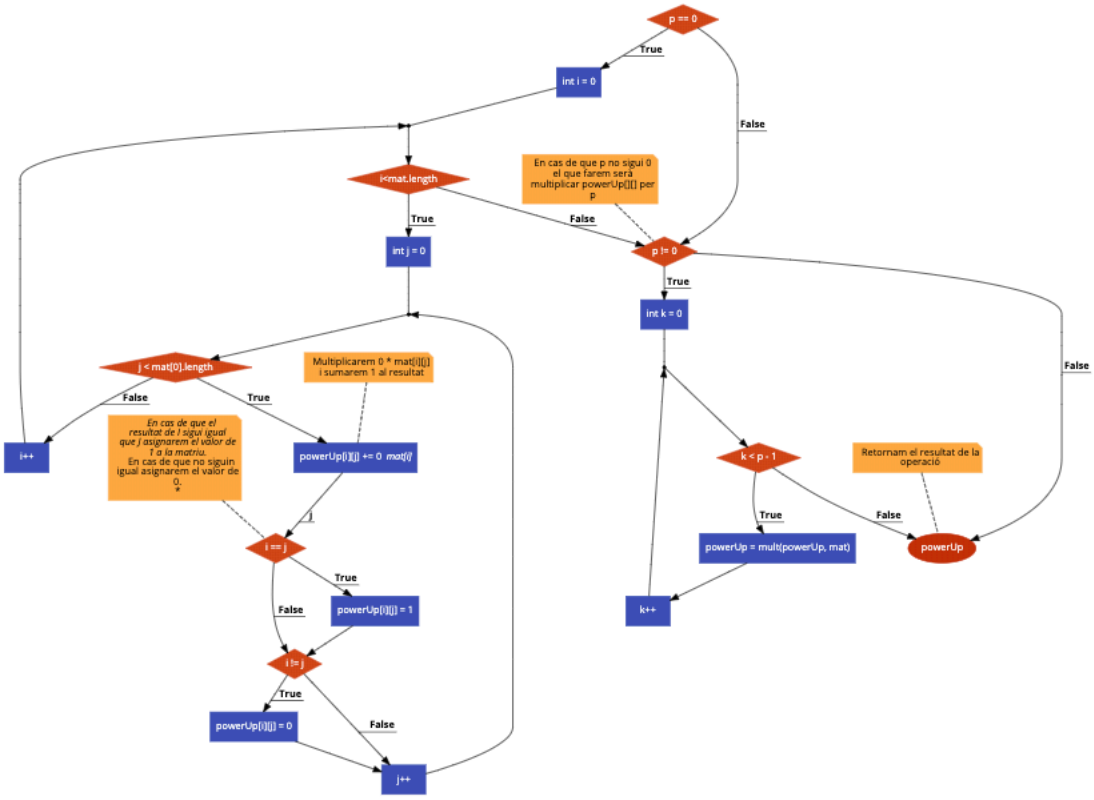
```
if (p == 0) {
    for(int i = 0; i<mat.length;i++){
        for(int j = 0; j < mat[0].length; j++){
            powerUp[i][j] += 0 * mat[i][j];

            if (i == j) {
                powerUp[i][j] = 1;
             }if(i != j) {
                powerUp[i][j] = 0;
            }
        }
    }
}
```

En cas de que el valor de l'exponent sigui diferent a 0 realitzarem un bucle on anirà restant 1 a l'exponent cada vegada que realitzi la operació, per tant ens queda el següent fragment de codi:

```
if (p != 0) {
    for (int k = 0; k < p - 1; k++) {
        powerUp = mult(powerUp, mat);
    }
}
```

La formula per fer l'exponent de la matriu es la següent:

I el seu consegüent diagrama de flux:



Funció DIV

Aquesta es la funció per poder dividir les matrius, ens passen dues matrius per realitzar la seva divisió. Començarem per comprovar que cap de les dues matrius siguin nules, en cas de que ho siguin retornarem un missatge d'error per avisar a l'usuari i un valor null.

