Practica 1

Álgebra lineal numérica

Introducción a la Computación Científica

Andrés Río Nogués

Part 1 Resolució de sistemes triangulars

Objectiu : Implementar una funció per a resoldre un sistema lineal triangular inferior Lx = b i una funció per a resoldre un sistema lineal triangular superior Ux = b.

Funcions Utilitzades :

ResolLinf

Aquesta funció el que fa es calcular la solució d'un sistema triangular inferior amb la forma Ax = b sent "A" una matriu qualsevol.

```
int resolLinf (int n, double **a, double *b, double *x, double tol, int ind_diag){

double suma = 0.0;

int i,j;
double valorAbs;

for (i = 0; i<n ; i++) {

        valorAbs = fabs(a[i][i]);

        if (valorAbs < tol){

                return 0;

        }

        suma = 0.0;

        for (j = 0; j < i; j++) {

                suma += a[i][j] * x[j];

        }


        if (ind_diag == 1){

                x[i] = ( b[i] - suma  );

        } else {

                x[i] = ( b[i] - suma  ) / a[i][i];

        }

}

return 1;
```

L'algorisme es basa en anar de la primera fila on només hi ha una incógnita fins l'última calculant el sistema. Les incògnites ja calculades les restem a el valor del vector b. Com la entrada es una matriu qualsevol tenim que evitar agafar valors que no siguin de la part superior per això imposem que la j (iterador de columnes) mai sigui major a i (iterador de files).

Per comprovar que la divisió feta no sigui amb valors propers a zero, comprovem que els valors de la diagonal sigui diferent a una tolerancia donada (sobretot per evitar propagació d'errors).

A més per fer l'algorisme més simple, comprovem amb un paràmetre que la diagonal no sea d'uns, per evitar càlculs innecessaris i per implementacions futures.

ResolUsup

Aquesta funció el que fa es calcular la solució d'un sistema triangular superior amb la forma Ax = b sent "A" una matriu qualsevol.

```
int resolUsup (int n, double **a, double *b, double *x, double tol){
double suma = 0.0;
int i,j;
double valorAbs;
for (i = n-1 ; i >= 0; i--) {
        valorAbs = fabs(a[i][i]);
        if (valorAbs < tol){
                return 0;
        }
        suma = 0.0;
        for (j = i; j<n; j++) {
                suma += a[i][j] * x[j];
        }
        x[i] = ( b[i] - suma  ) / a[i][i];
        }
return 1;
}
```

Mateix algorisme pero fent l'iteració al revés, imposant que el iterador de columnes j sempre sigui i o superior i sense tenir en compte la possibilitat de tenir uns a la diagonal (no seran necessaris per a implementacions futures).

Residu

Aquesta funció calcula el vector residu de la forma r = b - Ax.

```
void residu (int n, double **a, double *b, double *x, double *r){

int i;
prodMatVec ( n, n, a, x, r);
for (i = 0; i<n ; i++ ){
r[i] = b[i] - r[i];
}


}
```

Per obtenir Ax utilitzem la funció prodMatVec que guarda el valor en el vector r i el restem a b.

ProdMatVec

Calcula el producte Matriu per vector i el guarda en altre vector

```
void prodMatVec ( int n, int m, double **a, double *x, double *y) {
int fila,columna;
for (fila = 0; fila < n; fila ++) {
y[fila] = 0;
for (columna = 0; columna < m; columna ++){
        y[fila] += a[fila][columna] * x[columna];
}}}
```

El que fa aquest algorisme es multiplicar cada valor de una fila de a per cada valor d'una columna del vector i guardarlo en la respectiva fila del vector resultant

Prod_esc

El que fa es calcular la suma dels producte de 2 vectors (utilitzat per calcular la norma 2 del vector residu).

```
double prod_esc ( int n, double *x, double* y){
double resultado = 0.0;
int i;
for (i=0; i<n; i++) {
        resultado += x[i] * y[i];
}
return resultado;
}
```

Main Triang

Important : Aquest main llegeix els inputs d'un fitxer exclusivament anomenat "`inputsTriang.dat`" i la seva estructura és:

```
4 0 0 1e-12

1 0 0 0
2 3 0 0
3 2 -1 0
4 3 2 1

-4 -5 -6 -2
```

En la primera fila tenim (en ordre):

n - tamany de la matriu quadrada
ind_diag - per indicar si la matriu té uns a la diagonal
ind_trasposta - per indicar si la matriu es inferior (0) o superior (1)
tolerancia - valor considerat com 0

A continuació esta la matriu A (sempre serà triangular inferior i està representada correctament amb files i columnes)

La última fila serà el vector B

Implementació del main

La majoria del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes.

Parts importants:

Transposar la matriu si és necessari

```
if (transposada == 1) {

for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {


            temp = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = temp;
```

**en aquest intercanvi utilitzem una variable temporal per evitar la pèrdua de dades, a més de revisar només les columnes superiors a les files per evitar intercanviar la mateixa dada 2 cops (acabariem amb la mateixa matriu).*

Seleccionar quina funció trucar

```
if (transposada == 1){

if ( resolUsup(n,a,b,x,tol) == 0 ){

        printf("Error al resol U sup \n");
        return -1;
}

} else {

if ( resolLinf(n,a,b,x,tol,ind_diag) == 0 ){

        printf("Error al resol L inf \n");
        return -1;
```

Si la funció és trasposta farem resolUsup i sinó resolLinf. A més si detectem qualsevol error amb la tolerancia, parem per complet el programa.

Crida de funcions

```
residu (n,a,b,x,r);
prodEsc = prod_esc(n,r,r);
prodEsc = sqrt (prodEsc);
```

Truquem residu per aconseguir la r i fem producte escalar de 2 r per aconseguir la norma 2 ($\sqrt{\Sigma x_i^2}$)

La resta del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes i veure els resultats.

RESULTATS MAIN TRIANG

$$L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 3 & 2 & -1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix}, b_1 = \begin{pmatrix} -4 \\ -5 \\ -6 \\ -2 \end{pmatrix}$$

Resultats de Lx = b i $L^t x$ = b con diagonal 0 i 1.

