

Prácticas MNC.

Práctica 2. Computación Numérica.

Actividad 1.

Implementa una función que multiplique dos matrices utilizando la multiplicación propia de Octave y que mida el tiempo de ejecución

- Implementa una función que multiplique dos matrices utilizando el algoritmo tradicional y que mida el tiempo de ejecución
- Crea dos matrices aleatorias, multiplícalas usando ambas funciones y comprueba que el resultado obtenido es el mismo

```
Matrices multiplicando con matlab =
```

```
0.4035    0.2667    0.4467    0.3408  
0.8663    1.4987    1.5458    1.4350  
0.6137    1.4148    0.7641    1.0705  
0.6292    1.2492    0.7291    0.9499
```

```
Tiempo con matlab =
```

```
1.3000e-04
```

```
Matrices multiplicando sin matlab =
```

```
0.4035    0.2667    0.4467    0.3408  
0.8663    1.4987    1.5458    1.4350  
0.6137    1.4148    0.7641    1.0705  
0.6292    1.2492    0.7291    0.9499
```

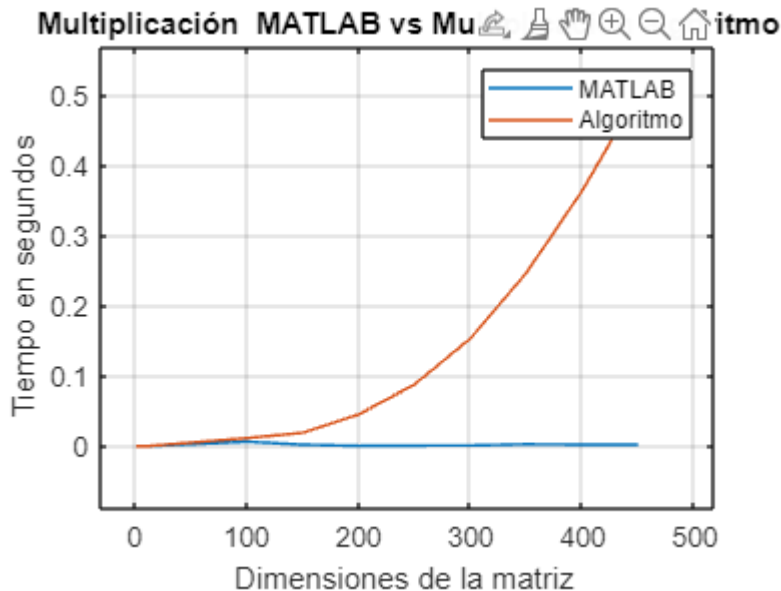
```
Tiempo sin matlab =
```

```
0.0013
```

```
>>
```

- Dibuja una gráfica en la que se compare el tiempo de ejecución de ambas funciones para matrices de tamaño creciente; el tiempo de ejecución de cada tamaño de matriz debe calcularse como el tiempo medio de diez multiplicaciones para ese tamaño en concreto

A continuación podemos ver en la gráfica que los tiempos en la función de matlab a medida que aumenta la dimensión de la matriz apenas aumenta mientras que el tiempo de la multiplicación con el algoritmo de las diapositivas aumenta de manera exponencial.

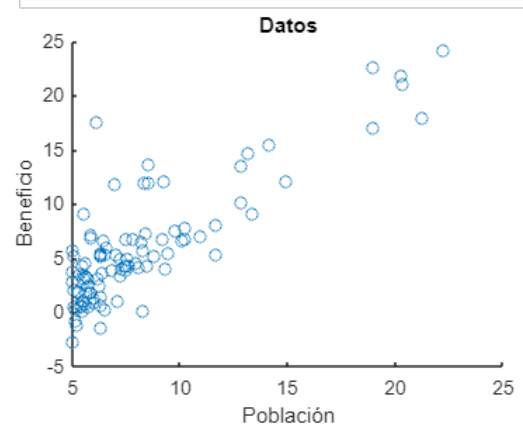


Actividad 2.

- Carga el fichero de datos proporcionado load data
 - La columna 1 muestra la población de una serie de ciudades (decenas de miles de personas)
 - La columna 2 muestra los beneficios de una cadena de tiendas (decenas de miles de euros)
- Crea una figura en la que se muestre cada punto del conjunto de datos
 - En el eje x se mostrará la población
 - En el eje y se mostrarán los beneficios (puede ser negativo)
- Añade un título, etiquetas a los ejes y todos aquellos elementos que creas conveniente para hacer la figura más legible

A continuación vemos la gráfica y el resultado de extraer los datos de población y beneficio. Usamos una función de matlab para pintar en la grafica la nube de puntos

```
1 clc;
2 clear;
3
4 %%Carga de datos
5 load('data.mat', '-ascii');
6
7 %%organizamos las datos extrayendo las columnas que queremos
8 pob = data(:, 1);
9 ben = data(:, 2);
10
11 scatter(pob, ben);
12 |
13 xlabel("Población");
14 ylabel("Beneficio");
15 title("Datos");
```



Actividad 3.