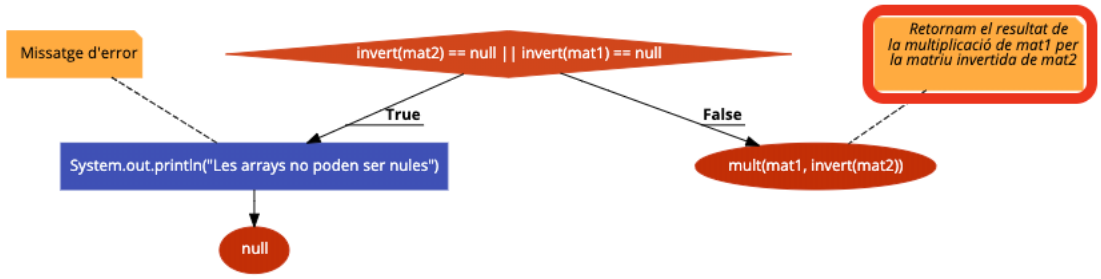
En cas contrari, es a dir, que cap de les dues siguin nules el que farem serà multiplicar la primera matriu per la inversa de la segona matriu, amb aquestà formula obtindrem el resultat de la divisió de la matriu. Per això el retornarem de la següent manera.

Aquesta es la funció ens fa cridar a dues funcions externes que són les funcions de multiplicació (mult) i inversa (invert).

El codi es el següent:

```
if(invert(mat2) == null || invert(mat1) == null){
    System.out.println("Les arrays no poden ser nules");
    return null;
}else{
    return mult(mat1, invert(mat2));
}
```

I el diagrama de flux de la funció:



Funció SUBMATRIX

Aquesta es la funció per calcular la submatriu de la matriu original que ens passen, també ens passaràn valors de X i de Y amb els que haurem de calcular el seu diferencial, per això el que farem serà calcular el diferencial de X i de Y de la següent manera:

```
int diferencialX = x2 - x1 + 1;
int diferencialY = y2 - y1 + 1;
```

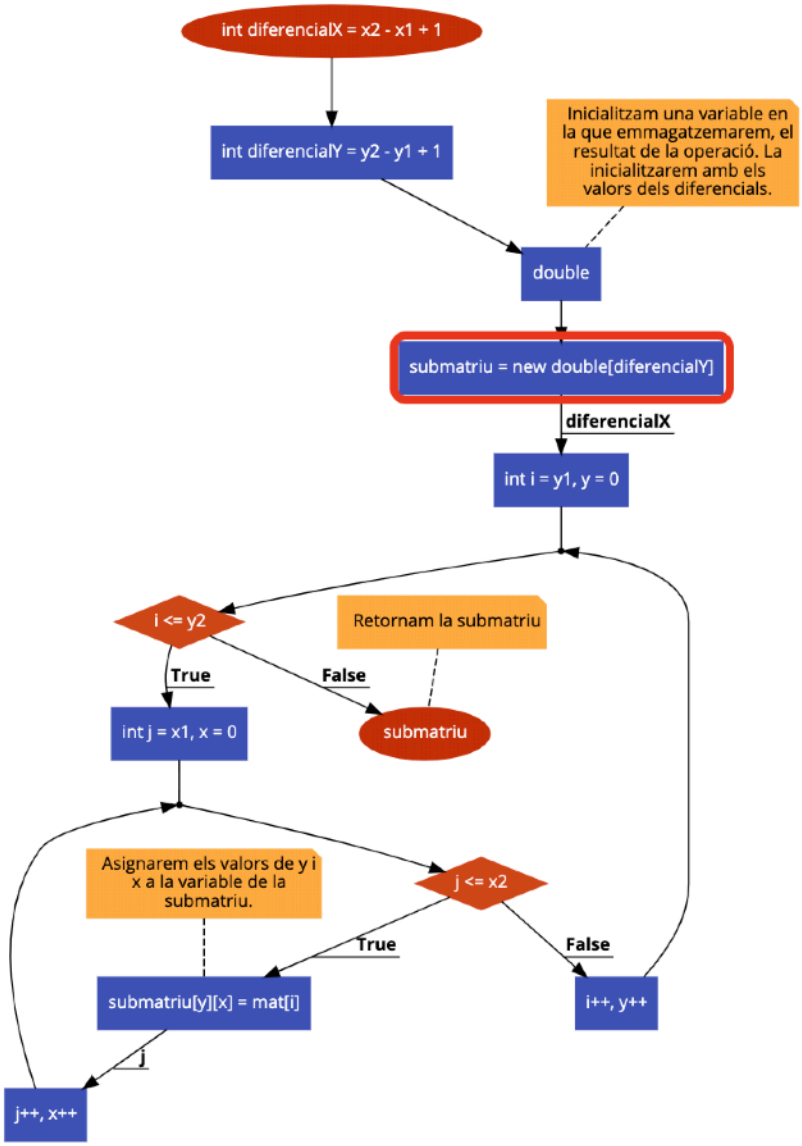
Una vegada tenim els diferencials inicialitzarem una nova matriu amb els valors de les seves dimensions assignats amb els dos diferencials que hem calculat abans.