Diagonal 0 trasposta 0

```
INPUTS
n : 4
ind_diag : 0
ind_transposada : 0
tolerancia :    1.0000000e-12

Matriu a
     1.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
    2.0000000e+00    3.0000000e+00    0.0000000e+00    0.0000000e+00
    3.0000000e+00    2.0000000e+00   -1.0000000e+00    0.0000000e+00
    4.0000000e+00    3.0000000e+00    2.0000000e+00    1.0000000e+00
Vector b
    -4.0000000e+00   -5.0000000e+00   -6.0000000e+00   -2.0000000e+00


 *** POST FUNCIONS ***

 Vector x
    -4.0000000e+00    1.0000000e+00   -4.0000000e+00    1.9000000e+01
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

Diagonal 0 trasposta 1

```
INPUTS
n : 4
ind_diag : 0
ind_transposada : 1
tolerancia :    1.0000000e-12

Matriu a
     1.0000000e+00    2.0000000e+00    3.0000000e+00    4.0000000e+00
    0.0000000e+00    3.0000000e+00    2.0000000e+00    3.0000000e+00
    0.0000000e+00    0.0000000e+00   -1.0000000e+00    2.0000000e+00
    0.0000000e+00    0.0000000e+00    0.0000000e+00    1.0000000e+00
Vector b
    -4.0000000e+00   -5.0000000e+00   -6.0000000e+00   -2.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     0.0000000e+00   -1.0000000e+00    2.0000000e+00   -2.0000000e+00
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

Diagonal 1 trasposta 0

```
INPUTS
n : 4
ind_diag : 1
ind_transposada : 0
tolerancia :    1.0000000e-12

Matriu a
     1.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
    2.0000000e+00    3.0000000e+00    0.0000000e+00    0.0000000e+00
    3.0000000e+00    2.0000000e+00   -1.0000000e+00    0.0000000e+00
    4.0000000e+00    3.0000000e+00    2.0000000e+00    1.0000000e+00
Vector b
    -4.0000000e+00   -5.0000000e+00   -6.0000000e+00   -2.0000000e+00


 *** POST FUNCIONS ***

 Vector x
    -4.0000000e+00    3.0000000e+00    0.0000000e+00    5.0000000e+00
Vector r
     0.0000000e+00   -6.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    6.0000000e+00
```

Diagonal 1 trasposta 1

```
INPUTS
n : 4
ind_diag : 1
ind_transposada : 1
tolerancia :    1.0000000e-12

Matriu a
     1.0000000e+00    2.0000000e+00    3.0000000e+00    4.0000000e+00
    0.0000000e+00    3.0000000e+00    2.0000000e+00    3.0000000e+00
    0.0000000e+00    0.0000000e+00   -1.0000000e+00    2.0000000e+00
    0.0000000e+00    0.0000000e+00    0.0000000e+00    1.0000000e+00
Vector b
    -4.0000000e+00   -5.0000000e+00   -6.0000000e+00   -2.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     0.0000000e+00   -1.0000000e+00    2.0000000e+00   -2.0000000e+00
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

$$L_2 = \begin{pmatrix} 1.0234 & 0 & 0 & 0 \\ 2.0981 & -6.9876 & 0 & 0 \\ 9.9871 & 2.2222 & -1.9870 & 0 \\ 1.1 & 0.3333 & 20.121 & 1.1234 \end{pmatrix}, b_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Resultats de Lx = b i $L^t x = b$ con diagonal 0 i 1.

Diagonal 0 Trasposta 0

```
INPUTS
n : 4
ind_diag : 0
ind_transposada : 0
tolerancia :    1.0000000e-12

Matriu a
     1.0234000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
    2.0981000e+00   -6.9876000e+00    0.0000000e+00    0.0000000e+00
    9.9871000e+00    2.2222000e+00   -1.9870000e+00    0.0000000e+00
    1.1000000e+00    3.3330000e-01    2.0121000e+01    1.1234000e+00
Vector b
     1.0000000e+00    0.0000000e+00    1.0000000e+00    0.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     9.7713504e-01    2.9339502e-01    4.7361489e+00   -8.5872074e+01
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

Diagonal 0 Trasposta 1

```
INPUTS
n : 4
ind_diag : 0
ind_transposada : 1
tolerancia :    1.0000000e-12

Matriu a
     1.0234000e+00    2.0981000e+00    9.9871000e+00    1.1000000e+00
    0.0000000e+00   -6.9876000e+00    2.2222000e+00    3.3330000e-01
    0.0000000e+00    0.0000000e+00   -1.9870000e+00    2.0121000e+01
    0.0000000e+00    0.0000000e+00    0.0000000e+00    1.1234000e+00
Vector b
     1.0000000e+00    0.0000000e+00    1.0000000e+00    0.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     6.2165552e+00   -1.6005058e-01   -5.0327126e-01    0.0000000e+00
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

Diagonal 1 Trasposta 0

```
INPUTS
n : 4
ind_diag : 1
ind_transposada : 0
tolerancia :    1.0000000e-12

Matriu a
     1.0234000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
    2.0981000e+00   -6.9876000e+00    0.0000000e+00    0.0000000e+00
    9.9871000e+00    2.2222000e+00   -1.9870000e+00    0.0000000e+00
    1.1000000e+00    3.3330000e-01    2.0121000e+01    1.1234000e+00
Vector b
     1.0000000e+00    0.0000000e+00    1.0000000e+00    0.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     1.0000000e+00   -2.0981000e+00   -4.3247022e+00    8.6616629e+01
Vector r
    -2.3400000e-02   -1.6758784e+01   -1.2917885e+01   -1.0688492e+01
Norma 2 :    2.3705970e+01
```

Diagonal 1 Trasposta 1

```
INPUTS
n : 4
ind_diag : 1
ind_transposada : 1
tolerancia :    1.0000000e-12

Matriu a
     1.0234000e+00    2.0981000e+00    9.9871000e+00    1.1000000e+00
    0.0000000e+00   -6.9876000e+00    2.2222000e+00    3.3330000e-01
    0.0000000e+00    0.0000000e+00   -1.9870000e+00    2.0121000e+01
    0.0000000e+00    0.0000000e+00    0.0000000e+00    1.1234000e+00
Vector b
     1.0000000e+00    0.0000000e+00    1.0000000e+00    0.0000000e+00


 *** POST FUNCIONS ***

 Vector x
     6.2165552e+00   -1.6005058e-01   -5.0327126e-01    0.0000000e+00
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
Norma 2 :    0.0000000e+00
```

Part 2 Factorització LU sense pivotatge

Objectiu : resoldre el sistema de matrius Ax = b usant la descomposició LU = A sense pivotatge.

Descomposició LU:

part triangular superior (U) : resultat d'eliminació gaussiana sense pivotatge
part triangular estrictament inferior (L) : multiplicadors de l'eliminació gaussiana

Funcions Utilitzades :

LU

```
int lu (int n, double **a, double tol) {


int fila, filaInferior, columna;
double multiplicador;
double valorAbs;


for (fila = 0; fila < n  ; fila ++ ){

for (filaInferior = fila + 1; filaInferior < n ; filaInferior ++) {

        valorAbs = fabs(a[fila][fila]);

        if (valorAbs < tol){

                return 0;

        }

        multiplicador = a[filaInferior][fila] / a[fila][fila];
        a[filaInferior][fila] = multiplicador;

        for (columna = fila + 1; columna < n; columna ++) {

        a[filaInferior][columna] -= a[fila][columna] * multiplicador;

}}}

return 1;

