E.T.N 3 D.E 9º - Laboratorio III Año 2010

Docente a Cargo: *Fernández, Nicolás*

TrabajoPrácticoNro. 1, Trimestre 1: Un poco de Matemática no hace mal

Fecha de Realización: 04-04-2010

Nota: 8(ocho)

Observaciones:
***El informe esta muy bueno y claro.**
***Si le mando los siguientes parámetro 0 o 1 como dimension, lanza exception.**
***El resto del trabajo esta muy bien. Si bien no se hizo el método de Gauss, se hizo los dos auxiliares con lo cual lo contabilizo.**
***Se podría haber reducido en varios métodos, el uso de variables auxiliares.**

Apellidos: Vileriño

Nombres: Silvio

Rotación: Grupo B

Año: **6to 1ra**

Firma del docente:

Introduccion:

Se pide un trabajo practico que pueda realizar los siguientes puntos:

1) Calcular el determinante por el método de los cofactores, tratando de hacer la menor cantidad de cuentas.
2) Utilizar el método de Sarrus si el coeficiente de la matriz es 2 o 3 .
3) Para ciertos casos particulares, triangular la matriz por el método de Gauss.
4) Indicar si la matriz es definida positiva o no.
5) Suponiendo que se dispone de un vector para armar un sistema de nxn ecuaciones, se debe ver la posibilidad de utilizar la regla de Crámer y en caso que se pueda dar la solución x1,x2,…..,xn
6) Permitir saber si dos matrices son iguales.
7) Permitir saber cuantas instancias de la clase matriz se generaron.
8) Hallar la Matriz Inversa (A^-1)
9) Permitir saber si la matriz es idempotente (A.A=A)

Desarrollo del trabajo practico:

Paso a explicar cada uno de los métodos que desarrolle para resolver el practico.

Interfaz de la clase (Metodos Publicos):
Constructores:

Matriz(int n,int minimo,int maximo) (constructor)

Matriz(int [][],int dimension) (constructor con matriz de origen)

Metodos Publicos:

int getN() : Devuelve la dimension del atributo mat

int getElementobyPos(int fila,int columna): Devuelve el elemento mat[fila][columna]

int getInstances() (estatico):devuelve cantidad de instancias de la clase Matriz

void printMat(): imprime la matriz

boolean matCmp(Matriz,Matriz,dimension): Compara dos objetos de la clase Matriz

int getDeterminante(): Devuelve el determinante de la matriz, ya sea por sarrus o cofactores segun corresponda

bool defPositiva(): Devuelve true o false segun la Matriz sea o no definida positiva

void triangularGauss() : triangula la matriz por el metodo de gauss

void getCramer(int solucion[]) : calcula los valores Xn, hay que pasarle la columna solucion de longuitud n

void getInversa(): calcula la matriz inversa y la imprime,no devuelve ya que la matriz inv. es de punto flotante

bool esIdempotente(): devuelve true o false segun la matriz sea o no idempotente

Detalle de los Metodos:

Metodo de Determinantes:

```
public int getDeterminante(){
    if(n<4)//Si es de dimension menor o igual a 3, saca determinante por sarrus sino, llama al
        return(getSarrusDet());
    return(getCofDet());
}
```

Este método bifurca a 2 metodos mas específicos de calculo determinantes:

Sarrus: Calcula determinantes para matrices de dimensión menor o igual a 3

```
private int getSarrusDet(){
    //El metodo almacena los productos de las diagonales en sumas y restas y los va volcando en
    int det=0,sumas=1,restas=1,a=0,posi=0;
    if(n==1) return(mat[0][0]);//dimension 1(validacion para adjuntas peque;as)
    if(n==3){
        for(a=0;a<n;a++){
            //Diagonales Suma
            for(posi=(a*n);posi<=((a*n)+8);posi+=4)//posi=(a*n);posi<=((a*n)+((2*n)+2));posi+=(r
                if(posi>8)
                    sumas*=mat[(posi-9)/(int)n][(posi-9)%n];
                else
                    sumas*=mat[posi/(int)n][posi%n];
            det+=sumas;
            sumas=1;
            //Diagonales Resta
            for(posi=(a*n)+2;posi<=(a*n)+6;posi+=2)
                if(posi>8)
                    restas*=mat[(posi-9)/(int)n][(posi-9)%n];
                else
                    restas*=mat[posi/(int)n][posi%n];
            det-=restas;
            restas=1;
        }
    }else
        det=(mat[0][0]*mat[1][1])-(mat[0][1]*mat[1][0]);
    return(det);
}
```

Dimension 3

El método recorre las diagonales y almacena la sumas y restas (respectivamente) de los productos de los 3 elementos de la diagonal en una variable.