Quan tenim la nova matriu cream dos nous bucles on anirem incrementant els valors de x i de y que assignarem a la submatriu **calculant que I i J siguin menors que y2 i x2**, per tant ens quedarà el següent fragment:

```
for (int i = y1, y = 0; i <= y2; i++, y++) {
    for (int j = x1, x = 0; j <= x2; j++, x++) {
        //Asignarem els valors de y i x a la variable de la submatriu.
        submatriu[y][x] = mat[i][j];
    }
}
```

Una vegada calculat la submatriu la retornarem.

I el diagrama de flux és el següent:



Funcio MULT (matriu * decimal)

Aquesta funció és igual a la funció de multiplicar matrius però en aquest cas el que farem serà multiplicar la matriu per un número decimal que ens passen amb la funció.

Per començar crearem una nova matriu amb les dimensions de la original per poder operar amb ella. Seguidament crearem dos bucles per recorrer de manera horitzontal i verticalment, una vegada la anam recorrent assignam a la matriu que emmagatzema el resultat el valor de multiplicar la posició en la que ens trobem per el número decimal que ens donen. El codi es el següent:

```
for (int i = 0; i < mat.length; i++) {
    for (int j = 0; j < mat[0].length; j++) {
        multiplicacioEnters[i][j] = mat[i][j] * n;
    }
}
```

Una vegada tenim tot això fet retornarem el resultat.

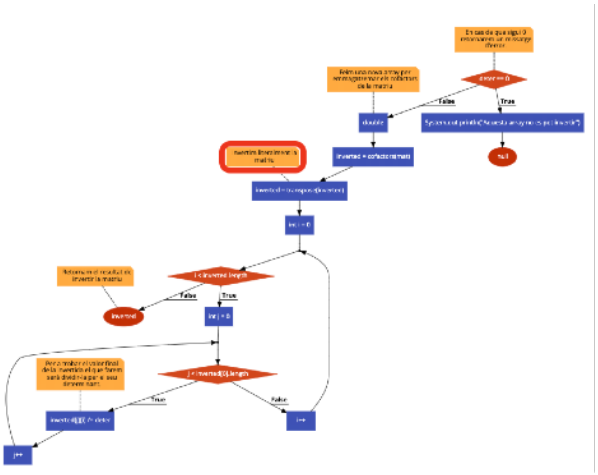
Funció INVERT

Amb aquesta funció el que ens permet es trobar la inversa de la matriu, però per això ens haurem de fer servir de la funció del determinant, transpose, cofactors. El primer pas a realitzar es obtenir el determinant de la matriu, en cas de que sigui 0 retornarem un valor null i un missatge d'error. Si no es 0 continuarem amb el programa, la següent passa es crear una variable auxiliar on emmagatzemar el resultat final. De moment el que tenim es això:

```
double deter = determinant(mat);
//En cas de que sigui 0 retornarem un missatge d'error.
if(deter == 0){
    System.out.println("Aquesta array no es pot invertir");
    return null;
}
//Feim una nova array per emmagatzemar els cofactors de la matriu
double[][] inverted = cofactors(mat);
```

Ara el que tenim es una variable amb el valor dels cofactors de la matriu, amb això el que hem de fer es obtenir la transposició de la matriu amb la funció transpose, fet això, hem de trobar el valor de la matriu invertida amb dos bucles per recorrer la matriu, el que farem amb això serà dividir les posicions [j][i] per el determinant que hem trobat al començament del programa. Amb això ja tenim la matriu inversa, ens queda retornar el seu valor.