}
```

Aquest codi com l'anterior revisa que el valor a dividir no sigui menor al valor mínim acceptat per el programa (la tolerància).

Aquest algorisme el primer que fa es calcular el multiplicador ($m = a_{ik} / a_{kk}$ [1])

1 : " sent i la fila inferior de k "

i guardar-lo directament en la posició on hi hauria el 0 de la matriu gaussiana. Posteriorment actualitzem tota la fila inferior.

Per actualitzar iterem tota la fila, columna per columna anant del valor de després del multiplicador fins al final, i guardem el valor calculat en la posició corresponent.

Aquest valor calculat depèn del multiplicador utilitzat en la fila corresponent i dels valors de la columna iterada.

El resultat d'aquesta funció és la matriu a com les matrius L i U.

Main_LUSolver

Important : Aquest main llegeix els inputs d'un fitxer exclusivament anomenat "`inputsLU.dat`" i la seva estructura és:

```
4 1e-3

1 2 3 4
-2 1 2 3
-3 -2 1 2
-4 -3 -2 1

1 1 1 1
```

En la primera fila tenim (en ordre):

n - tamany de la matriu quadrada
tolerancia - valor considerat com 0

a continuació esta la matriu A (està representada correctament amb files i columnes)

La última fila serà el vector B

Implementació del main

La majoria del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes. Les parts més importants :

Crida de funcions

```
if ( lu(n,a,tol) == 0 ){
        printf("Error al LU de a \n");
        return -1;
}

if (resolLinf (n, a, b, y, tol, 1) == 0 ){
        printf("Error al resol L inf \n");
        return -1;

}
if (resolUsup (n, a, y, x, tol) == 0){
        printf("Error al resol U sup \n");
        return -1;
}

residu (n,a,b,x,r);
prodEsc = prod_esc(n,r,r);
prodEsc = sqrt (prodEsc);
```

tornem a trucar totes les funcions en l'ordre correcte revisant que no hi hagi cap error amb la tolerancia (si no parem el codi).

L'ordre d'execució és primer pasar de Ax = b → LUx = b per tenir el sistema

Ly = b
Ux = y

Amb aquest sistema podem aplicar les funcions anteriors per tenir la x desitjada. Com es pot observar l'ordre és : primer resoldre Ly = b per obtenir la y i després Ux = y per obtenir la x.

A l'hora de resoldre la part inferior com només necessitem la part inferior estricte li indiquem a la funció que la diagonal seran uns per evitar agafar les dades de U.

Una forma alternativa d'executar aquest codi seria aplicar el sistema de la manera següent.

$Lx = b$
$Ux = x$

(utilitzar x com vector resultat i vector incognita alhora)

Seria una alternativa més eficient ja que ens evitem la creació d'un vector adicional. En el codi no s'ha implementat per la posible necessitat de fer canvis a les funcions anteriors (segons com han sigut implementades) i per tenir un codi més llegible amb més possibilitat de detectar errors.

RESULTATS MAIN_LUSOLVER

$$A_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 3 \\ -3 & -2 & 1 & 2 \\ -4 & -3 & -2 & 1 \end{pmatrix}, b_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Resultat tolerancia 10e-3

```
INPUTS
n : 4
tol :    1.0000000e-03
Matriu a
    1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
  -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
Vector b
     1.0000000e+00    1.0000000e+00    1.0000000e+00    1.0000000e+00

 ********* LU ********

 matriu a amb forma LU
    1.0000000e+00    2.0000000e+00    3.0000000e+00    4.0000000e+00
   -2.0000000e+00    5.0000000e+00    8.0000000e+00    1.1000000e+01
   -3.0000000e+00    8.0000000e-01    3.6000000e+00    5.2000000e+00
   -4.0000000e+00    1.0000000e+00    5.5555556e-01    3.1111111e+00

 *** POST FUNCIONS ***

Vector x
    -7.1428571e-02   -7.1428571e-02   -7.1428571e-02    3.5714286e-01
Vector r
     0.0000000e+00   -2.1428571e+00   -7.5714286e-01   -2.8571429e-01
Norma 2 :    2.2905752e+00
```

Amb aquest resultat podem observar com el mínim multiplicador i el mínim valor de la diagonal obtingut de LU es 1 , llavors qualsevol tolerancia que sigui igual o menor a 1 no tindrà cap impacte al codi (no cal probar 10e-6,10e-9,10e-12).

De fet si apliquem una tolerancia de 1.0000000001 ens hauria de parar el codi mentres que una tolerancia de 1 no.

COMPROVACIONS EXTRA

Tolerancia 1.0000001

```
INPUTS
n : 4
tol :    1.0000001e+00
Matriu a
    1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
  -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
Vector b
     1.0000000e+00    1.0000000e+00    1.0000000e+00    1.0000000e+00

 ********* LU ********

Error al LU de a
```

Tolerancia 1

```
INPUTS
n : 4
tol :    1.0000000e+00
Matriu a
    1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
  -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
Vector b
     1.0000000e+00    1.0000000e+00    1.0000000e+00    1.0000000e+00

 ********* LU ********

 matriu a amb forma LU
    1.0000000e+00    2.0000000e+00    3.0000000e+00    4.0000000e+00
   -2.0000000e+00    5.0000000e+00    8.0000000e+00    1.1000000e+01
   -3.0000000e+00    8.0000000e-01    3.6000000e+00    5.2000000e+00
   -4.0000000e+00    1.0000000e+00    5.5555556e-01    3.1111111e+00

 *** POST FUNCIONS ***

Vector x
    -7.1428571e-02   -7.1428571e-02   -7.1428571e-02    3.5714286e-01
Vector r
     0.0000000e+00   -2.1428571e+00   -7.5714286e-01   -2.8571429e-01
Norma 2 :    2.2905752e+00
```

Es pot observar com efectivament la tolerancia mínima d'aquest sistema és 1

$$A_2 = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}, b_2 = \begin{pmatrix} \frac{25}{12} \\ \frac{77}{60} \\ \frac{19}{20} \\ \frac{319}{420} \end{pmatrix}$$

Resultat tolerancia 10e-3

```
INPUTS
n : 4
tol :    1.0000000e-03
Matriu a
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector b
     2.0833333e+00    1.2833333e+00    9.5000000e-01    7.5833333e-01

 ********* LU ********

 matriu a amb forma LU
    1.0000000e+00    5.0000000e-01    3.3333333e-01    2.5000000e-01
    5.0000000e-01    8.3333333e-02    8.3333333e-02    7.5000000e-02
    3.3333333e-01    1.0000000e+00    5.5555556e-03    8.3333333e-03
    2.5000000e-01    9.0000000e-01    1.5000000e+00    3.5714286e-04
Error al resol L inf
```

Resultat tolerancia 10e-6

```
INPUTS
n : 4
tol :    1.0000000e-06
Matriu a
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector b
     2.0833333e+00    1.2833333e+00    9.5000000e-01    7.5833333e-01