- Implementa el cálculo de la función de costo de forma iterativa
 - Comprueba que el resultado para $\theta_0=0$ y $\theta_1=0$ es 32'07
 - Comprueba que el resultado para $\theta_0=-1$ y $\theta_1=2$ es 54'24
- Obtén el tiempo de ejecución de este código ejecutándolo 5 veces para ambos casos y calculando la media

Tras realizar los cálculos de la función de costo los resultados son:

Command Window

Resultado y tiempos para $\text{Sigma0} = 0$ $\text{Sigma1} = 0$

32.0727

0.0011

Resultado y tiempos para $\text{Sigma0} = -1$ $\text{Sigma1} = 2$

54.2425

2.3440e-04

>>

Actividad 4.

Implementa el cálculo de la función de costo de forma matricial

- Comprueba que el resultado para $\theta_0=0$ y $\theta_1=0$ es 32'07
- Comprueba que el resultado para $\theta_0=-1$ y $\theta_1=2$ es 54'24

- Obtén el tiempo de ejecución de este código ejecutándose 5 veces para ambos casos y calculando la media

Tras realizar con matlab las operaciones del algoritmo para el cálculo de la función de costo con matrices el resultado comprobamos que es correcto.

Command Window

```
Resultado y tiempos para Sigma0 = 0 Sigma1 = 0  
32.0727
```

```
0.0018
```

```
Resultado y tiempos para Sigma0 = -1 Sigma1 = 2  
54.2425
```

```
2.4460e-04
```

```
>>
```

Actividad 5.

Implementa el algoritmo de descenso según el gradiente de forma iterativa

– Aplica 1.500 iteraciones con un valor $\alpha = 0.01$

– Comprueba que el resultado es $\theta_0 = -3.63$ y $\theta_1 = 1.17$ para iteraciones desde 1 hasta 1500

- Obtén el tiempo de ejecución de este código ejecutándolo 10 veces y calculando la media

Tras Calcular con matlab el algoritmo de descenso según el gradiente.

El resultado que nos dá despues de aplicar el algoritmo descrito en las diapositivas de clase no devuelve el resultado esperado para sum0 y sum 1.

Es de suponer qué he cometido algún error en la programación del algoritmo o el uso de los datos.

Command Window

Resultado y tiempos para Sigma0 = Sigma1 = 0 y alpha = 0,01

sum0

4.6505

sum1

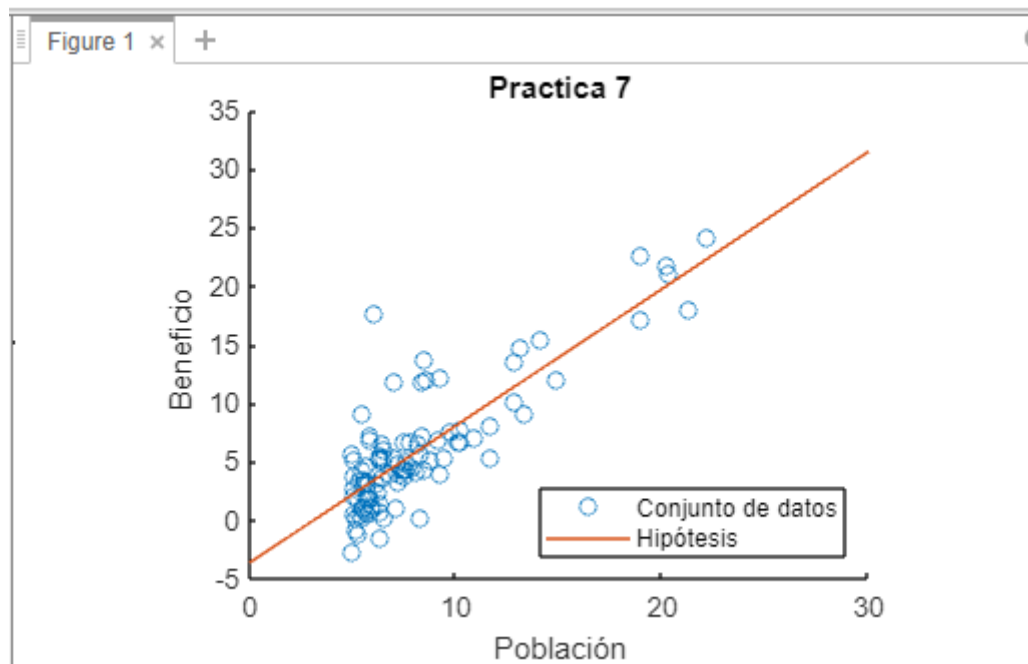
-0.4672

0.0018

>>

Actividad 7.

Dibuja sobre la figura del conjunto de datos la línea que representa el modelo óptimo y añade una leyenda para explicarla



Actividad 8.

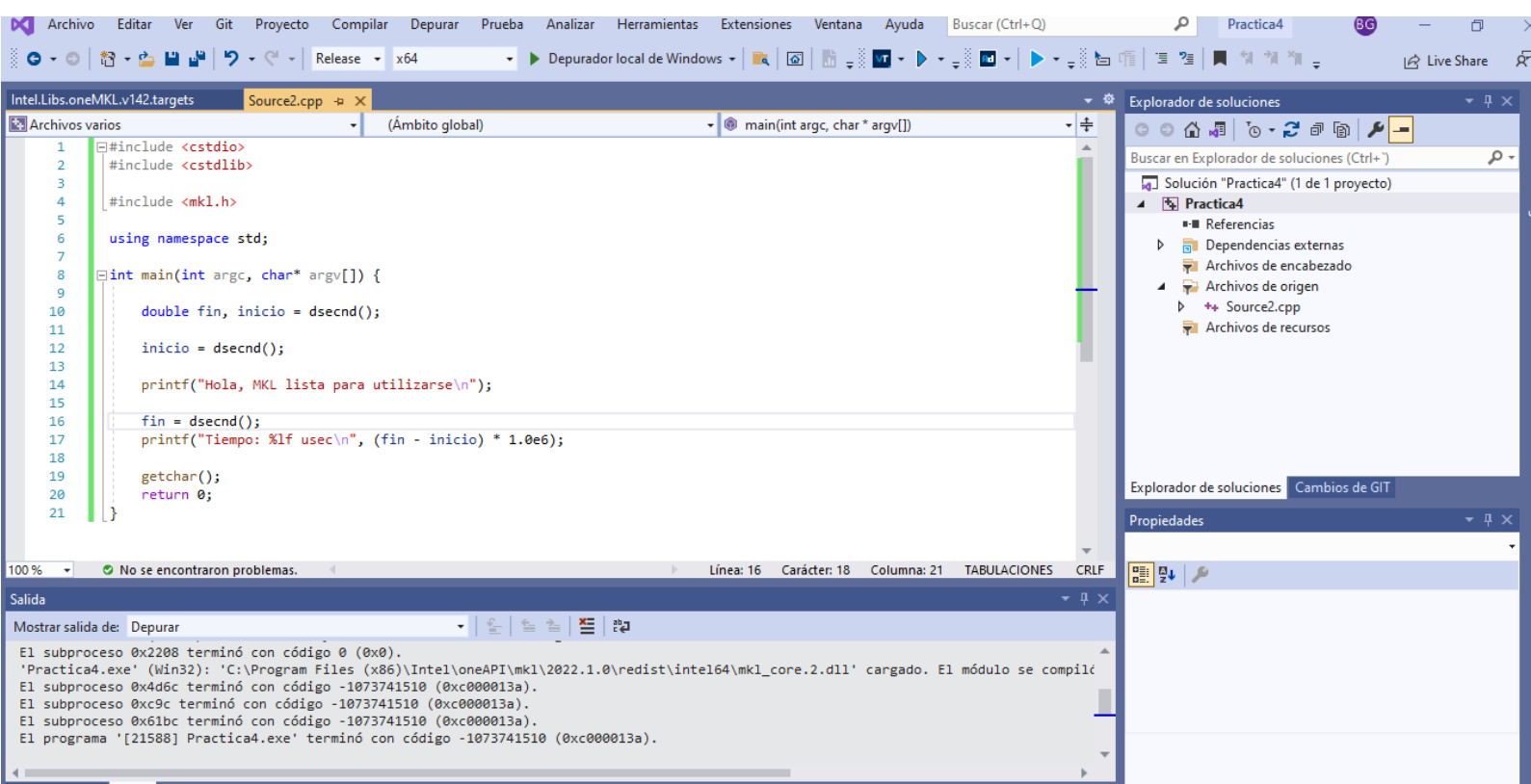
Pendiente Ver el tema de coger valores introucidos por usuario.

PRACTICA 3-4. BLASS y MKL.

A continuación exponemos la realización de la práctica 3 con sus correspondientes capturas de pantalla.

Actividad 1. Prueba MKL.

Pequeño ejemplo como el que se muestra en la captura para comprobar la correcta instalación de MKL.



```
1 #include <stdio>
2 #include <stdlib>
3
4 #include <mkl.h>
5
6 using namespace std;
7
8 int main(int argc, char* argv[]) {
9
10     double fin, inicio = dsecnd();
11
12     inicio = dsecnd();
13
14     printf("Hola, MKL lista para utilizarse\n");
15
16     fin = dsecnd();
17     printf("Tiempo: %lf usec\n", (fin - inicio) * 1.0e6);
18
19     getchar();
20     return 0;
21 }
```

Salida

Mostrar salida de: Depurar

```
El subproceso 0x2208 terminó con código 0 (0x0).
'Practica4.exe' (Win32): 'C:\Program Files (x86)\Intel\oneAPI\mkl\2022.1.0\redist\intel64\mkl_core.2.dll' cargado. El módulo se compiló
El subproceso 0x4d6c terminó con código -1073741510 (0xc00013a).
El subproceso 0xc9c terminó con código -1073741510 (0xc00013a).
El subproceso 0x61bc terminó con código -1073741510 (0xc00013a).
El programa '[21588] Practica4.exe' terminó con código -1073741510 (0xc00013a).
```

Esta sencilla prueba pone un contador de segundos y muestra por pantalla cuanto ha tardado.


```
C:\Users\TESTER\source\repos\Practica4\x64\Release\Practica4.exe
Hola, MKL lista para utilizarse
Tiempo: 103.439568 usec
```

Actividad 2.

Realizar los siguientes ejercicios usando la librería CBLAS nivel 1. Emplear Matlab/Octave para verificar que el resultado es correcto.

