Laboratorio 2: Sistemas Lineales

El objetivo de este laboratorio es utilizar eficientemente métodos directos e indirectos para la solución de sistemas lineales de forma $A\overrightarrow{x} = \overrightarrow{b}$.

A continuación se comprobará que $A\backslash b$ es equivalente a calcular la descomposición LU de A con pivote parcial. Si A es estrictamente diagonal dominante, no será necesario realizar permutaciones de filas al calcular su descomposición LU con pivote parcial, es decir, $P = I$.

1. Haga una *function* que genere una matriz tridiagonal de orden n de la siguiente forma:

$$A = \begin{pmatrix} a & b & 0 & \cdots & 0 \\ c & a & b & \cdots & 0 \\ 0 & c & a & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & b \\ 0 & 0 & \cdots & c & a \end{pmatrix}.$$

   Los datos de entrada deben ser a, b, c y n, donde n corresponde a la dimensión de la matriz $n\,x\,n$.

2. Mediante el comando *rank* (devuelve el rango de una matriz), determine si la matriz tridiagonal y simétrica A correspondiente a $n = 10, a = 4, b = c = 1$ es invertible. Note que con los valores de a, b y c, la matriz A es estrictamente diagonal dominante.

3. Resuelva el sistema $A\overrightarrow{x} = \overrightarrow{b}$ con la matriz tridiagonal anterior y una matriz $\overrightarrow{b}$ cualquiera con cualquiera de los siguientes modos:

```
x=A\b
```

```
[L,U,P]=lu(a);
x=U\(L\b)
```

   Compruebe que $P = I$ y $A = LU$.

4. Compare las soluciones obtenidas y calcule los residuos $r = b - Ax$ respectivos.

1. Haga una *function* que genere una matriz A con el comando *rand* de tamaño 10.

2. Con el comando *rand* genere un vector $b \,\epsilon\, \Re^{10}$

3. Mediante el comando *rank*, determine si A es invertible. Si no lo es, genere una nueva matriz con *rand* hasta obtener una matriz invertible.

4. Resuelva el sistema $A\overrightarrow{x} = \overrightarrow{b}$ con la matriz y el vector generados anteriormente con cualquiera de los siguientes métodos:

```
x=A\b
```

```
[L,U,P]=lu(a);
x=U\(L\(P*b))
```

   Compruebe que $PA = LU$.

5. Compare las soluciones obtenidas y calcule los residuos $r = b - Ax$ respectivos.

En muchas aplicaciones es necesario resolver varios sistemas de ecuaciones $Ax_i = b_i$, con la misma matriz $A\,\epsilon\,\Re^{nxn}$ y distintos segundos miembros b_i, $i = 1, \ldots m$. Para hacer esto en *Matlab* resulta conveniente generar la matriz de segundos miembros

$$B = \left[\begin{array}{c|c|c} b_1 & \cdots & b_m \end{array}\right] \epsilon\,\Re^{nxm}$$

y resolver el sistema matricial $A\overrightarrow{x} = \overrightarrow{b}$, cuya solución

$$X = \left[\begin{array}{c|c|c} x_1 & \cdots & x_m \end{array}\right] \epsilon\,\Re^{nxm}$$

es la matriz de vectores solución x_1, $i = 1, \ldots, m$, de los sistemas anteriores.
El siguiente programa de *Matlab* tiene por objetivo verificar esto experimentalmente:

```
A=rand(50);
%Comprobar si es invertible
while rank(A)~=50
    A=rand(50);
end
B=rand(50,100);

tic
X=A\B
toc
tic
for i=1:100
    Y(:,i)=A\B(:,i);
end
toc

dif=norm(X-Y,inf)
```

1. Escriba y ejecute el programa anterior.
2. Analice lo que hace este programa y justifique la diferencia de tiempos de ejecución.
3. Utilice el comando de matlab $X = A\backslash B$ con una matriz B adecuada para calcular A^{-1}. Compare el resultado obtenido con el del comando $inv(A)$.

El objetivo de este ejercicio es demostrar que **no es conveniente** resolver un sistema de ecuaciones mediante la inversa de la matriz del mismo.
Considere el siguiente programa:

```
nn=[100:100:500];
t1=[];
t2=[];
for n=nn
    A=rand(n);
    y=ones(n,1);
    t0=cputime;
    x=inv(A)*y;
    t1=[t1 cputime-t0];
    t0=cputime;
```

```
    x=A\y;
    t2=[t2 cputime-t0];
end
plot(nn,t1,'*-',nn,t2,'o-')
```

Analice este programa y verifique cuál método es más conveniente.

1. Método de Jacobi

Considere el siguiente programa que resuelve mediante el método de **Jacobi** un sistema de ecuaciones $A\overrightarrow{x} = \overrightarrow{b}$ con error menor que una tolerancia dada **tol**. Parta de una semilla inicial x^0 nulo y utilice el criterio de detención estudiado en las clases teóricas. El programa también debe detenerse si se alcanza un número máximo de iteraciones *maxit* sin que se satisfaga el criterio de convergencia.

Pruebe el método con una tolerancia de $tol = 10^{-6}$. Verifique que la solución obtenida aproxima a la solución exacta con tolerancia del error estipulado. Determine la cantidad de iteraciones realizadas.

```
% Deben definir A (matriz diagonal dominante) y b del ejercicio 1
% deben ingresar una variable tol=
% deben ingresar el numero maximo de iteraciones maxit=
n=length(A);
N=diag(diag(A)); % Jacobi
%N=tril(A); % Gauss-Seidel comentar N de Jacobi
P=N-A;
x=zeros(n,1);
corr=1;
errest=1;
iter=1;

while errest>tol & iter<maxit
    iter=iter+1;
    x0=x;
    corr0=corr;
    x=N\(P*x0+b);
    corr=norm(x-x0,inf);
    normest=corr/corr0;
    if normest>=1
        error('norma de la matriz de iteracion > 1')
    end
    errest=normest/(1-normest)*corr;
end
disp('numero de iteraciones')
iter
```

Observación: El código presentado es específico para la matriz A del ejemplo 1 de esta guía. No sirve para cualquier matriz.

Ejercicio: Repita lo anterior con el método de **Gauss-Seidel**. Indique cuál de los métodos requiere menos iteraciones.