 ********* LU ********

 matriu a amb forma LU
    1.0000000e+00    5.0000000e-01    3.3333333e-01    2.5000000e-01
    5.0000000e-01    8.3333333e-02    8.3333333e-02    7.5000000e-02
    3.3333333e-01    1.0000000e+00    5.5555556e-03    8.3333333e-03
    2.5000000e-01    9.0000000e-01    1.5000000e+00    3.5714286e-04

 *** POST FUNCIONS ***

Vector x
     1.1666667e+00   -1.0000000e+00    6.0000000e+00   -2.3333333e+00
Vector r
     0.0000000e+00    4.5833333e-01    1.5472222e+00   -7.6325000e+00
Norma 2 :    7.8012193e+00
```

Com havíem vist abans ara els valors de la diagonal i els multiplicadors són més inferiors (ordre de 10e-4) per això una tol 10e-3 no permet executar el codi.
La resta de toleràncies no tindràn cap problema.

Part 3 Factorització PA=LU

Objectiu : resoldre el sistema de matrius Ax = b usant la descomposició PA = LU.

Cal modificar la funció LU, ara aplicant pivotatge parcial a l'hora de fer gauss.

Descomposició pAx = LUx = pb :

part triangular superior (U) : resultat d'eliminació gaussiana sense pivotatge
part triangular estrictament inferior (L) : multiplicadors de l'eliminació gaussiana
permutacions (p) : indica la posició de les files permutades

Funcions Utilitzades :

PALU

```
int palu(int n,double **a,int *p,double tol) {

int k, i, j, l , signe = -1,temp2;
double multiplicador;
double max_val, temp;


for (k = 0; k < n  ; k ++ ){


        max_val = 0.0;
        l = k;

        /* Encontrar el índice de la fila con el máximo valor en la columna k */
        for (i = k; i < n; i++) {
            if (fabs(a[i][k]) > max_val) {
                max_val = fabs(a[i][k]);
                l = i;
            }
        }

        /* Verificar si el máximo valor es menor que la tolerancia */
        if (max_val < tol) {
            return 0; /* No se puede realizar la factorización LU */
        }


        /* Intercambiar filas en la matriz y actualizar el vector de permutación p */
        if (l != k) {
            for (j = 0; j < n; j++) {
                temp = a[k][j];
                a[k][j] = a[l][j];
                a[l][j] = temp;
            }
            temp2=p[k];
            p[k] = p[l];
            p[l] = temp2;
            signe = -signe ; /* Cambiar la paridad de la permutación */
        }


        for (i = k + 1; i < n ; i ++) {

        multiplicador = a[i][k] / a[k][k];
        a[i][k] = multiplicador;

        for (j = k + 1; j < n; j ++) {

        a[i][j] -= a[k][j] * multiplicador;

}}}

return signe;
```

L'objectiu d'aquesta funció és aconseguir una matriu LU però amb una propagació d'error menor, ja que la permutem per evitar divisions amb números petits.

Tornem a aplicar algorisme anterior de calcular el multiplicador, guardar-lo i actualitzar tota la fila

($m = a_{ik} / a_{kk}$ [1])

1 : " sent i la fila inferior de k "

però ara abans de aplicar l'algorisme , fem una iteració per tota la columna per veure si tenim un valor major (en valor absolut). Si trobem un valor major, el guardem amb la seva fila corresponent (variable l).

Nota: aquest for declara i com i = k, perquè la k es per saber a quin pas de la eliminació gaussiana ens trobem, aixins no comparem amb els elements d'una fila superior

seria el segon for del codi

Revisem que la tolerancia sigui menor al número per evitar errors

seria el primer if del codi

Si l'índex del valor màxim no és l'index de la fila on s'hauria de fer Gauss.

seria el segon if

intercanviem les dues files, el vector permutació i la paritat del signe.

Ara amb la matriu ja actualitzada apliquem l'algorisme del codi anterior i fem 1 pass d'eliminació gaussiana.

Tornem a repetir per la següent filera.

El resultat d'aquesta funció seria la L i la U de a, el vector p per indicar quines permutacions hem fet i la paritat d'aquesta permutació

Main PALU

Important : Aquest main llegeix els inputs d'un fitxer exclusivament anomenat "`inputsPALU.dat`" i la seva estructura és:

```
4 1e-3

1 2 3 4
-2 1 2 3
-3 -2 1 2
-4 -3 -2 1

1 1 1 1
```

En la primera fila tenim (en ordre):

n - tamany de la matriu quadrada
tolerancia - valor considerat com 0

a continuació esta la matriu A (està representada correctament amb files i columnes)

La última fila serà el vector B

Implementació del main

La majoria del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes.

Parts importants :

CopiaA

En aquest cas ha sigut necessari guardar la matriu original de A ja que la propia a es transformarà en L i U.

```
for( i = 0; i<n ; i++ ) {
for( j = 0; j<n ; j++ ) {

copiaA[i][j] = a[i][j];

}}
```

Inicialitzar el vector p

Aquest vector té que començar amb els valors originals de A (o sigui 0,1,2,3...,n)

```
for( i = 0; i<n ; i++ ) {

        p[i] = i ;
```

Mostrar resultats de PALU

Per mostrar L i U no hi ha cap problema ja que només és imprimir A que estarà modificada. En canvi per veure PA utilitzem la seva copia i en comptes d'utilitzar el seu index, utilitzem l'índex de p

```
signe = palu(n,a,p,tol);

if (signe == 0) {

        printf ("Error al PALU \n");
        return -1;

}


printf( " matriu PA \n ");

for( i = 0; i<n ; i++ ) {
for( j = 0; j<n ; j++ ) {

        printf("%16.7e", copiaA[ p[i] ][j] );

}

printf ("\n");

}

printf( " matriu LU \n ");

for( i = 0; i<n ; i++ ) {
for( j = 0; j<n ; j++ ) {

        printf("%16.7e", a[i][j] );
```

A més, si signe = 0 indica que hem dividit per un número menor a la tolerancia. parem el codi.

PRODUCTE L * U

Per implementar el producte de L * U sense cap matriu auxiliar implica que només amb la matriu a (mitad L, mitad U) tenim que aconseguir el valor de la multiplicació.