Diagrama de flux:

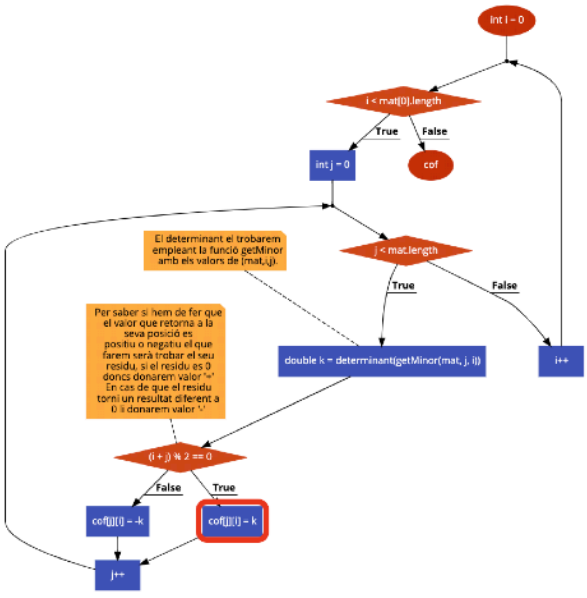


Funció COFACTORS

Amb aquesta funció podem trobar els cofactors d'una matriu que ens passen a la funció. La primera passa serà amb 2 bucles per recórrer l'array de manera vertical i horitzontal. dintre d'aquest bucle crearem una variable on emmagatzemarem el determinant de realitzar el getMinor de mat,I i J. També crearem un if dintre del bucle per saber si el valor serà positiu o negatiu, per saber-ho haurem de sumar i+j i trobar el seu residu, en cas de que sigui 0 el valor serà positiu, i en cas de que sigui diferent a 0 el valor serà negatiu, aquest sería el codi per realitzar la operació:

```
for (int i = 0; i < mat[0].length; i++) {
    for (int j = 0; j < mat.length; j++) {
        double k = determinant(getMinor(mat, j, i));
        if((i + j) % 2 == 0){
            cof[j][i] = k;
        } else {
            cof[j][i] = -k;
        }
    }
}
```

Ara amb això fet retornarem el valor de cof, que es la variable que emmagatzema el valor final.
Aquest sería el diagrama de flux:



Funció getMinor

Aquesta es la funció que ens permet es trobar el valor més petit de la columna i la vertical en funció de la posició en la que ens trobem dins de la matriu. Per fer això el que hem de fer es crear la variable on hem de emmagatzemar el resultat, fet això hem de recórrer la array amb dos bucles i dintre del bucle comprovar que si el valor de j sigui diferent del de y assignarem les dimensions de la variable auxiliar amb els valors que el programa consideri que son els més idonis per a l'operació dels que obtenim amb mat[i][j].

Amb això ja obtendríem els minors. Aquí tenim el codi i el diagrama de flux:
Codi:

```
for (int i = 0; i < mat.length; i++) {
    for (int j = 0; i != x && j < mat[i].length; j++) {
        if (j != y) {
            minor[i < x ? i : i - 1][j < y ? j : j - 1] = mat[i][j];
        }
    }
}
```

Diagrama:

Funció DETERMINANT

Cream un valor de 0 per començar el programa, fet això continuarem comprovant la profunditat que té la matriu de la que volem trobar el determinant així que el que farem serà crear un condicional per a les dimensions de 1 i de 2x2, en els que calcularem diferentment.