1. Definir dos vectores 3D ortogonales y comprobar que su producto escalar es nulo.
2. Construir dos vectores conteniendo el valor ASCII los 10 primeros caracteres de tu nombre (vN) y de tu apellido (vA), completando con 0 si es necesario.
Obtener el resultado de sumar al primer vector el triple del segundo y mostrar el resultado mapeado a caracteres 'a...z'.
3. Crear un vector conteniendo todos los dígitos de tu fecha de nacimiento. La nota final de la asignatura de MNC será el resultado de calcular el módulo 11 de la norma2 de ese vector.

```

#include <stdio.h>
#include <iostream>
#include <mkl.h>

#include "Actividad2.h"

void Actividad2::execute() {
    //Vectores Ortogonales
    double A[4] = { 0.0, 8.0, 0.0, 6.0 };
    double B[4] = { 1.0, 0.0, 3.0, 0.0 };
    //Incrementos

    //Producto escalar de dos vectores usamos cblas_ddot calcula producto escalar pasando por parametros el tamaño de los vectores los propios
    double pe = cblas_ddot(4, A, 1, B, 1);
    printf("Producto escalar de dos vectores ortogonales de A y B (siempre es): %lf\n", pe);

    //apartado 2

    double vN[10] = { int('b'), int('r'), int('i'), int('a'), int('n'), 0, 0, 0, 0, };
    double vA[10] = { int('p'), int('a'), int('l'), int('m'), int('e'), int('s'), 0, 0, 0, 0, };

    //Producto de dos vectores y = ax + y
    cblas_daxpy(10, 3, vA, 1, vN, 1);

    printf("Imprimir vector resultante de la suma de los vectores con tu nombre ---->\n");

    for (size_t i = 0; i < 10; i++) {
        std::cout << char(vN[i]) << ' ';
    }
    std::cout << '\n';

    //apartado 3

    double vD[3] = { 24, 7, 1993 };

    //cblas_dnrm2 Calcula la norma de un vector
    int res = int(cblas_dnrm2(3, vD, 1)) % 11;

    printf("mi nota en MNC es: (%i)\n", res);
}

```

Actividad 3.

Realizar los siguientes ejercicios usando la librería CBLAS nivel 2.

Emplear Octave para verificar que el resultado es correcto.

1. Definir una matriz (A) y dos vectores (x, y) y realizar las siguientes operaciones: a) $A \cdot x$ b) $3 \cdot A \cdot x + 4 \cdot y$

```
void Actividad3::execute()
{
    double m[3 * 3] = { 3, 3, 1, 6, 5, 8, 9, 8, 8 };

    double v1[3] = { 4, 1, 5 };
    double v2[3] = { 2, 1, 9 };

    //vector vacío
    double v3[3] = { 0, 0, 0 };

    //Apartado 1 void cblas_dgemv ( const CBLAS_LAYOUT layout,const CBLAS_TRANSPOSE TransA,const int m,
    //const double* A, const int lda,const double* X, const int incX,
    //const double beta, double* Y, const int incY )

    //Resultado de  $A \cdot x$  ya que tanto beta como y valen cero
    cblas_dgemv(CblasRowMajor, CblasNoTrans, 3, 3, 1, m, 3, v1, 1, 0, v3, 1);

    printf("Mostramos el resultado de  $y = A \cdot x$  ---->\n");
    for (double i : v3)
    {
        std::cout << i << ' ';
    }
    std::cout << '\n';
}
```

2. Probar el efecto de los parámetros de layout y trasposición en el ejercicio anterior.

```
//Apartado 2 usa dgemv para obtener  $y = 3 \cdot A \cdot x + 4 \cdot y$ 

double alpha = 3;
double beta = 4;

printf("Mostramos el resultado de  $y = 3 \cdot A \cdot x + 4 \cdot y$  ---->\n");
cblas_dgemv(CblasRowMajor, CblasNoTrans, 3, 3, alpha, m, 3, v1, 1, beta, v2, 1);

for (double i : v2)
{
    std::cout << i << ' ';
}
std::cout << '\n';

printf("Mostramos el resultado de  $y = 3 \cdot A \cdot x + 4 \cdot y$  pero variando los valores del layout y la traspuesta ---->\n");

//Los valores CblasRowMajor, CblasNoTrans son valores struct dentro de las librerías de blas aquí cambiaremos CblasRowMajor=101 para CblasColMajor
//Al usar CblasTrans multiplicamos con la traspuesta así quedaría  $3 \cdot A^T \cdot x + 4 \cdot y$ 
cblas_dgemv(CblasColMajor, CblasTrans, 3, 3, alpha, m, 3, v1, 1, beta, v2, 1);

//También cambiamos el valor de LDA pero no entiendo muy bien porqué he buscado en internet y suelen usarlo en los ejemplos como en Octave
```

Los parámetros de layout y trasposición son tipos estructurados dentro de las librerías de blas.