```
printf("Producte  L * U  \n");


for (i = 0; i < n ; i++){
        for (j = 0; j < n ; j++){

                producteLU = 0;

                for ( k = 0; k < i+1 && k < j+1 ; k++) {


                        if ( i == k ) {

                        producteLU += a[k][j];

                        } else {

                        producteLU += a[i][k] * a[k][j];

                        }


                        }
                        printf(" %16.7e", producteLU );
                        }
```

Per fer això hem fet un triple bucle on la idea és:

i - seleccionar la fila
j - seleccionar la columna
k - Seleccionar quins elements es multipliquen

Això significa que després de cada j tenim que tornar a ‘0’ el valor de producteLU perquè estem a una nova posició de la matriu i que Tenim que tenir un control absolut de la k per evitar accedir a elements que no siguin de la propia L o U.

Per evitar això imposem que la k mai sigui una fila o columna superior a la actual.
dins del bucle de k el que fem es fixar la fila (i) en L i la columna (j) en U i anem avançant la k fins que la k accedeix a un valor d'una matriu no corresponent (k < i+1 && k < j +1).

També està el cas on la i==k (multiplicar dos elements de la diagonal). Això cal tenir-lo en comptes perquè la diagonal té elements de U i no de L (L tindria diagonals d'uns) per això no fem cap multiplicació, només sumem l'element de U.

Un cop fet la suma de la fila de L per la columna de U, imprimim el valor resultant i anem a la proxima posició de A

RESULTATS MAIN PALU

$$A_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 3 \\ -3 & -2 & 1 & 2 \\ -4 & -3 & -2 & 1 \end{pmatrix}, b_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

```
n : 4
tol :    1.0000000e-03
Matriu a
    1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
  -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
Vector p
  0 1 2 3

EXECUTEM LA FUNCIO PALU

 matriu PA
   -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
   1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
 matriu LU
   -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
   5.0000000e-01   2.5000000e+00   3.0000000e+00   2.5000000e+00
   7.5000000e-01   1.0000000e-01   2.2000000e+00   1.0000000e+00
  -2.5000000e-01   5.0000000e-01   4.5454545e-01   2.5454545e+00
 vector p
  3 1 2 0
 paritat permutacio 1
Producte  L * U
   -4.0000000e+00   -3.0000000e+00   -2.0000000e+00    1.0000000e+00
   -2.0000000e+00    1.0000000e+00    2.0000000e+00    3.0000000e+00
   -3.0000000e+00   -2.0000000e+00    1.0000000e+00    2.0000000e+00
    1.0000000e+00    2.0000000e+00    3.0000000e+00    4.0000000e+00
```

Es pot observar como L*U = P * A

$$A_2 = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}, b_2 = \begin{pmatrix} \frac{25}{12} \\ \frac{77}{60} \\ \frac{19}{20} \\ \frac{319}{420} \end{pmatrix}$$

```
n : 4
tol :    1.0000000e-03
Matriu a
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector p
  0 1 2 3

EXECUTEM LA FUNCIO PALU

Error al PALU
```

Ara podem observar com amb la tolerancia 10e-3 segueix causant problemes.

```
n : 4
tol :    1.0000000e-06
Matriu a
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector p
  0 1 2 3

EXECUTEM LA FUNCIO PALU

 matriu PA
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
 matriu LU
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   3.3333333e-01   8.3333333e-02   8.8888889e-02   8.3333333e-02
   5.0000000e-01   1.0000000e+00  -5.5555556e-03  -8.3333333e-03
   2.5000000e-01   9.0000000e-01  -6.0000000e-01   3.5714286e-04
 vector p
  0 2 1 3
 paritat permutacio 1
Producte  L * U
    1.0000000e+00    5.0000000e-01    3.3333333e-01    2.5000000e-01
    3.3333333e-01    2.5000000e-01    2.0000000e-01    1.6666667e-01
    5.0000000e-01    3.3333333e-01    2.5000000e-01    2.0000000e-01
    2.5000000e-01    2.0000000e-01    1.6666667e-01    1.4285714e-01
```

si apliquem un altre cop 10e-6 sí que surt el resultat

Main PALU solver

Important : Aquest main llegeix els inputs d'un fitxer exclusivament anomenat "`inputsPALUsolver.dat`" i la seva estructura és:

```
4 1e-3

1 2 3 4
-2 1 2 3
-3 -2 1 2
-4 -3 -2 1

1 1 1 1
```

En la primera fila tenim (en ordre):

n - tamany de la matriu quadrada
tolerancia - valor considerat com 0

a continuació esta la matriu A (està representada correctament amb files i columnes)

La última fila serà el vector B

Implementació del main

La majoria del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes.

Parts importants :

CopiaA i CopiaB

Tornem a necessitar una copia de les matrius originals per fer PA - LU i per fer el residu (no volem les matrius permutades).

Permutar B

```
/*
PERMUTEM B usando r como vector temporal
*/
for (i = 0; i < n; i++ ){

        r[i] = b[i];
}

for (i = 0; i < n; i++ ){

        b[i] = r[ p[i] ];
}
```

Amb un vector temporal i el resultat del vector p de PALU permutem B

Crida de funcions

```
signe = palu (n,a,p,tol);


if (resolLinf (n, a, b, y, tol, 1) == 0 ){
        printf("Error al resol L inf \n");
        return -1;

}
if (resolUsup (n, a, y, x, tol) == 0){
        printf("Error al resol U sup \n");
        return -1;
}

residu (n,copiaA,copiaB,x,r);
prodEsc = prod_esc(n,r,r);
prodEsc = sqrt (prodEsc);
```

Cridem les funcions amb l'ordre correcte per acabar amb la x resultant, el residu r amb les matrius originals.

Truquem residu amb les còpies per que a i b han sigut prèviament modificades

Ordre de la descomposició

PA = LU
Ax = b
PAx = Pb
LUx = Pb - crida de palu + permutacio b

Ly = Pb - crida resolLinf
Ux = y - crida resolUsup

Tot tenint en compte que no fem cap divisió sense sobrepasar la tolerancia donada.

NORMA INFINIT DE P*A - L*U

cal fer-ho amb una copia de A perque la a esta modificada. També, per accedir a les seves files, cal utilitzar el vector p (la matriu hauria d'estar permutada).

```
for (i = 0; i < n ; i++){

        normaInf = 0;

        for (j = 0; j < n ; j++){

                producteLU = 0;

                for ( k = 0; k < i+1 && k < j+1 ; k++) {


                        if ( i == k ) {

                        producteLU += a[k][j];

                        } else {

                        producteLU += a[i][k] * a[k][j];

                        }


                        }

                normaInf += fabs(copiaA[ p[i] ][j] - producteLU);



                printf("[%d][%d] = %16.7e |", i,j, producteLU );

                }

        if (normaInf > normaInfMax ) {

                normaInfMax = normaInf;
```

Recordar que la norma infinit és el màxim de la suma en valor absolut de cada fila

Tenim el mateix algorisme d'abans per calcular L*U.

En aquest cas ens demanen la norma infinit de PA-LU llavors un cop tenim calculat el producteLU per a una fila i columna determinada el restem amb PA i el guardem en una variable que recull la suma de tota la fila. Un cop surt del bucle de totes les columnes de una fila, el compara per veure si es el màxim valor de tota la matriu (normaInfMax comença en 0). Un cop torna a la primera columna de la següent fila torna el seu valor a 0 per tornar a guardar la suma dels n elements de la fila restats amb L*U.

RESULTAT PALU SOLVER

$$A_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ -2 & 1 & 2 & 3 \\ -3 & -2 & 1 & 2 \\ -4 & -3 & -2 & 1 \end{pmatrix}, b_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