En cas de que la longitud de la matriu sigui de 1 farem un return de les posicions [0][0] de la matriu original.

```
if(mat.length == 1){
    return mat[0][0];
}
```

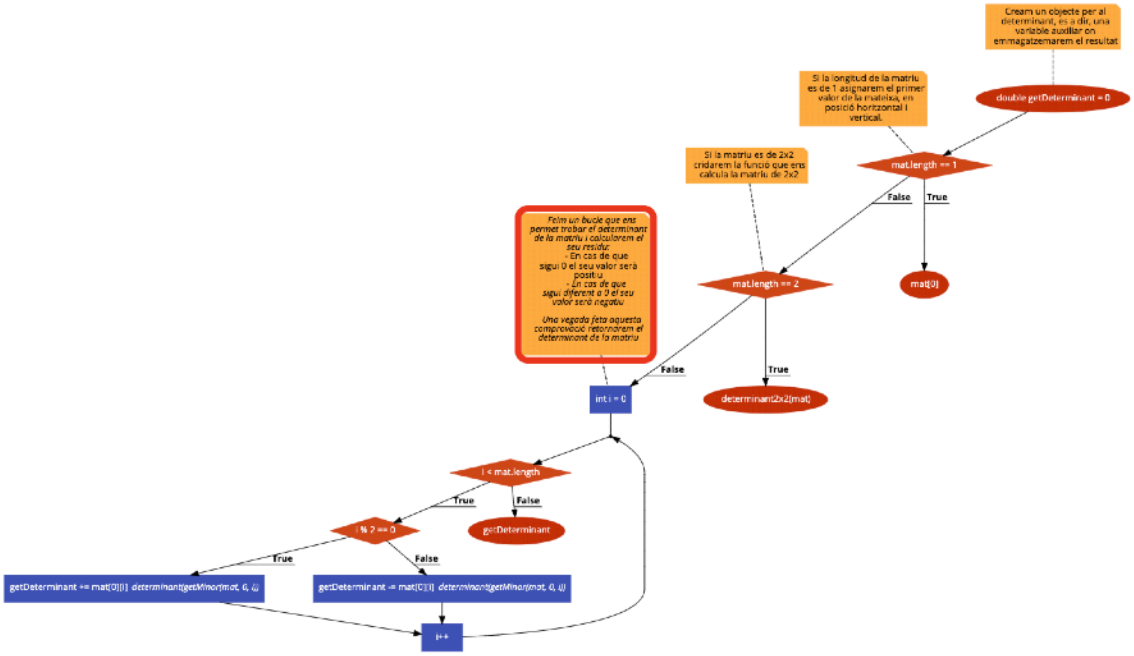
En cas de que sigui la longitud 2x2 retornarem el valor calculat amb una funció que calcula el determinant 2x2, es a dir, cridarem a una funció de fora per fer la operació.

```
if(mat.length == 2){
    return determinant2x2(mat);
}
```

En cas de que sigui superior a 2x2 crearem un bucle amb un condicional dintre del bucle que ens dirà si el valor ha de ser positiu o negatiu en funció del residu que obtenim. Aquí tenim el codi que ho calcula, empleant-se a ella mateixa la funció i cridant a la funció de getMinor.

```
for (int i = 0; i < mat.length; i++) {
    if (i % 2 == 0) {
        getDeterminant += mat[0][i] * determinant(getMinor(mat, 0, i));
    } else {
        getDeterminant -= mat[0][i] * determinant(getMinor(mat, 0, i));
    }
}
```

Aquí tenim el diagrama de flux de la funció:



Funció TRANSPOSE

Aquesta funció la farem servir per a donar-li la volta a la matriu, per això el que hem de fer es crear com a cada funció la variable auxiliar a la que hem de assignar-li el valor de la longitud de la matriu abans de crear els dos bucles que ens permetrà donar-li la volta. Aquest es el codi amb els dos bucles per donar la volta a la matriu, el que feim es assignar a Transposicio[j][i] els valors de mat[i][j]:

```
for(int i = 0; i < mat[0].length; i++){
    for (int j = 0; j < mat.length; j++) {
        //Donam la volta a la matriu.
        Transposicio[j][i] = mat[i][j];
    }
}
```

