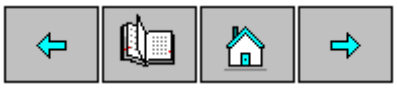
La clase *Matriz* y la clase *Vector*



Las matrices cuadradas

En esta página, definiremos la clase *Vector* y la clase *Matriz* cuadrada, las operaciones entre matrices, suma y producto de dos matrices, el producto de una matriz por un vector, el producto de una matriz por un escalar, la traza de una matriz y la matriz traspuesta de una matriz dada.

Vectores y matrices

Un vector es un array unidimensional de números. Se define la clase *Vector* con dos miembros dato, el número de datos que guarda y el array unidimensional que guarda dichos datos.

```
public class Vector {
    public int n;
    double[] x;
```

La clase *Vector* y la clase *Matriz* están en el mismo paquete. El miembro dato *x* de la clase *Vector* tiene el control de acceso por defecto, es decir, público dentro del mismo paquete pero privados fuera del paquete. Más abajo, en esta página al definir las funciones miembro de la clase *Matriz* que realizan operaciones entre matrices y vectores veremos que, los objetos de la clase *Vector* necesitan acceder a su miembro dato *x*.

Una matriz es un array bidimensional de números. En general, decimos que una matriz tiene una dimensión *m* x *n*, cuando los números están dispuestos en *m* filas y *n* columnas.

Se denominan matrices cuadradas a aquellas que tienen el mismo número de filas que de columnas. Estas matrices tienen especial importancia y serán las que tratemos en estas páginas.

```
public class Matriz{
    public int n;
    private double[][] x;
```

La clase *Matriz* tiene dos miembros dato, la dimensión de la matriz *n*, y un array bidimensional *x*, que crearemos e inicializaremos en los constructores.

Los constructores

Vamos a definir dos constructores en la clase *Vector*, al primero se le pasa el número de elementos que va a guardar e inicializan a cero todos sus elementos.

```
    public Vector(int n) {
        this.n=n;
        x=new double[n];
        for(int i=0; i<n; i++){
            x[i]=0.0;
        }
    }
```

Para crear un vector *v* de dimensión tres se escribe

```
        Vector v=new Vector(3);
```

Al segundo constructor, se le pasa el array unidiemensional, e inicializa el miembro dato *x* con los valores que guardan los elementos de dicho array en una única y simple operación de asignación

```
    public Vector(double[] x) {
        this.x=x;
        n=x.length;
    }
```

Para crear un vector *v* que guarde los datos del array *v1* se escribe

```
        double[] v1={1, 2, 3};
        Vector v=new Vector(v1);
```

Para la clase *Matriz* necesitamos definir dos constructores, al primero se le pasa la dimensión *n* de la matriz cuadrada, creando un array bidimensional de *n* filas y *n* columnas, e inicializa todos sus elementos a cero.

```
    public Matriz(int n) {
        this.n=n;
        x=new double[n][n];
        for(int i=0; i<n; i++){
```

```
            for(int j=0; j<n; j++){
                x[i][j]=0.0;
            }
        }
    }
```

Para crear una matriz *a* de dimensión tres cuyos elementos son todos ceros, se escribe.

```
        Matriz a=new Matriz(3);
```

Al segundo constructor, se le pasa un array bidimensional, e inicializa el miembro dato *x* con los valores que guardan los elementos de dicho array en una única y simple operación de asignación.

```
    public Matriz(double[][] x) {
        this.x=x;
        n=x.length;
   }
```

Para crear la matriz *a*

$$\begin{pmatrix} 1 & 2 & 3 \\ -2 & -4 & -5 \\ 3 & 5 & 6 \end{pmatrix}$$

se crea un array bidimensional *a1*, y se le pasa al constructor de la clase *Matriz*

```
        double[][] a1={{1, 2, 3},{-2, -4, -5},{3, 5, 6}};
        Matriz a=new Matriz(a1);
```

```
public class Vector {
    public int n;      //dimensión
    double[] x;
    public Vector(int n) {
        this.n=n;
        x=new double[n];
        for(int i=0; i<n; i++){
            x[i]=0.0;
        }
    }
    public Vector(double[] x) {
        this.x=x;
        n=x.length;
    }
//otras funciones miembro
}

                        ****************************************

public class Matriz{
    public int n;      //dimensión
    private double[][] x;
    public Matriz(int n) {
```

```
        this.n=n;
        x=new double[n][n];
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                x[i][j]=0.0;
            }
        }
    }
    public Matriz(double[][] x) {
        this.x=x;
        n=x.length;
   }
//otras funciones miembro
}
```

Mostrar un vector y una matriz

Para mostrar un vector redefinimos la función *toString* de la clase base *Object*, de la cual deriva implícitamente *Vector*. Separamos sus elementos mediante el carácter tabulador '*t*'.

```
    public String toString(){
        String texto=" ";
        for(int i=0; i<n; i++){
            texto+="\t "+(double)Math.round(1000*x[i])/1000;
        }
        texto+="\n";
        return texto;
    }
```