En el ejemplo cambiamos las opciones de para hacer las operaciones con una matriz traspuesta a modo $3 \cdot A^T \cdot x + 4 \cdot y$

3. [OPTATIVO] Probar el efecto de los parámetros de incremento y lda en el ejercicio anterior.

```
//Los valores CblasRowMajor, CblasNoTrans son valores struc dentro de las librerias de blas
//Al usar CblasTrans multiplicamos con la traspuesta así quedaría 3*A*T*x + 4*y
cblas_dgemv(CblasColMajor, CblasTrans, 3, 3, alpha, m, 3, v1, 1, beta, v2, 1);

//Tambien cambiamos el valor de LDA pero no entiendo muy bien porqué he buscado en internet y

for (double i : v2)
{
    std::cout << i << ' ';
}
std::cout << '\n';
```

Resultado por consola:

```
Actividad 3
Mostramos el resultado de y = A*x ---->
20 69 84
Mostramos el resultado de y = 3*A*x + 4*y ---->
68 211 288
Mostramos el resultado de y = 3*A*x + 4*y pero variando los valores del layout y la traspuesta ---->
332 1051 1404
```

Actividad 4

Realizar los siguientes ejercicios usando la librería CBLAS nivel 3.

Emplear Octave para verificar que el resultado es correcto.

1. Definir tres matrices (A, B y C) de dimensión 3x3 y realizar las operaciones: a) $A*B$ b) $A*BT$ c) $2*A*B + 3*C$

```
void Actividad4::execute()
{
    double m1[3 * 3] = { 3, 4, 1, 4, 5, 4, 3, 2, 7 };
    double m2[3 * 3] = { 2, 2, 1, 6, 2, 4, 9, 2, 7 };
    double m3[3 * 3] = { 3, 1, 1, 8, 5, 4, 3, 1, 1 };

    double m0[3 * 3] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    //void cblas_dgemm ( const CBLAS_LAYOUT layout,
    //const CBLAS_TRANSPOSE TransA,
    //const CBLAS_TRANSPOSE TransB,
    //const int M, const int N, const int K,
    //const double alpha, const double* A, const int lda,
    //const double* B, const int ldb,
    //const double beta, double* C, const int ldc )

    //lda ldb y ldc como suponíamos suele coincidir con el numero de columnas
    printf("Mostramos el resultado de C = A*B ---->\n");
    //C = A*B
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, 3, 3, 3, 1, m1, 3, m2, 3, 0, m0, 3);
    for (double i : m0)
    {
        std::cout << i << ' ';
    }
    std::cout << '\n';

    printf("Mostramos el resultado de C = A*Bt ---->\n");
    //C = A*Bt
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, 3, 3, 3, 1, m1, 3, m2, 3, 0, m0, 3);
    for (double i : m0)
    {
        std::cout << i << ' ';
    }
    std::cout << '\n';
    printf("Mostramos el resultado de C = A*B +3*C ---->\n");
    //C = 2*A*B + 3*C
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, 3, 3, 3, 2, m1, 3, m2, 3, 3, m3, 3);
    for (double i : m3)
    {
```

En este caso usamos la función `cblas_dgemm` que nos permite operar matrices con matrices.

2. [OPTATIVO] Realizar una operación que implique matrices no cuadradas, y que genere como resultado una matriz de 5x5.

Actividad 5.

```
//Incompleta, no se que son los gFlops ni como calcularlos

void Actividad5::execute(int N)
{
    //Generacion de matrices aleatorias y reserva de espacio
    double* A = (double*)mkl_malloc(N * N * sizeof(double), 64);
    double* B = (double*)mkl_malloc(N * N * sizeof(double), 64);
    double* C = (double*)mkl_malloc(N * N * sizeof(double), 64);

    for (int i = 0; i < N * N; ++i)
    {
        A[i] = (double)rand() / (double)100;
        B[i] = (double)rand() / (double)100;
        C[i] = (double)rand() / (double)100;
    }

    //100 Veces C = A*B
    for (int i = 0; i < 100; ++i)
    {
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1, A, N, B, N, 0, C, N);
    }

    mkl_free(A);
    mkl_free(B);
    mkl_free(C);
}
```

La actividad 5 no he podido completarla porque no sabía bien como calcular los Gflops.

Práctica 5 . Lapack

• LAPACK – LU computational

Realizar los siguientes ejercicios usando la librería LAPACK. Emplear Octave para verificar que el resultado es correcto.

1. Generar con Octave una matriz aleatoria de 6x6 con números de 1 a 10, comprobando que su determinante no sea nulo. Realizar la factorización LU con pivotamiento.

Realizamos el script en Matlab generamos la matriz hallamos su determinante siendo este no nulo y su factorización LU usando las funciones de matlab.

```
clc;
clear;

%%Generamos matriz aleatoria de 6x6
M = double(randi([1,10], 6,6));

detM = det(M);

disp("Matriz M de 6x6: ");
disp(M);

disp("Determinante de M: " + detM);
disp(" ");

disp("Factorización LU de M: ")
luA = lu(M);
disp(luA);
```

7	8	8	2	8	6
4	8	8	5	3	2
10	2	3	10	6	2
1	5	7	4	7	3
5	5	7	6	9	9
4	7	2	3	10	3

Determinante de M: 99632

10.0000	2.0000	3.0000	10.0000	6.0000	2.0000
0.4000	7.2000	6.8000	1.0000	0.6000	1.2000
0.4000	0.8611	-5.0556	-1.8611	7.0833	1.1667
0.7000	0.9167	0.0659	-5.7940	2.7830	3.4231
0.1000	0.6667	-0.4286	-0.2651	9.7734	3.4073
0.5000	0.5556	-0.3407	0.0327	0.8174	4.8337

>>

2. Utilizar LAPACK a nivel computacional para obtener:

- Factorización LU
- Determinante
- Matriz inversa a partir de resolver el sistema $AX = I$
- Calcular la inversa usando la rutina `_dgetri()`

3. [OPTATIVO] Evaluar el error que se comete en las operaciones.