Funció isOrtho

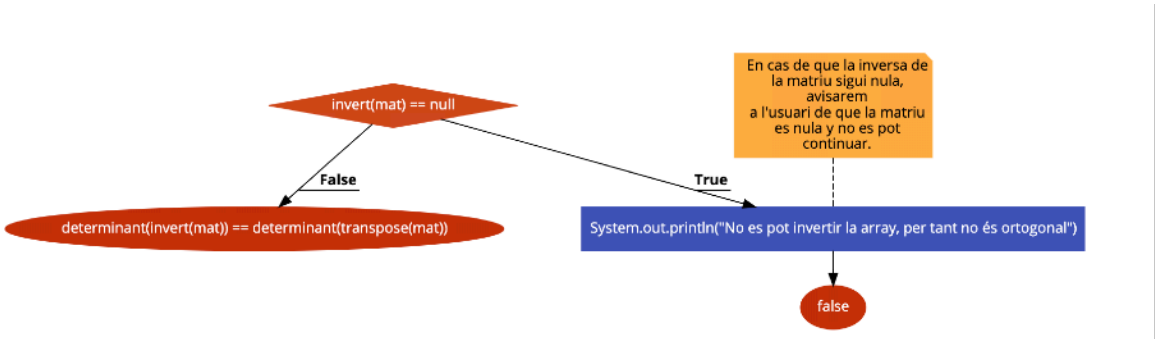
Aquesta funció ens retornarà un valor boolean de vertader o fals en funció de si la matriu es pot invertir, i en cas de que es pugui ha de ser igual al determinant de la matriu transposada. Per això hem de crear el condicional que ens comprovi que la matriu es pot invertir o no. En cas de que no es pugui invertir el que farem serà retornar un valor false i un missatge de error.

```
if(invert(mat) == null){
    System.out.println("No es pot invertir la array, per tant no és ortogonal");
    return false;
}
```

Si es pot invertir comprovarem que el determinant de la matriu inversa sigui igual al determinant de la matriu transposada, si son iguals retornarem un valor true.

```
else {
    return determinant(invert(mat)) == determinant(transpose(mat));
}
```

Aquest es el diagrama de flux del programa:



Funció CRAMER

La darrera funció que toca es la de el mètode cramer, per poder operar aquesta funció hem de crear 3 arrays, una que emplearem per al resultat, aquesta serà d'una dimensió únicament.

Les altres dues arrays seràn bidimensional i una la emplearem per als cofactors i l'altra per a trobar les incògnites. El primer pas que realitzarem serà trobar els cofactors, per això el que farem serà dos bucles per recórrer la matriu, aquest es el codi dels dos bucles:

```
for (int i = 0; i < cofactors.length; i++) {
    for (int j = 0; j < cofactors[0].length; j++) {
        //Asignam el valor a l'array cofactors[][]
        cofactors[i][j] = mat[i][j];
    }
}
```

La següent passa serà trobar l'incòginta i per això farem un únic bucle per recórrer la matriu i afegirem la darrera columna de la matriu original a la incòginta, el codi ens quedaría així:

```
for (int i = 0; i < incognita.length; i++) {
    incognita[i][0] = mat[i][mat[0].length - 1];
}
```

Cream una nova variable per a emmagatzemar el determinant dels cofactors, i cream un condicional per a comprovar que el seu valor no sigui null. Seguidament cream un bucle amb la longitud dels cofactors que dintre tendrà dos bucles més que estaràn separats, també crearem una copia de la matriu amb els valors dels cofactors per a cada dimensió de l'array.

```
if(determinantMatriu == 0) return null;
    for (int i = 0; i < cofactors.length; i++) {

        //Declaram la variable que emplearem per trobar el resultat del mètode de
cramer.
        double[][] segonaMatriu = new double[cofactors.length][cofactors[0].length];
```

La següent passa serà crear un bucle dins de un altra per a poder recórrer la matriu, on assignarem els valors de [j][k] a la variable que acabam de crear. Fet això el que feim serà un altre bucle que ens permeti substituir els valors de la matriu per els de l'incògnita.

```
for (int j = 0; j < segonaMatriu.length; j++) {
    for (int k = 0; k < segonaMatriu[0].length; k++) {
        segonaMatriu[j][k] = cofactors[j][k];
    }
}
//Substituim els valors de la matriu per els de la incògnita
for (int j = 0; j < cofactors[0].length; j++) {
    segonaMatriu[j][i] = incognita[j][0];
}
```

La darrera passa serà assignar valor a l'array del resultat del determinant de la variable que hem creat fa poc dividit per el determinant dels cofactors. Això ja ens permet retornar el valor del resultat.

```
resultat[i] = determinant(segonaMatriu) / determinantMatriu;
```

I aquest es el diagrama de flux de la funció: