

#### Practica 4

Esta practica consta de 3 ejercicios. Aquí escribiré los códigos fuente(en azul) y comentare los resultados obtenidos al ejecutarlos(en negro). Los archivos \*.cpp ,los archivos \*.h y los archivos \*.exe están en el disquete adjunto.

##### Ejercicio 1;

En este primer ejercicio escribiremos el código de xp4.cpp para realizar diferentes operaciones con matrices y vectores. Después comprobaremos su funcionamiento.

xp4.cpp

```
//Practica 4(ejercicio 1)xp4.cpp
//Programa realizado por Marc Belda
#include <iostream>
#include <iomanip>
#include <string>
#include <error.h>
#include <matutil.h>
#include <vector.h>
#include <matriz.h>
using namespace std;

int main ()
{
//En primer lugar construiremos las matrices B,C,y D
//Matriz B
int bfilas,bcolumn;
cout <<"En primer lugar construiremos las matrices B,C y D eligiendo el numero de
filas"<<endl;
cout <<"y de columnas."<<"\n";
cout <<"Introduzca el numero de filas de B,bfilas."<<"\n";
cin >> bfilas;
cout <<"Introduzca el numero de columnas de B,bcolumn."<<"\n";
cin >> bcolumn;
RMatriz B(bfilas,bcolumn);
cout <<"La matriz B consta de "<<B.filas()<<" filas y de "<<B.columnas()<<"
columnas."<<"\n";
for (int i=1;i<=bfilas;i++)
for (int j=1;j<=bcolumn;j++)
{
cout << "B("<<i<<","<<j<<")=";
cin >> B(i,j);
}
cout <<"Hemos construido la matriz B. Matriz B = "<< B <<"\n";
//Matriz C
int cfilas,ccolumn;
cout <<"Sigamos con la matriz C"<<"\n";
```

```

cout << "Introduzca el numero de filas de C, cfilas." << "\n";
cin >> cfilas;
cout << "Introduzca el numero de columnas de C, ccolum." << "\n";
cin >> ccolum;
RMatriz C(cfilas, ccolum);
cout << "La matriz C consta de " << C.filas() << " filas y de " << C.columnas() << "
columnas." << "\n";
for (int i=1; i<=cfilas; i++)
for (int j=1; j<=ccolum; j++)
{
cout << "C(" << i << ", " << j << ")=";
cin >> C(i,j);
}
cout << "Hemos construido la matriz C. Matriz C = " << C << "\n";
//Matriz D
int dfilas, dcolum;
cout << "Sigamos con la matriz D" << "\n";
cout << "Introduzca el numero de filas de D, dfilas." << "\n";
cin >> dfilas;
cout << "Introduzca el numero de columnas de D, dcolum." << "\n";
cin >> dcolum;
RMatriz D(dfilas, dcolum);
cout << "La matriz D consta de " << D.filas() << " filas y de " << D.columnas() << "
columnas." << "\n";
for (int i=1; i<=dfilas; i++)
for (int j=1; j<=dcolum; j++)
{
cout << "D(" << i << ", " << j << ")=";
cin >> D(i,j);
}
cout << "Hemos construido la matriz D. Matriz D = " << D << "\n";
//Operaciones con matrices
cout << "Calcularemos la matriz A que resulta de operar B,C,D de forma que
A=B*B(traspuesta)+C*D" << "\n";
cout << "Primero calcularemos de la traspuesta de B" << endl;
cout << "La traspuesta de B es (que llamaremos Y), Y= " << traspuesta(B) << "\n";
RMatriz Y = traspuesta(B);
cout << "El resultado de la operacion es la matriz A, matriz A= " << (B*Y)+(C*D) << endl;
RMatriz A = (B*Y)+(C*D);
cout << "Ahora sumaremos las matrices A y B (resultado de la operacion A+=B)" << "\n";
cout << "A+B= " << A+B << endl;
cout << "Ahora multiplicaremos la matriz C por un escalar lambda (resultado de la
operacion C*=lambda" << endl;
double lamda;
double mu;
cout << "Para eso elegiremos el valor de ese numero real lambda. Introduzca el valor de
lambda, lambda= " << "\n";

```

```

cin >> lamda;
cout << "El resultado de multiplicar C*lamda es " << C*lamda << endl;
cout << "Ahora calcularemos el resultado de la operacion mu*(A+B)-(B+A)*mu donde mu
es" << "\n";
cout << "otro escalar real que introduciremos a continuacion, mu= " << "\n";
cin >> mu;
cout << "El resultado de la operacion mu*(A+B)-(B+A)*mu es " << mu*(A+B)-
(B+A)*mu << "\n";
//Ahora construiremos los vectores v1 y v2, donde adim es la dimension de v1 y bdim es la
dimension de v2
int adim;
cout << "Introduzca la dimension de v1, adim= " << "\n";
cin >> adim;
RVector v1(adim);
cout << "Hemos creado un vector de dimension " << v1.dimension() << endl;
for (int i=1; i<=adim; i++)
{
cout << "v1(" << i << ")=";
cin >> v1(i);
}
cout << "El vector v1 es, v1= " << v1;
int bdim;
cout << "Introduzca la dimension de v2, bdim= " << "\n";
cin >> bdim;
RVector v2(bdim);
cout << "Hemos creado un vector de dimension " << v2.dimension() << endl;
for (int j=1; j<=bdim; j++)
{
cout << "v2(" << j << ")=";
cin >> v2(j);
}
cout << "El vector v2 es, v2= " << v2;
//Operaciones con vectores
cout << "Calcularemos el valor de la operacion lamda*(v1+v2)-(v2+v1)*lamda, siendo los
valores" << "\n";
cout << "de lamda y mu los elegidos en el apartado anterior. El resultado es
" << lamda*(v1+v2)-(v2+v1)*lamda << "\n";

return EXITO;
}

```

El ejercicio propone comprobar el funcionamiento del programa con las matrices  $B$ ,  $C$  y  $D$  y los vectores  $V_1$  y  $V_2$ .

a) Cálculo de la matriz  $A$ , siendo la matriz  $A$ ;

$$A = B * B^T + C * D$$

Donde  $B^T$  es la matriz  $B$  traspuesta. El programa funciona correctamente.

b) Sumamos la matriz  $A$  y la matriz  $B$ . Resultado de la operación  $A + B$ . El programa funciona correctamente.

c) Multiplicamos la matriz  $C$  por un escalar  $I = 1.5$ . Resultado de la operación  $C * I$ . El programa funciona correctamente.

d) Calculamos el resultado de operar las matrices  $A$  y  $B$  y los escalares  $I$  y  $m$  de la siguiente manera;

$$m * (A + B) - (B - A) * I$$

El programa funciona correctamente.

e) Por último calcularemos el resultado de operar los vectores  $V_1$  y  $V_2$  con los escalares  $I$  y  $m$  de la siguiente manera;

$$I * (V_1 + V_2) - (V_2 + V_1) * I$$

El programa funciona correctamente.

\*nota; no he reproducido el resultado ni el valor de las matrices del enunciado porque no he encontrado una presentación adecuada. Si he comprobado el correcto funcionamiento del programa. El programa está incluido en la práctica. Así que no tiene más que hacerlo correr para comprobar su validez.

## Ejercicio 2;

Este ejercicio nos pide ampliar la clase `RVector` para incluir nuevas funciones. Así que lo único que tengo que hacer es declarar esas nuevas funciones y definir las correctamente. Las declaraciones se encuentran en el fichero `vector.h` y las definiciones en `vector.cpp`. Modificando el código de `xvector.cpp` podemos comprobar su funcionamiento.

\*nota; solo incluiré, en cada caso el código que añado en cada uno de los ficheros y para cada una de las nuevas funciones.

a) La función `norma` para calcular la norma euclídea del vector;

Declaración de la función `norma(vector.h)`;

```
double norma(const RVector& v1);
```

Definición de la función `norma(vector.cpp)`;

```
double norma(const RVector& v1)
{
    double z;
    double norma=0;
    for (int i=1; i<=v1.dimension();i++)
    {
        norma+=v1(i)*v1(i);
    }
}
```

```

    }
    z=sqrt(norma);
    return z;
}

```

Comprobamos su funcionamiento(xvector.cpp);

```

double Norma;
Norma=norma(v1);
double z=Norma;
cout<<"La norma del vectro v2 es "<< z <<"\n";

```

Esta función ha sido declarada externa a la clase Rvector.

b) Las funciones máximo y mínimo, que devuelven el índice de elemento máximo y mínimo respectivamente.

Funciones máximo y mínimo;

Declaración de la función máximo y la función mínimo(vector.h);

```

int maximo(const RVector& v1,double& max);
int minimo(const RVector& v1,double& min);

```

Definición de la función máximo y la función mínimo(vector.cpp);

```

int maximo(const RVector& v1,double& max)
{
    int imax=1;
    max=v1(1);
    for (int i=2;i<=v1.dimension();i++)
    {
        if (v1(i)>max)
        {
            max=v1(i);
            imax=i;
        }
    }
    return imax;
}
int minimo(const RVector& v1,double& min)
{
    int imin=1;
    min=v1(1);
    for (int i=2;i<=v1.dimension();i++)
    {
        if (v1(i)<min)
        {
            min=v1(i);

```

```

        imin=i;
    }
}
return imin;
}

```

Comprobamos su funcionamiento(xvector.cpp);

```

double max;
cout <<"El indice del elemento maximo del vector v1 es "<< maximo(v1,max)<<"\n";
double min;
cout <<"El indice del elemento minimo del vector v1 es "<< minimo(v1,min)<<"\n";

```

Las funciones han sido declaradas externas a la clase.

c)La función intercambia, que intercambia el elemento *i* y el elemento *j* del vector.

Declaración de la función intercambia(vector.h);

```
RVector& intercambia (int i,int j);
```

Definición de la función intercambia(vector.cpp);

```

RVector& RVector::intercambia(int i,int j)
{
    if (i < 1 || i > pDim)
        error("Error,indice fuera de rango",CERANGO);
    double temp=0;
    temp=(*this)(i);
    (*this)(i)=(*this)(j);
    (*this)(j)=temp;
    return *this;
}

```

Comprobamos su funcionamiento(xvector.cpp);

```

cout <<"Ahora intercambiamos dos elemntos cualquiera del vectro v1 entre si."<<"\n";
cout <<"Elige los dos indices que quieres intercambiar,i y j. i= "<<endl;
int i;
cin >> i;
cout <<"Elige ahora en indice j, j= "<<endl;
int j;
cin >>j;
v1.intercambia(i,j);
cout <<"El nuevo vector con los elementos i y j intercambiado es "<<v1<<endl;

```

La función ha sido declarada interna a la clase.

La función intercambia es interna porque modifica los miembros de la clase. Un vector  $V$  es un elemento interno de la clase Rvector. El resultado de intercambiar dos elementos de un vector es otro vector que a su vez es un elemento de la clase Rvector, esa es la razón de declarar interna la función intercambia.

En el caso de las funciones norma, máximo y mínimo ocurre justo lo contrario, el resultado de aplicar cualquiera de estas tres funciones da como resultado un escalar, no un vector. Un escalar es un elemento externo a la clase Rvector, por tanto también lo son las funciones norma, máximo y mínimo.

El programa xvector.exe funciona correctamente.

Ejercicio 3;

Este ejercicio nos pide ampliar la clase RMatriz para incluir nuevas funciones. Así que lo único que tengo que hacer es declarar esas nuevas funciones y definirlas correctamente. Las declaraciones se encuentran en el fichero matriz.h y las definiciones en matriz.cpp. Modificando el código de xmatriz.cpp podemos comprobar su funcionamiento.

a) La función Traza, que devuelve la traza de la matriz;

Declaración de la función traza(matriz.h);

```
int Traza(const RMatriz& m);
```

Definición de la función Traza(matriz.cpp);

```
int Traza(const RMatriz& m)
{
    for (int i=1;i<=m.filas;i++)
        for (int j=1;j<=m.columnas;j++)
            cout <<"m("<<i<<","<<j<<")= "<<m(i,j)<<endl;
    return 0;
}
```

Comprobamos su funcionamiento(xmatriz.cpp);

```
double traza;
traza=Traza(const RMatriz& m);
cout <<"La traza de la matriz m es, mtraza "<<traza<<endl;
```

La función ha sido declarada externa a la clase.

b) La función proyectaFila, que crea un vector y copia en este una fila escogida(en el argumento a la función) de la matriz. Y la función proyectaColumna, que crea un vector y copia en este una columna escogida(en el argumento a la función) de la matriz.