Dimension 2

El método devuelve directamente el determinante de 2x2.

Dimension 1

Devuelve el primer elemento

Cofactores: Calcula determinantes para matrices de dimensiones mayores por medio de la siguiente formula:

$$|\mathbf{A}| = \sum_{i=1}^{n} a_{1i} \cdot \mathbf{C}_{1i}$$

```
private int getCofDet(){
    int retorno=0;
    for(int col=0;col<n;col++)//Saca cofactores de los elementos de la primer fila
        if(mat[0][col]!=0){
            Matriz A=new Matriz(getAdjMatbyPos(0,col),(n-1));//creo una matriz de n-1*n-1 y almaceno
            retorno+=mat[0][col]*Math.pow((-1),col)*A.getDeterminante();//Aij*(-1)^i+j*|ADJij|   (el
        }
    return(retorno);
}
```

Recorro la primera fila de la matriz:
Creo un objeto matriz con el atributo mat seteado desde la matriz devuelta por la función getAdjMatbyPos(fila,columna) que devuelve la adjunta del elemento pasado por parámetros.

La variable retorno almacena la sumatoria de:

Aij.(-1)^(i+j).|Det(ADJ(Aij)) |

Metodo getAdjMatbyPos():

```
private int[][] getAdjMatbyPos(int fila,int columna){
    //Devuelve una matriz de (n-1)*(n-1) anulando la fila y columna pasada por parametros.
    int aux[][]=new int[n-1][n-1];
    int posiaux=0;
    for(int posi=0;posi<(n*n);posi++)
        if(((posi/(int)n)!=fila) && ((posi%n)!=columna)){
            aux[posiaux/(int)(n-1)][posiaux%(n-1)]=mat[posi/(int)n][posi%n];
            posiaux++;
        }
    return(aux);
}
```

Devuelve la matriz de dimension (n-1) con la fila y columna anuladas de los parámetros fila y columna.

Triangulacion por método de Gauss:

Este método no me salió.

Definida Positiva:

```
public boolean defPositiva(){
    if(mat[0][0]<=0)return(false);//comprueba el elemento 0,0
    int a=1;
    Matriz A;
    do{//Comprueba que de menor(2x2) a mayor(N-1xN-1) , los determinantes de las matrices reducid
        A=new Matriz(getMatReducida(a),a+1);
        a++;
    }while((a<n) && (A.getDeterminante()>0));
    if(getDeterminante()<=0)return(false);//Comprueba el determinante de mat
    return!(a<n);//si no llego a n, es porque antes habia un determinante que no cumplia con la c
}
```

Comprueba los determinantes de las matrices superiores izquierdas de dimension desde 1 hasta n.
Retorno (a<n) , si es verdadero, es porque antes de terminar las verificaciones de los determinantes de las matrices reducidas, algún determinante no cumplió la condición >0.

Regla de Cramer:

```
public void getCramer(int col_solucion[]){
    int a=0,col=0,fil=0;
    Matriz A;
    int aux[][];
    System.out.println("\nSistema de: " + n + "x" + n + ":\n");
    for(fil=0;fil<n;fil++){
        for(col=0;col<n;col++)
            System.out.print("+(" + mat[fil][col] + ")*x" + col + " ");
        System.out.println("  =   (" + col_solucion[fil] + ")");
    }
    System.out.println("\nMatriz Asociada al Sistema:\n");
    for(fil=0;fil<n;fil++){
        for(col=0;col<n;col++)
            System.out.print("(" + mat[fil][col] + ")");
        System.out.println(" | (" + col_solucion[fil] + ")");
    }
    System.out.println("");
    float soluciones[]=new float[n];
    int detMat=getDeterminante();
    if(detMat!=0){//Compruebo DET A !=0 para obtener Cramer
        for(a=0;a<n;a++){//columnas Xa;Soluciones Xa
            aux=getCopyMat();//Obtengo una copia exacta de mat
            aux=setColInMat(a,col_solucion,aux);//intercambio por la columna solucion para cada
            A=new Matriz(aux,n);
            soluciones[a]=A.getDeterminante()/(float)detMat;
            System.out.println("Solucion X" + a + " = (" + soluciones[a] + ")");
        }
    }else
        System.out.println("Determinante Nulo, Matriz Invalida para aplicar la regla de Cramer."
}
```

