Ejercicio 3.1 Escribir una M-función `function [x] = Bajada(A, b)` para calcular la solución `x` del sistema $Ax = b$, siendo A una matriz cuadrada triangular inferior.

Algoritmo de bajada
$n =$ dimensión de A
Para cada $i = 1, 2, \ldots n$,

$$x_i = \frac{1}{A_{ii}}\Big(b_i - \sum_{j=1}^{i-1} A_{ij}x_j\Big)$$

Fin

```
function [x] = Bajada(A, b)
%
%  Bajada(A, b) es la solucion del sistema lineal de
%       matriz triangular inferior Ax = b
%
%------ tolerancia para el test de singularidad
tol = 1.e-10;
%------ inicializaciones
n = length(b);
x = zeros(n,1);
for i = 1:n
    Aii = A(i,i);
    if  abs(Aii) < tol
        warning(' La matriz A es singular ')
    end
    suma = 0;                         % en version vectorial, sin usar for,
    for j = 1:i-1                     % se puede calcular la suma con:
        suma = suma + A(i,j)*x(j);    %
    end                               % suma = A(i,1:i-1)*x(1:i-1)
        x(i) = (b(i) - suma)/Aii;
end
```

Para comprobar el funcionamiento del programa, construir una matriz 20×20 (por ejemplo) y un vector columna b de números generados aleatoriamente (con la función `rand` o bien con `randi`) y luego extraer su parte triangular inferior con la función `tril`. (Consultar en el help de MATLAB la utilización de estas funciones).

Ejercicio 3.2 Escribir una M-función `function [x] = Subida(A, b)` para calcular la solución `x` del sistema $Ax = b$, siendo A una matriz cuadrada triangular superior.

Algoritmo de subida
$n =$ dimensión de A
Para cada $i = n, \ldots, 2, 1$

$$x_i = \frac{1}{A_{ii}}\Big(b_i - \sum_{j=i+1}^{n} A_{ij}x_j\Big)$$

Fin

Para comprobar el funcionamiento del programa, construir una matriz A y un vector b de números generados aleatoriamente (como en el ejercicio anterior) y luego extraer su parte triangular inferior con la función `triu`.

Ejercicio 3.3 Escribir una M-función `function [x, res] = LU(A, b)` que calcule la solución `x` del sistema $Ax = b$ y el residuo `res`$=Ax - b$ siguiendo los pasos siguientes:
— Calcular la factorización LU mediante la función `lu` de MATLAB
— Calcular la solución del sistema $Lv = Pb$ mediante la M-función `Bajada`
— Calcular la solución del sistema $Ux = v$ mediante la M-función `Subida`

Utilizar la M-función `LU` para calcular la solución del sistema

$$\begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 2 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Ejercicio 3.4 Las matrices de Hilbert definidas por $H_{ij} = \dfrac{1}{i+j-1}$ son un ejemplo notable de matrices mal condicionadas, incluso para dimensión pequeña.

Utilizar la M-función `LU` para resolver el sistema $Hx = b$, donde H es la matriz de Hilbert de dimensión $n = 15$ (por ejemplo) y b es un vector de números generados aleatoriamente. Comprobar que el residuo es grande. Comprobar que la matriz H está mal condicionada calculando su número de condición.

La función `hilb(n)` construye la matriz de Hilbert de dimensión n.

Sugerencia para “fans”: escribir una M-función `function [H] = mhilbert(n)` que construya la matriz de Hilbert de dimensión n.