```
n : 4
tol :    1.0000000e-03
Matriu a
   1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
  -2.0000000e+00   1.0000000e+00   2.0000000e+00   3.0000000e+00
  -3.0000000e+00  -2.0000000e+00   1.0000000e+00   2.0000000e+00
  -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
Vector b
   1.0000000e+00   1.0000000e+00   1.0000000e+00   1.0000000e+00

******* EXECUTEM PALU ******
Matriu LU
   -4.0000000e+00  -3.0000000e+00  -2.0000000e+00   1.0000000e+00
   5.0000000e-01   2.5000000e+00   3.0000000e+00   2.5000000e+00
   7.5000000e-01   1.0000000e-01   2.2000000e+00   1.0000000e+00
  -2.5000000e-01   5.0000000e-01   4.5454545e-01   2.5454545e+00
Matriu b permutada
   1.0000000e+00   1.0000000e+00   1.0000000e+00   1.0000000e+00

RESULTADOS
Vector p
  3 1 2 0
Vector x
    -7.1428571e-02   -7.1428571e-02   -7.1428571e-02    3.5714286e-01
Vector r
     0.0000000e+00    0.0000000e+00    0.0000000e+00    0.0000000e+00
 matriu PA
[0][0] =   -4.0000000e+00 |[0][1] =   -3.0000000e+00 |[0][2] =   -2.0000000e+00 |[0][3] =    1.0000000e+00 |
[1][0] =   -2.0000000e+00 |[1][1] =    1.0000000e+00 |[1][2] =    2.0000000e+00 |[1][3] =    3.0000000e+00 |
[2][0] =   -3.0000000e+00 |[2][1] =   -2.0000000e+00 |[2][2] =    1.0000000e+00 |[2][3] =    2.0000000e+00 |
[3][0] =    1.0000000e+00 |[3][1] =    2.0000000e+00 |[3][2] =    3.0000000e+00 |[3][3] =    4.0000000e+00 |
Matriz L * U :
[0][0] =   -4.0000000e+00 |[0][1] =   -3.0000000e+00 |[0][2] =   -2.0000000e+00 |[0][3] =    1.0000000e+00 |
[1][0] =   -2.0000000e+00 |[1][1] =    1.0000000e+00 |[1][2] =    2.0000000e+00 |[1][3] =    3.0000000e+00 |
[2][0] =   -3.0000000e+00 |[2][1] =   -2.0000000e+00 |[2][2] =    1.0000000e+00 |[2][3] =    2.0000000e+00 |
[3][0] =    1.0000000e+00 |[3][1] =    2.0000000e+00 |[3][2] =    3.0000000e+00 |[3][3] =    4.0000000e+00 |

Norma 2 :    0.0000000e+00

Norma Infinito :    2.2204460e-16
```

$$A_2 = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}, b_2 = \begin{pmatrix} \frac{25}{12} \\ \frac{77}{60} \\ \frac{19}{20} \\ \frac{319}{420} \end{pmatrix}$$

Tolerancia 1e-3

```
n : 4
tol :    1.0000000e-03
Matriu a
   1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector b
   2.0833333e+00   1.2833333e+00   9.5000000e-01   7.5833333e-01
Error a PALUub-installparty@installParty:~/Escritorio$
```

Tolerancia 1e-6

```
n : 4
tol :    1.0000000e-06
Matriu a
   1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   5.0000000e-01   3.3333333e-01   2.5000000e-01   2.0000000e-01
   3.3333333e-01   2.5000000e-01   2.0000000e-01   1.6666667e-01
   2.5000000e-01   2.0000000e-01   1.6666667e-01   1.4285714e-01
Vector b
   2.0833333e+00   1.2833333e+00   9.5000000e-01   7.5833333e-01

******* EXECUTEM PALU ******
Matriu LU
    1.0000000e+00   5.0000000e-01   3.3333333e-01   2.5000000e-01
   3.3333333e-01   8.3333333e-02   8.8888889e-02   8.3333333e-02
   5.0000000e-01   1.0000000e+00  -5.5555556e-03  -8.3333333e-03
   2.5000000e-01   9.0000000e-01  -6.0000000e-01   3.5714286e-04
Matriu b permutada
   2.0833333e+00   9.5000000e-01   1.2833333e+00   7.5833333e-01

RESULTADOS
Vector p
  0 2 1 3
Vector x
     1.1666667e+00   -1.0000000e+00    6.0000000e+00   -2.3333333e+00
Vector r
     0.0000000e+00   -2.2204460e-16   -2.2204460e-16   -1.1102230e-16
 matriu PA
[0][0] =    1.0000000e+00 |[0][1] =    5.0000000e-01 |[0][2] =    3.3333333e-01 |[0][3] =    2.5000000e-01 |
[1][0] =    3.3333333e-01 |[1][1] =    2.5000000e-01 |[1][2] =    2.0000000e-01 |[1][3] =    1.6666667e-01 |
[2][0] =    5.0000000e-01 |[2][1] =    3.3333333e-01 |[2][2] =    2.5000000e-01 |[2][3] =    2.0000000e-01 |
[3][0] =    2.5000000e-01 |[3][1] =    2.0000000e-01 |[3][2] =    1.6666667e-01 |[3][3] =    1.4285714e-01 |
Matriz L * U :
[0][0] =    1.0000000e+00 |[0][1] =    5.0000000e-01 |[0][2] =    3.3333333e-01 |[0][3] =    2.5000000e-01 |
[1][0] =    3.3333333e-01 |[1][1] =    2.5000000e-01 |[1][2] =    2.0000000e-01 |[1][3] =    1.6666667e-01 |
[2][0] =    5.0000000e-01 |[2][1] =    3.3333333e-01 |[2][2] =    2.5000000e-01 |[2][3] =    2.0000000e-01 |
[3][0] =    2.5000000e-01 |[3][1] =    2.0000000e-01 |[3][2] =    1.6666667e-01 |[3][3] =    1.4285714e-01 |

Norma 2 :    3.3306691e-16

Norma Infinito :    5.5511151e-17
```

Main temps PALU

Important : Aquest main guarda els resultats en un fitxer exclusivament anomenat "`resultatTemps.txt`" i la seva estructura és:

```
1    1.2325952e-32    2.2000000e-01
2    6.1629758e-32    5.2500000e-01
3    6.3519798e+00    3.7100000e-01
4    9.2214206e+00    4.5100000e-01
5    1.0333543e+01    5.6100000e-01
6    3.5983733e+01    6.2700000e-01
7    2.9173342e+01    7.2900000e-01
8    1.0191067e+01    9.2900000e-01
9    2.5575710e+01    3.4340000e+00
10    3.7985387e+01    3.2050000e+00
11    1.5403212e+01    1.5670000e+00
12    2.9650517e+01    2.3830000e+00
13    4.6490418e+01    5.5080000e+00
14    2.4955140e+01    2.5910000e+00
15    3.9581749e+01    2.8530000e+00
16    3.6325232e+01    4.9150000e+00
17    5.1391872e+01    4.8330000e+00
18    2.4453186e+01    4.8700000e+00
19    5.8254673e+01    4.7490000e+00
20    6.7792934e+01    5.2530000e+00
21    2.7122279e+02    5.6240000e+00
22    5.3355994e+01    6.5500000e+00
23    6.0968884e+01    8.5740000e+00
24    7.9141565e+01    7.9330000e+00
25    6.8343760e+01    1.0963000e+01
26    2.0671571e+02    8.8200000e+00
27    1.7734292e+02    1.0586000e+01
28    8.9932745e+01    9.9310000e+00
29    1.8513331e+02    1.0763000e+01
30    1.1225362e+02    1.0992000e+01
31    2.8171974e+02    1.3297000e+01
32    8.3864540e+01    1.4191000e+01
33    9.1447639e+01    1.5860000e+01
34    7.0812821e+01    1.6609000e+01
35    4.8293628e+02    1.5989000e+01
```

A la columna esquerra hi ha el tamany de la matriu
A la columna central hi ha la norma 2 màxima de les 1000 matrius calculades
A la columna dreta el temps de còmput mig

Implementació del main

La majoria del main són lectures de fitxers, declaracions de variables dinàmiques , alliberar memoria o prints a la terminal per comprovar que les dades introduïdes són correctes.

Parts importants :

Creació de dos bucles per fer les 100.000 matrius

```
for (m = 1 ; m < n ; m++){

        norma2Max = 0;



        start_time = clock();
        srand(time(NULL));

        for (M = 0 ; M < N ; M++){

                a = (double **) malloc (m * sizeof (double));
                copiaA = (double **) malloc (m * sizeof (double));
                b = (double *) malloc (m * sizeof (double));
                copiaB = (double *) malloc (m * sizeof (double));
                x = (double *) malloc (m * sizeof (double));
                y = (double *) malloc (m * sizeof (double));
                r = (double *) malloc (m * sizeof (double));
                p = (int *) malloc (m * sizeof (int));
```

hem de crear dos bucles per a que cada n possible (en aquest codi el tamany de les matrius serà anomenat m).

Dintre del segon bucle fem tota la inicialització de les matrius i les assignem valors random

```
for( i = 0; i<m ; i++ ) {
for( j = 0; j<m ; j++ ) {

        a[i][j] = o+(l-o)*((1.*rand())/RAND_MAX);
        copiaA[i][j] = a[i][j];

        }
        b[i] = o+(l-o)*((1.*rand())/RAND_MAX);
        copiaB[i] = b[i];

}
```

Crida de funcions

Exactament igual al apartat 3a

```
signe = palu(m,a,p,tol);

if (signe == 0 ){

        printf("Error a PALU");
        return -1;

}

if (resolLinf (m, a, b, y, tol, 1) == 0 ){
        printf("Error al resol L inf \n");
        return -1;

}
if (resolUsup (m, a, y, x, tol) == 0){
        printf("Error al resol U sup \n");
        return -1;
}

residu (m,copiaA,copiaB,x,r);
prodEsc = prod_esc(m,r,r);
norma2 = sqrt(prodEsc);
```

Selecció de la norma Major

```
if (norma2 > norma2Max ){
        norma2Max = norma2;
}
```

Dintre del 2^n bucle seleccionem quina és la major norma calculada
Abans de començar cada iteració del 2^n bucle la inicialitzem a 0.

Alliberament de memoria

```
for (i = 0; i<m ; i++) {

        free(a[i]);
        free(copiaA[i]);

}

free(a);
free(copiaA);
free(p);
free(b);
free(copiaB);
free(x);
free(y);
free(r);
```

durant tots els mains és important per liberar memoria però ara que treballem amb 100.000 matrius és molt important assegurar-se que es fa correctament al final de cada iteració del segon bucle

Guardar dades al fitxer

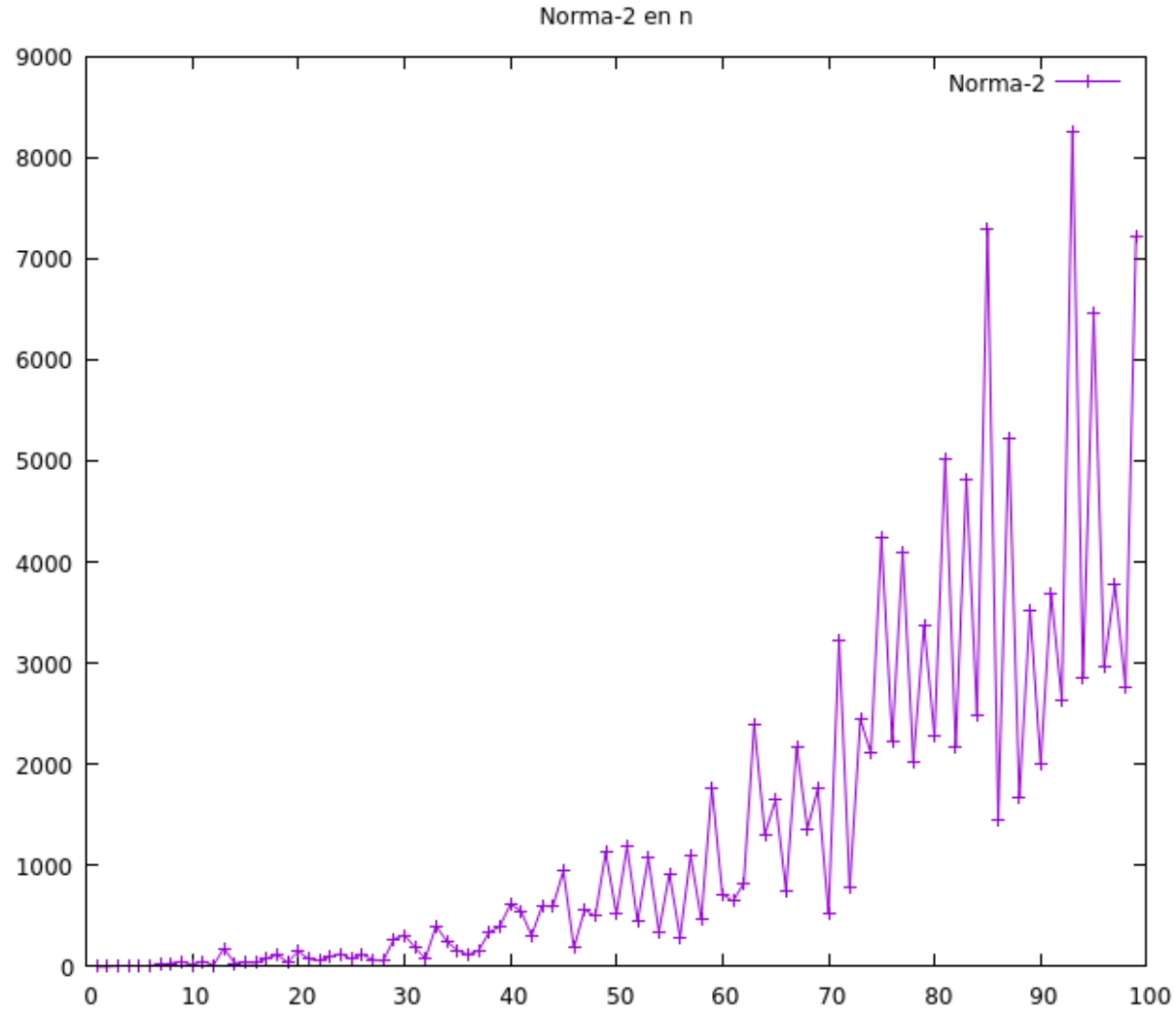
```
end_time = clock();
total_time = end_time - start_time;