Este método devuelve las soluciones Xn del sistema NxN , pasándole un vector solución.Tiene como requisito que el determinante de la matriz sea distinto de 0.
La formula es: det(Xn)/det(mat)
Donde Xn es la matriz con la columna Xn reemplazada por el vector solución.

getCopyMat() y setColInMat()

```
private int[][] getCopyMat(){//Devuelve una copia exacta del atributo privado mat
    int aux[][]=new int[n][n];
    for(int posi=0;posi<(n*n);posi++)
        aux[posi/(int)n][posi%n]=mat[posi/(int)n][posi%n];
    return(aux);
}

private int[][] setColInMat(int col,int contenido[],int matriz[][]){
    /*Llena la columna col de la matriz indicada por parametro con el vector
     pasado por parametro.*/
    for(int fil=0;fil<n;fil++)
        matriz[fil][col]=contenido[fil];
    return(matriz);
}
```

El método getCopyMat() devuelve un int[][] con una copia exacta del atributo mat de dicho objeto.
El método setColInMat(int col,int contenedor[],int matriz[][]) reemplaza la columna indicada por parámetro con el vector contenedor[].

Comparador de Objetos Matriz:

```
public static boolean matCmp(Matriz mat1,Matriz mat2,int dim){//Compara dos objetos Matriz
    int posi=0;
    for(posi=0;( (posi<((dim*dim)-1) ) && (mat1.getElementbyPos(posi/(int)dim,posi%dim)==mat2.ge
    return(posi==((dim*dim)-1));
}
```

Recorre la matriz hasta encontrar un elemento distinto o llegar al final de las matrices.
Las dimensiones de la matriz deben ser iguales en ambos objetos.

Contador de Instancias:

La declaración de un atributo estatico llamado instancias y su incremento en los constructores permite mediante el método estatico getInstances() , la cantidad de instancias de la clase se crearon durante la ejecución del programa.

Matriz Inversa:

```
public /*float [][]*/void getInversa(){//no devuelvo nada porque la matriz inv. es de punto flot
    //A^(-1)=T(ADJ(A))/Det(A);
    System.out.println("mat^(-1)\n");
    int adj_A[][]=new int[n][n];//float adj_A...
    int det_mat=getDeterminante();
    Matriz adjs;
    int posi=0;
    if(det_mat!=0){
        for(posi=0;posi<(n*n);posi++){//saco los Adj(this[i][j])
            adjs=new Matriz(getAdjMatbyPos((posi/(int)n),(posi%n)),(n-1));
            adj_A[posi/(int)n][posi%n]=adjs.getDeterminante();
        }
        adj_A=trasponerMat(adj_A,n);//saco T(ADJ(A))
        for(posi=0;posi<(n*n);posi++){
            //adj_A[posi/(int)n][posi%n]*=(1/det_mat); punto flotante
            //System.out.print("( " + adj_A[posi/(int)n][posi%n]*(1/(float)det_mat) + " )");
            System.out.print("( " + adj_A[posi/(int)n][posi%n] + "/" + det_mat + " )");
            if(posi%n==(n-1))System.out.println();
        }
    }else
        System.out.println("Determinante Nulo, Matriz No Inversible.");
    //return(adj_A);
}
```

Este método imprime la matriz inversa de mat. Es calculada utilizando la matriz adjunta o de cofactores, formada por los elementos Trasp(ADJ(i,j)) y dividiéndola por el escalar resultado del determinante de mat.Si el determinante es igual a 0. La matriz no es reversible o invertible.

Idempotente:

```
public boolean isIdempotente(){//this = referencia al objeto actual
    Matriz mat2=new Matriz(productoMat(getCopyMat(),getCopyMat(),n),n);
    return matCmp(this,mat2,n);//multiplico la matriz por si misma y utilizo el metodo comparado
}

private int[][] productoMat(int mat1[][],int mat2[][],int dim){
    int retorno[][]=new int[dim][dim];//mat3
    for(int posi=0;posi<(dim*dim);posi++)//indices para mat3
        for(int posprod=0;posprod<dim;posprod++)//recorro filas y columnas en mat2 y mat1 respec
            retorno[posi/(int)n][posi%n]+=mat1[posi/(int)n][posprod]*mat2[posprod][posi%n];
    return(retorno);
```

Este método calcula el producto de una matriz por si misma y utiliza el método comparador para determinar si es Idempotente o no, si lo es debe cumplir la siguiente condición A.A=A.

Conclusion:

El trabajo practico me costo bastante esfuerzo y tiempo pero finalmente me gusto haberlo logrado y sobretodo que me hizo pensar bastante y me saco de las “recetas” de cómo venia programando hasta ahora.