Ejercicio 3.5 Escribir una M-función `function [x, res] = CHOL(A, b)` que calcule la solución `x` del sistema con matriz simétrica definida positiva $Ax = b$ y el residuo `res`$=Ax - b$ siguiendo los pasos siguientes:
— Calcular la matriz L de la factorización de Cholesky LL^t de A mediante la función `chol` de MATLAB
— Calcular la solución del sistema $Lv = b$ mediante la M-función `Bajada`
— Calcular la solución del sistema $L^t x = v$ mediante la M-función `Subida`

Para comprobar el funcionamiento del programa, se puede generar alguna de las matrices definida positivas siguientes:
— `M = gallery('moler', n)`
— `T = gallery('toeppd', n)`
— `P = pascal(n)`

Ejercicio 3.6 Escribir una M-función para resolver un sistema por el método de Jacobi:
`function [u, flag] = MIJacobi(A, b, u0, tol, Itermax)`

Argumentos:

`A` la matriz del sistema
`b` el segundo miembro
`u0` el punto para comenzar las iteraciones
`tol` tolerancia para detener las iteraciones
`Itermax` número máximo de iteraciones
`u` la solución aproximada
`flag` si el método converge, es el número de iteraciones realizadas; si no converge, `flag`=0.

```
function [u, flag] = MIJacobi(A, b, u0, tol, Itermax)
%
%  u = MIJacobi(A, b, u0, tol, Itermax) devuelve la solucion del sistema
%         lineal Au=b, calculada mediante el metodo iterativo de Jacobi,
%         comenzando las iteraciones con el vector u0.
%         Se detienen las iteraciones cuando ||u^{k+1}-u^k}|| < tol
%         o bien cuando se alcanza al numero maximo de iter. Itermax
%  [u, flag] = MIJacobi(A, b, u0, tol, Itermax) devuelve, ademas, un
%         indicador del desarrollo del algoritmo:
%         flag = 0 si el algoritmo no converge en el numero maximo
%                de iteraciones Itermax fijado
%         flag = k > 0 si el algoritmo converge en k iteraciones
%

D    = diag(A);
u    = u0;
flag = 0;

for k = 1:Itermax
    v = (b-A*u)./D;
    u = u + v;
    if norm(v) < tol
        flag = k;
        return
    end
end
```

Comprobar el funcionamiento del programa con los siguientes sistemas:

(a) $A = \begin{pmatrix} 5 & -2 & 1 \\ -1 & -7 & 3 \\ 2 & -1 & 8 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix}$

(b) $A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 19 \\ 19 \\ -3 \\ -12 \end{pmatrix}$

(c) El sistema $Au = b$ cuya matriz ampliada está almacenada en el fichero `sistema1.txt`. La matriz ampliada se puede cargar en memoria leyendo el fichero con la orden

```
A = load('sistema1.txt')
```

(d) Lo mismo, con el fichero `sistema2.txt`

Ejercicio 3.7 Modificar adecuadamente el programa anterior de forma que se visualice la evolución de las iteraciones: en cada una de ellas imprimir el número de la iteración y el valor de $\|u^{k+1} - u^k\|$.

Utilizar este programa para resolver el sistema $Hu = b$ donde H es la matriz de Hilbert de dimensión 4 (ya definida en el Ejercicio 3.4), y b es alguno de los siguientes

$$b^1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad b^2 = \begin{pmatrix} 0.905 \\ 1 \\ 1.2 \\ 1.01 \end{pmatrix}, \quad b^3 = \begin{pmatrix} 1 \\ 0.9 \\ 1 \\ 1.1 \end{pmatrix}.$$

Comprobar que la matriz H está mal condicionada y que el método iterado no converge, o lo hace muy lentamente. Comparar los resultados con los obtenidos utilizando el operador de división matricial por la izquierda `\`. Obsérvese el efecto que producen pequeñas perturbaciones del segundo miembro en la solución del sistema.

Ejercicio 3.8 En la implementación del método de Gauss-Seidel se puede utilizar un único vector u para almacenar las sucesivas iteraciones: para cada $i = 1, \ldots, n$, se calcula la nueva componente u_i^{k+1} y se almacena "machacando" la anterior.

Partiendo del programa `MIJacobi`, escribir una M-función

```
function [u, flag] = MIGaussSeidel(A, b, u0, tol, Itermax)
```

Utilizar este programa para resolver los sistemas anteriores.