Declaración de las funciones proyectaFila y proyectaColumna(matriz.h);

```
RVector proyectaFila(const RMatriz& m,int fila);
RVector proyectaColumna(const RMatriz& m,int col);
```

Definición de la funciones proyectaFila y proyectaColumna(matriz.cpp);

```
RVector proyectaFila(const RMatriz& m,int fila)
{
    int fdim=m.columnas();
    RVector vf(fdim);
    for (int i=1;i<=fdim;i++)
        vf(i)=m(fila,i);
    return vf;
}
RVector proyectaColumna(const RMatriz& m,int col)
{
    int cdim=m.filas();
    RVector vc(cdim);
    for (int i=1;i<=cdim;i++)
        vc(i)=m(i,col);
    return vc;
}
```

Comprobamos su funcionamiento(xmatriz.cpp);

```
int fila;
cout <<"Elija una fila cualquiera de la matriz m para proyectarla, fila= "<<endl;
cin >>fila;
RVector v=proyectaFila (m,fila);
cout <<"La proyeccion de la fila es, fila= "<<v<<endl;
int col;
cout <<"Elija una columna cualquiera de la matriz m para proyectarla, columna= "<<endl;
cin >>col;
RVector w=proyectaColumna (m,col);
cout <<"La proyeccion de la columna es, columna= "<<w<<endl;
```

Las funciones han sido declaradas externas a la clase.

c)La función intercambiaFila, que intercambia la fila i y la fila j de la matriz. La función intercambiaColumna que intercambia la columna i y la columna j de la matriz.

Declaración de la funciones intercambiaFila y intercambiaColumna(matriz.h);

```
RMatriz& intercambiaFilas(int i,int j);
RMatriz& intercambiaColumnas(int i,int j);
```

Definición de las funciones intercambiaFila y intercambiaColumna(matriz.cpp);

```
RMatriz& RMatriz::intercambiaFilas(int i,int j)
{
    RVector v1(i);
    for (int k=1;k<=pCols;k++)
```



```

    {
    v1(i)=(*this)(i,k);
    (*this)(i,k)=(*this)(j,k);
    (*this)(j,k)=v1(i);
    }
    return *this;
}
RMatriz& RMatriz::intercambiaColumnas(int i,int j)
{
RVector v1(i);
for (int k=1;k<=pFilas;k++)
{
v1(i)=(*this)(k,i);
(*this)(k,i)=(*this)(k,j);
(*this)(k,j)=v1(i);
}
return *this;
}

```

Comprobamos su funcionamiento(xmatriz.cpp);

```

int l,s;
cout <<"Elige las fila de la matriz m que quieras intercambiar, fila1="<<endl;
cin >>l;
cout <<"Elige la otra fila, fila2= "<<endl;
cin >>s;
m1.intercambiaFilas (l,s);
cout <<"La matriz m es ahora, nueva m= "<<m1<<endl;
int p,q;
cout <<"Elige las columnas de la matriz m que quieras intercambiar, columna1="<<endl;
cin >>p;
cout <<"Elige la otra columna, columna2= "<<endl;
cin >>q;
m2.intercambiaColumnas (p,q);
cout <<"La matriz m es ahora, nueva m= "<<m2<<endl;

```

Las funciones han sido declaradas internas a la clase.

Por las mismas razones que en el ejercicio anterior, hemos declarado las funciones intercambiaFila y intercambiaColumna internas a la clase porque el resultado de intercambiar una fila o una columna de una matriz es otra matriz, por tanto el resultado sigue perteneciendo a la clase RMatriz.

Recíprocamente, el resultado de calcular la traza o proyectar las filas o las columnas de una matriz es un vector, o lo que es lo mismo, un elemento externo a la clase RMatriz. Por tanto hemos declarado estas funciones como externas a la clase RMatriz.

No he conseguido hacer funcionar el programa xmatriz.exe .El problema es que no he sabido transformar el resultado de calcular la traza de una matriz en un vector, el código que es escrito posee errores y todos derivan(a grandes rasgos) de esa parte del programa.