fprintf(entrada, "%d %16.7e %16.7e\n", m, norma2Max, total_time );
```

al finalitzar el segon bucle parem el timer, calculem el temps total del bucle i guardem , en el format correcte, a el fitxer.

Resultats

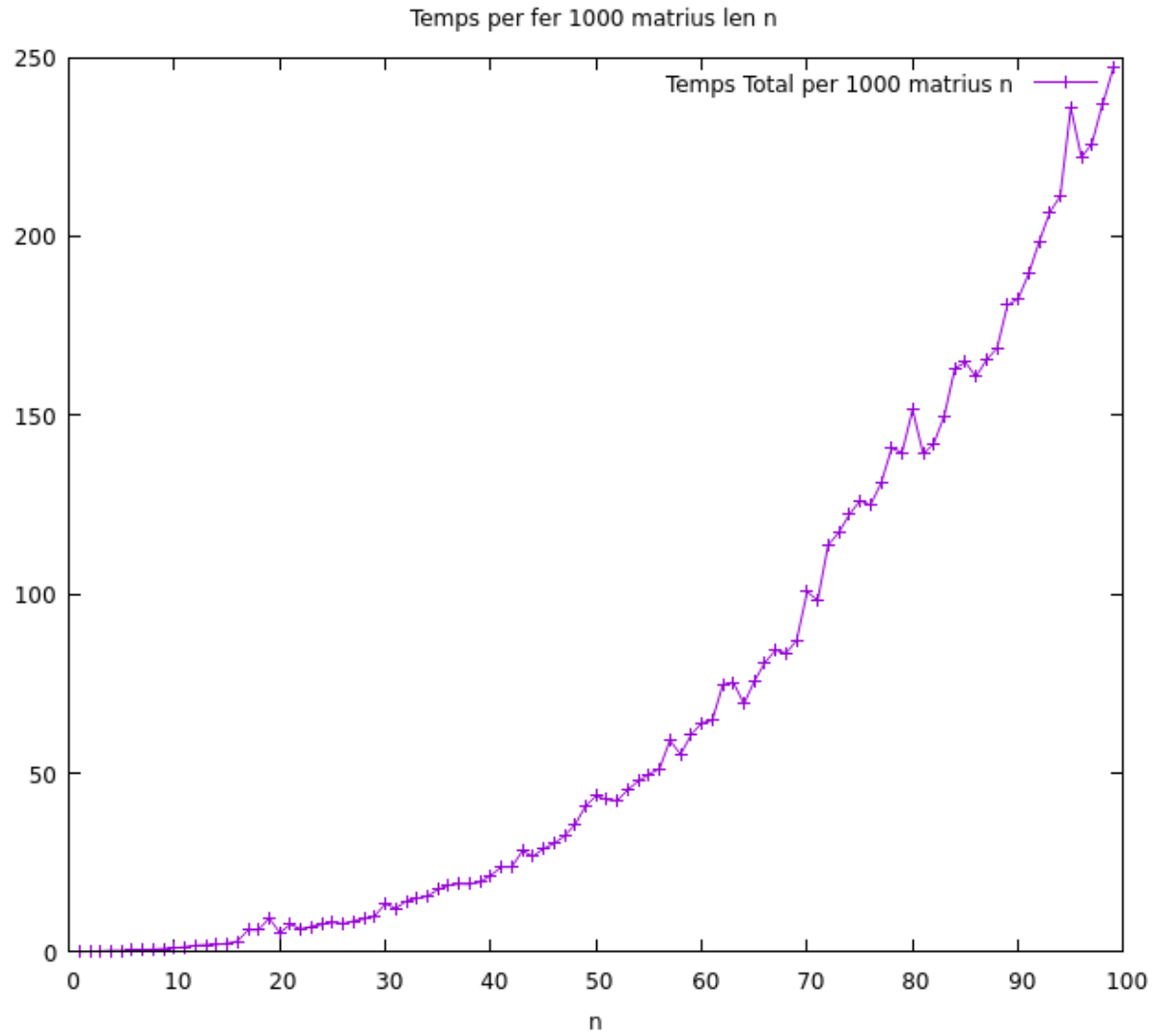
Al executar el codi acabem amb dues gràfiques diferents



En la primera gràfica

eix x - tamany de la matriu n
eix y - màxim de la norma 2

Es pot observar com segons la mida de la n va augmentant, la propagació d'errors es va fent major i major. En aquest test una matriu 90*90 arriba a tindre un error 9000 cops més gran que una matriu 10*10



En la segona gràfica

eix x - tamany de la matriu n
eix y - temps de computació per resoldre la matriu

Es pot observar com òbviament contra més gran sigui la matriu, més operacions caldrà fer i més temps trigarà en executar aquestes instruccions.

Cal destacar que no creix de forma monótona, el que implica que les optimitzacions fetes ajuden a estalviar passos computacionals segons les matrius generades.

Conclusions

És molt important controlar la precisió dels càlculs, ja que com hem vist, la propagació d'errors es un problema que a simple vista pot ser irellevant.

Un error de 10e-16 potser mai afectarà a ningú però com hem vist contra major siguin les dades i contra més precisos siguin els nombres aquests errors ja comencen a afectar a la solució del problema.

Per això és important tenir en comptes altres formes de calcular el mateix valor per evitar propagacions molt exagerades , com las variacions que hem vist.

Les gràfiques han sigut d'ajuda per visualitzar els resultats donats però hagués estat bé comparar-les amb gràfiques amb funcions no optimitzades, per veure quina diferència estem guanyant.

Andrés Río
UB - ICC
profe Arturo Vieiro
Entrega 27 - 10 - 2023