Vamos añadiendo al string *texto*, los elementos del vector y el carácter separador entre elementos, limitamos el número de decimales a tres mediante la función *Math.round*. Para concluir la fila y pasar a la siguiente en la pantalla de texto, añadimos un carácter retorno de carro '*n*'. Mediante la operación + definida en la clase *String* podemos concatenar fácilmente los distintos elementos y crear la representación textual del vector que devuelve la función *toString*

Para mostrar *el* vector *v* en la pantalla de texto, basta escribir la sentencia

```
        Vector v=new Vector(v1);
        System.out.println(v);
```

Mostrar una matriz en la pantalla de texto es difícil, ya que Java no dispone de una función que sitúe el cursor de texto en una posición de la pantalla, como lo hace la función *gotoxy* disponible en los lenguajes C/C++. La única alternativa que nos queda es mostrar los elementos de una fila unos a continuación de los otros separados por un tabulador, después otra fila y así hasta mostrar todos los elementos de la matriz.

Para mostrar los elementos de la matriz, redefinimos la función *toString* de la clase base *Object*, de la cual deriva implícitamente *Matriz.* Separamos los elementos de una fila mediante el carácter tabulador '\t', y limitamos el número de decimales a tres mediante la función *Math.round*. Cuando se acaba una fila se inserta un retorno de carro '\n' y se continua con la siguiente fila, y así sucesivamente.

```
            for(int j=0; j<n; j++){
                texto+="\t "+(double)Math.round(1000*x[i][j])/1000;
            }
            texto+="\n";
```

Vamos añadiendo al string *texto*, los elementos de la matriz y los caracteres separadores entre elementos y entre filas de elementos.

```
    public String toString(){
        String texto="\n";
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                texto+="\t "+(double)Math.round(1000*x[i][j])/1000;
            }
            texto+="\n";
        }
        texto+="\n";
        return texto;
  }
```

Para mostrar una matriz *a* en la pantalla de texto basta escribir la sentencia

```
        Matriz a=new Matriz(a1);
        System.out.println(a);
```

```
public class Vector {
    public int n;      //dimensión
    double[] x;
//...
    public String toString(){
        String texto=" ";
        for(int i=0; i<n; i++){
            texto+="\t "+(double)Math.round(1000*x[i])/1000;
        }
        texto+="\n";
        return texto;
  }
}

                              ****************************

public class Matriz{
    public int n;      //dimensión
    private double[][] x;
//...
    public String toString(){
        String texto="\n";
```

```
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                texto+="\t "+(double)Math.round(1000*x[i][j])/1000;
            }
            texto+="\n";
        }
        texto+="\n";
        return texto;
  }
}
```

Copia de una matriz dada

Cuando calculamos la matriz inversa de una dada, pasamos una matriz en el único parámetro de la función estática denominada *inversa*. En el cuerpo de dicha función se realizan operaciones con los elementos de dicha matriz. Dado que en Java se pasan por valor las referencias a objetos, la matriz original resulta modificada en el curso de la llamada a la función *inversa*.

Si queremos mantener la matriz original, hacemos una copia de dicha matriz en el cuerpo de la función y realizamos las operaciones con la matriz copia dejando la original sin modificar.

Para realizar una copia hemos de implementar el interface *Cloneable*, y redefinir la función miembro *clone* de la clase base *Object*, de la cual derivan todas las clases en Java. El primer paso es la llamada a la función *clone* de la clase base *Object*.

```
public class Matriz implements Cloneable{
    public int n;      //dimensión
    private double[][] x;
//.........

public Object clone(){
        Matriz obj=null;
        try{
//llama a clone de la clase base Object
            obj=(Matriz)super.clone();
        }catch(CloneNotSupportedException ex){
            System.out.println(" no se puede duplicar");
        }
//copia la matriz bidimensional
        obj.x=(double[][])obj.x.clone();
        for(int i=0; i<obj.x.length; i++){
            obj.x[i]=(double[])obj.x[i].clone();
        }
        return obj;
    }
}
```

Para obtener una copia *a* de una matriz *d* se escribe.

```
Matriz a=(Matriz)d.clone();
```

La promoción (casting) es necesaria ya que *clone* devuelve una referencia a un objeto de la clase base *Object*.