Tema 6. Matrices Dispersas.

Actividad 1. Matlab

Ejercicio 1.

Codificación COO y CSR – Matlab/Octave

1. Programar dos funciones en Matlab/Octave que devuelvan la codificación COO y CSR para una matriz que se le pase como entrada.

```
[row, col, val] = COO( A );
```

```
1.m x +  
function [row, col, val] = COO(A)  
    n = nnz(A); %Numero de elemntos no cero de la matriz  
  
    %vectores COO  
    row = zeros(1,n);  
    col = zeros(1,n);  
    val = zeros(1,n);  
  
    k=1;  
    for i = 1:size(A,1)  
        for j = 1:size(A,2)  
            if A(i,j) ~= 0  
                row(k) = i;  
                col(k) = j;  
                val(k) = A(i,j);  
                k = k +1;  
            end  
        end  
    end  
end
```


[rowOff, col, val] = CSR(A);

```
function [rowOff, col, val] = CSR(A)
    n = nnz(A); %elementos no cero

    z = 0;
    %Tamaño fila + 1
    rowOff = zeros(1,size(A,1)+1); %cada posicion del vector indica el numero de elemento no nules encontrados en total hasta la fila anterior
    col = zeros(1,n);
    val = zeros(1,n);

    k = 1;
    l = 1;

    %Se comprueba que la primera poscicion de la matriz no es nula
    if A(1,1) ~= 0
        rowOff(1) = 0;
        l = 2;
    end

    for i = 1:size(A,1)
        for j = 1:size(A,2)
            if A(i,j) ~= 0
                col(k) = j;
                val(k) = A(i,j);
                k = k + 1;
                z = z + 1;
            end
        end
        rowOff(l) = z;
        l = l + 1;
    end
end
```

Vemos la comprobación de los resultados.

```

M =

     1     0     3     0
     0     0     1     2
    12     0     2     3
     2     0     0     0

*****
Despues de plicar la codificacion COO
Filas
     1     1     2     2     3     3     3     4

Columnas
     1     3     3     4     1     3     4     1

Valores
     1     3     1     2    12     2     3     2

*****
Despues de plicar la codificacion CSR
Valores no cero en las Filas
     0     2     4     7     8

Columnas
     1     3     3     4     1     3     4     1

Valores
     1     3     1     2    12     2     3     2

*****
..

```

Ejercicio 2.

Programar un generador de matrices aleatorias escasas en Matlab/Octave, tomando como entrada la densidad deseada de valores nulos para la matriz y el rango de valores.

[A] = GenerateSparse(nRow, nCol, zDensity, vMin, vMax);

```

function [A] = generateSparse(nRow, nCol, zDensity, vMin, vMax)

%Creamos una matriz randomizada con tamaño row*col y con un rango de valores
%min-max
A = randi([vMin,vMax],nRow,nCol);

%Rellenamos la matriz con el numero de ceros dados por nDensity

z = zDensity +1;
for i = 1:z
    r = randi(nRow);%Numeros random desde 1 al tamaño col-row
    c = randi(nCol);
    %Lo introducimos en la matriz
    if(A(r,c) ~= 0)% Por si me encuentro un 0 de una iteracion anterior
        A(r,c) = 0;
    else
        i = i+1;
    end
end

end

end

```

Y ahora mostramos una matriz generada.

M =

3	6	3	0
4	5	0	0
7	3	1	5
2	2	0	2

Generamos una matriz aleatoria

3	6	3	0
4	5	0	0
7	3	1	5
2	2	0	2

1

Practica 8. OpenMP

Actividad 1.

Ejecuta el programa varias veces para comprobar que no es determinista, ya que el orden en el que aparecen los mensajes varía en cada ejecución .

Varias ejecuciones que demuestran que no da resultados deterministas

```
C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe
[COMIENZO]
Hola desde el hilo 0 --- Soy el hilo maestro, somos 12
Hola desde el hilo 10
Hola desde el hilo 7
Hola desde el hilo 6
Hola desde el hilo 8
Hola desde el hilo 3
Hola desde el hilo 2
Hola desde el hilo 4
Hola desde el hilo 9
Hola desde el hilo 1
Hola desde el hilo 5
Hola desde el hilo 11
[FINAL]

C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe
[COMIENZO]
Hola desde el hilo 0 --- Soy el hilo maestro, somos 12
Hola desde el hilo 1
Hola desde el hilo 2
Hola desde el hilo 8
Hola desde el hilo 5
Hola desde el hilo 3
Hola desde el hilo 4
Hola desde el hilo 7
Hola desde el hilo 6
Hola desde el hilo 9
Hola desde el hilo 10
Hola desde el hilo 11
[FINAL]
```