Traza de una matriz

Se denomina traza de una matriz cuadrada a la suma de los elementos de su diagonal principal.

```
public double traza(){
    double tr=0.0;
    for(int i=0; i<n; i++){
        tr+=x[i][i];
    }
    return tr;
}
```

Para obtener la traza de la matriz *a* de la sección anterior se escribe

```
double traza=a.traza();
```

Operaciones con matrices y vectores

Suma de dos matrices cuadradas

Cuando se suman dos matrices de las mismas dimensiones

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

Se obtiene otra matriz *c* en la que sus elementos c_{ij} son las suma de los correspondientes elementos de las matrices *a* y *b*, es decir

$c_{ij}=a_{ij}+b_{ij}$

Para sumar dos matrices, se define una función miembro estática denominada *suma*. Dentro de la función, se crea una matriz temporal *resultado*, con la misma dimensión de las matrices que intervienen en la operación, y se guardan en sus elementos el resultado de la suma de las

matrices *a* y *b*. Finalmente, la función *suma* devuelve la matriz *resultado*.

```
public static Matriz suma(Matriz a, Matriz b){
    Matriz resultado=new Matriz(a.n);
    for(int i=0; i<a.n; i++){
        for(int j=0; j<a.n; j++){
            resultado.x[i][j]=a.x[i][j]+b.x[i][j];
        }
    }
    return resultado;
}
```

Veamos ahora como se llama a la función que suma dos matrices.

```
double[][] a1={{1, 2, 3},{4,5,6},{7,8,9}};
Matriz a=new Matriz(a1);
double[][] b1={{1, 0, -1},{2,1,3},{-1, 0, 2}};
Matriz b=new Matriz(b1);
Matriz re=Matriz.suma(a, b);
System.out.println("matriz "+re);
```

Producto de dos matrices

La regla para multiplicar dos matrices es bastante más complicada que para sumar dos matrices de las mismas dimensiones. En general, se pueden multiplicar dos matrices de dimensiones *m x n* y *n x q*, dando como resultado una matriz de dimensiones *m x q*. En este apartado nos circunscribiremos exclusivamente a matrices cuadradas de dimensión *n*.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} x \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

Los elementos c_{ij} se obtienen multiplicando los elementos a_{ik} de la fila *i* por los elementos a_{kj} de la columna *j*, y sumando los resultados.

$$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + \ldots a_{in-1}b_{n-1j} \qquad (i = 0 \ldots n-1, j = 0 \ldots n-1)$$

La codificación se realiza empleando un tripe bucle **for**, guardando en los elementos de la la matriz local *resultado* la suma de los productos de la fórmula anterior.

```
public static Matriz producto(Matriz a, Matriz b){
   Matriz resultado=new Matriz(a.n);
   for(int i=0; i<a.n; i++){
       for(int j=0; j<a.n; j++){
           for(int k=0; k<a.n; k++){
               resultado.x[i][j]+=a.x[i][k]*b.x[k][j];
           }
```

```
            }
        }
        return resultado;
    }
```

Otras variantes de la operación producto son:

El producto de un escalar (número real) por una matriz que da como resultado otra matriz cuyos elementos están todos multiplicados por dicho escalar. Se define también la operación conmutativa

```
public static Matriz producto(double d, Matriz a){
    Matriz resultado=new Matriz(a.n);
    for(int i=0; i<a.n; i++){
        for(int j=0; j<a.n; j++){
            resultado.x[i][j]=a.x[i][j]*d;
        }
    }
    return resultado;
}
```

Al multiplicar una matriz cuadrada de dimensión *n*, por un vector columna de la misma dimensión obtenemos otro vector columna. Cada elemento del vector resultante se obtiene multiplicando los elementos de una fila de la matriz por los correspondientes elementos del vector columna y se suman los resultados. La codificación de esta función producto es la siguiente:

```
    public static Vector producto(Matriz a, Vector v){
        int n=v.n;
        Vector b=new Vector(n);
        for(int i=0; i<n; i++){
            for(int k=0; k<n; k++){
                b.x[i]+=a.x[i][k]*v.x[k];
            }
        }
        return b;
    }
```

Al multiplicar un vector fila por una matriz cuadrada de la misma dimensión obtenemos otro vector fila. El código es semejante al de la función *producto* definida previamente.

```
public static Vector producto(Vector v, Matriz a){
    int n=v.n;
    Vector b=new Vector(n);
    for(int j=0; j<n; j++){
        for(int k=0; k<n; k++){
            b.x[j]+=v.x[k]*a.x[k][j];
        }
    }
    return b;
}
```

Matriz traspuesta

Una matriz traspuesta de otra matriz es aquella que tiene los mismos elementos pero dispuestos en forma distinta. Las columnas de la matriz original se transforman en filas de la matriz traspuesta. La definición de la función estática *traspuesta* no reviste dificultad alguna

```
public static Matriz traspuesta(Matriz a){
    int n=a.n;
    Matriz resultado=new Matriz(a.n);
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            resultado.x[i][j]=a.x[j][i];
        }
    }
    return resultado;
}
```

Para hallar la matriz traspuesta de la matriz *a* se escribe

```
double[][] a1={{1, 2, 3},{4,5,6},{7,8,9}};
Matriz a=new Matriz(a1);
Matriz tras=Matriz.traspuesta(a);
System.out.println("matriz traspuesta"+tras);
```