Modifica el programa para que solo el hilo maestro indique el total de hilos.

```
#include <omp.h>
#include <stdio.h>
void main()
{
    printf("[COMIENZO]\n");
    #pragma omp parallel num_threads(12)
    {
        int id = omp_get_thread_num();
        int num = omp_get_num_threads();
        if (id == 0) {
            printf("Hola desde el hilo %d --- Soy el hilo maestro, somos %d\n", id, num);
        }
        else {
            printf("Hola desde el hilo %d\n", id);
        }
    }
    printf("[FINAL]\n");
    std::getchar();
    return;
}
```

```
[COMIENZO]
Hola desde el hilo 4
Hola desde el hilo 6
Hola desde el hilo 5
Hola desde el hilo 0 --- Soy el hilo maestro, somos 12
Hola desde el hilo 1
Hola desde el hilo 3
Hola desde el hilo 7
Hola desde el hilo 2
Hola desde el hilo 8
Hola desde el hilo 9
Hola desde el hilo 10
Hola desde el hilo 11
[FINAL]
```

Actividad 2

Modifica el programa anterior para que cada hilo realice una tarea que consuma tiempo como, por ejemplo, multiplicar dos números en coma flotante varios millones de veces

```

/*ACTIVIDAD 2
* printf("[COMIENZO]\n");

#pragma omp parallel num_threads(5) // nº de hilos
{
    int id = omp_get_thread_num();
    int num = omp_get_num_threads();

    double tinicial = omp_get_wtime(); //recogemos en tiempo inicial
    float res = 4.5;
    for (float i = 1; i < 15000000; i++) {
        res = res * i * res;
    }
    double tfinal = omp_get_wtime();
    double tiempo = tfinal - tinicial;

    printf("Operacion hilo nº %d Tiempo = %4.8f segundos el resultado es %4.4f\n", id, tiempo, res);
}
//A partir de 5 hilos los tiempos empiezan a variar

printf("[FINAL]\n");
*/

```

- Añade al mensaje que muestra cada hilo el tiempo que ha tardado en ejecutar la operación
- Realiza varias pruebas cambiando el número total de hilos y determina si, a partir de los datos obtenidos, puedes verificar el número de hilos que es capaz de ejecutar el procesador de forma simultánea

En las capturas de abajo podemos ver cómo a partir de 5 hilos en paralelo los tiempos aumentan.

C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe

```
[COMIENZO]
Operacion hilo n 3 Tiempo = 0.04105010 segundos el resultado es inf
Operacion hilo n 0 Tiempo = 0.04200550 segundos el resultado es inf
Operacion hilo n 1 Tiempo = 0.04429700 segundos el resultado es inf
Operacion hilo n 2 Tiempo = 0.04541240 segundos el resultado es inf
[FINAL]
```

C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe

```
[COMIENZO]
Operacion hilo n 0 Tiempo = 0.04263550 segundos el resultado es inf
Operacion hilo n 3 Tiempo = 0.04559470 segundos el resultado es inf
Operacion hilo n 2 Tiempo = 0.04564520 segundos el resultado es inf
Operacion hilo n 1 Tiempo = 0.05269920 segundos el resultado es inf
Operacion hilo n 4 Tiempo = 0.05342760 segundos el resultado es inf
[FINAL]
```

Actividad 3.

- Escribe un programa que sume dos vectores de números en coma flotante
 - Declara tres vectores de 100 elementos
 - Inicializa cada elemento del primero con el valor de su índice
 - Inicializa cada elemento del segundo con el doble del valor de su índice
 - Suma los dos vectores en el tercero y comprueba el resultado
- Paraleliza el código de forma que haya cuatro hilos entre los que se repartan grupos de 10 iteraciones planificadas de forma dinámica; ten cuidado al determinar qué variables son privadas y cuáles compartidas

```
#define CHUNK 10
```

```
#pragma omp for schedule(dynamic,CHUNK)
```

- Modifica el programa para que cada hilo muestre qué elementos del vector resultado ha calculado y comprueba las variaciones que se producen en diferentes ejecuciones

```

* //Actividad 3
int tam = 100;
double* v1 = new double[tam];
double* v2 = new double[tam];
double* v3 = new double[tam];

for (int i = 0; i < tam; i++)
{
    v1[i] = (double)i;
    v2[i] = 2.0 * (double)i;
}

#pragma omp parallel num_threads(4)
{
    int id = omp_get_thread_num();
    int num = omp_get_num_threads();

#define CHUNK 10

#pragma omp for schedule(dynamic, CHUNK) //Al tener un comportamiento dinámico los hilos no es
    for (int i = 0; i < tam; i++) {
        v3[i] = v1[i] + v2[i];
        printf(" Hilo %d esta calculando %f Indice=%d \n ", id, v3[i], i);
    }
}
*/

```

C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe

```
Hilo 1 esta calculando 258.000000 Indice=86
Hilo 1 esta calculando 261.000000 Indice=87
Hilo 1 esta calculando 264.000000 Indice=88
Hilo 1 esta calculando 267.000000 Indice=89
Hilo 1 esta calculando 270.000000 Indice=90
Hilo 1 esta calculando 273.000000 Indice=91
Hilo 1 esta calculando 276.000000 Indice=92
Hilo 1 esta calculando 279.000000 Indice=93
Hilo 1 esta calculando 282.000000 Indice=94
Hilo 1 esta calculando 285.000000 Indice=95
Hilo 1 esta calculando 288.000000 Indice=96
Hilo 1 esta calculando 291.000000 Indice=97
Hilo 1 esta calculando 294.000000 Indice=98
Hilo 1 esta calculando 297.000000 Indice=99
Hilo 0 esta calculando 141.000000 Indice=47
Hilo 0 esta calculando 144.000000 Indice=48
Hilo 0 esta calculando 147.000000 Indice=49
Hilo 2 esta calculando 210.000000 Indice=70
Hilo 2 esta calculando 213.000000 Indice=71
Hilo 2 esta calculando 216.000000 Indice=72
Hilo 2 esta calculando 219.000000 Indice=73
Hilo 2 esta calculando 222.000000 Indice=74
Hilo 2 esta calculando 225.000000 Indice=75
Hilo 2 esta calculando 228.000000 Indice=76
Hilo 2 esta calculando 231.000000 Indice=77
Hilo 2 esta calculando 234.000000 Indice=78
Hilo 2 esta calculando 237.000000 Indice=79
Hilo 3 esta calculando 90.000000 Indice=30
Hilo 3 esta calculando 93.000000 Indice=31
Hilo 3 esta calculando 96.000000 Indice=32
Hilo 3 esta calculando 99.000000 Indice=33
Hilo 3 esta calculando 102.000000 Indice=34
Hilo 3 esta calculando 105.000000 Indice=35
Hilo 3 esta calculando 108.000000 Indice=36
Hilo 3 esta calculando 111.000000 Indice=37
Hilo 3 esta calculando 114.000000 Indice=38
Hilo 3 esta calculando 117.000000 Indice=39
```

Desde que un hilo se libera sigue sumando ya que la suma es dinámica.

Actividad 4.

- Modifica el programa anterior para que la planificación se realice de forma estática y ejecútalo varias veces para comprobar los cambios que se producen en su comportamiento
- Añade al programa las instrucciones necesarias para medir el tiempo de ejecución de la suma de los dos vectores y comprueba las diferencias que se producen
 - Utiliza varios tamaños de vector (100, 1000, 10000...)
 - Utiliza varios tamaños de grupo de iteraciones (10, 100...)
 - Dibuja una gráfica que detalle las diferencias encontradas

```
* //Actividad 4
printf("[COMIENZO]\n");


for ( int tam = 100; tam < 10000000 ; tam+= 1) //Bucle for que irá aumentando el tamaño de los vec
{
    double* v1 = new double[tam];
    double* v2 = new double[tam];
    double* v3 = new double[tam];

    for (int i = 0; i < tam; i++) //preparamos los vectores para la suma como el ej anterior
    {
        v1[i] = (double)i;
        v2[i] = 2.0 * (double)i;
    }

    int chunk = 10;
    double tinicial = omp_get_wtime(); //Rcogemos el tiempo
    #pragma omp parallel num_threads(4)
    {
        int id = omp_get_thread_num();
        int num = omp_get_num_threads();

        #pragma omp for schedule(static, chunk) //Estatico

        for (int i = 0; i < tam; i++) {
            v3[i] = v1[i] + v2[i];
        }
    }
}
```

 C:\Users\TESTER\source\repos\practicaOpenMP\x64\Release\practicaOpenMP.exe

```
[COMIENZO]
tiempo con tamaño 100 y 10 iteraciones: 0.000306
tiempo con tamaño 1001 y 10 iteraciones: 0.000010
tiempo con tamaño 10011 y 10 iteraciones: 0.000076
tiempo con tamaño 100111 y 10 iteraciones: 0.000566
tiempo con tamaño 1001111 y 10 iteraciones: 0.003427
tiempo con tamaño 100 y 1000 iteraciones: 0.000005
tiempo con tamaño 1001 y 1000 iteraciones: 0.000005
tiempo con tamaño 10011 y 1000 iteraciones: 0.000031
tiempo con tamaño 100111 y 1000 iteraciones: 0.000218
tiempo con tamaño 1001111 y 1000 iteraciones: 0.003222
[FINAL]
```

Actividad 5.

- Escribe un programa que sume y multiplique, de forma separada, los elementos de dos vectores de números en coma flotante
 - Declara cuatro vectores de 100 elementos
 - Inicializa cada elemento del primero con el valor de su índice
 - Inicializa cada elemento del segundo con el doble del valor de su índice
 - Suma los dos vectores en el tercero y comprueba el resultado
 - Multiplica los dos vectores en el cuarto y comprueba el resultado
- Paraleliza el código de forma que haya dos secciones; ten cuidado al determinar qué variables son privadas y qué variables son compartidas
 - La primera sección realizará la suma
 - La segunda sección realizará la multiplicación
- Modifica el programa para que cada hilo muestre qué elementos de cada vector resultado ha calculado y comprueba las variaciones que se producen en diferentes ejecuciones