3 Resolución de sistemas lineales

3.1 Métodos directos: resumen de resultados teóricos

- **Definición:** Se dice que las matrices L (triangular inferior) y U (triangular superior) forman una factorización LU de una matriz A si se tiene $A = LU$.

- **Observaciones:**
  Si se impone la condición adicional de que la matriz L tenga todos sus elementos diagonales iguales a 1, entonces esta factorización se llama también de **Doolittle**.
  Si, por el contrario, se impone que sea la matriz U la que tenga sus elementos diagonales iguales a 1, entonces la factorización se llama de **Crout**.

- **Proposición:** Si $\det(A) \neq 0$, entonces la factorización de Doolittle de A (LU con unos en la diagonal de L) si existe, es única.

- **Teorema de Doolittle:** Si los menores principales de una matriz cuadrada A son no nulos, entonces A admite una factorización LU. Esta factorización es unica si se impone la condición de que los elementos diagonales de L sean todos iguales a uno.

- **Observaciones:**
  — Si A es no singular y no satisface las hipótesis del Teorema de Doolittle, puede suceder que sí se satisfagan dichas condiciones efectuando permutaciones sobre las filas de la matriz.
  — También puede suceder que una matriz A singular admita factorización LU (que no será única).
  — La factorización LU no es más que la expresión matricial del método de eliminación de Gauss.
  — Cuando se aplica una estrategia de pivot durante el proceso de eliminación, es decir, se permutan a conveniencia determinadas filas de la matriz para reducir los errores de redondeo, lo que se obtiene es la factorización LU de una matriz PA obtenida de A al aplicarle las permutaciones mencionadas. En este caso se tiene $PA = LU$, donde P es una matriz de permutaciones.

- **Definición:** Se dice que una matriz cuadrada A de dimensión n es de diagonal estrictamente dominante si se verifica

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad \forall i = 1, \ldots, n$$

- **Teorema:** Sea A una matriz cuadrada de diagonal estrictamente dominante. Entonces se tiene:
  — A es regular.
  — A admite una única factorización LU de Doolittle (es decir, con la diagonal de L formada por unos).

- **Definición:** Se dice que una matriz real cuadrada A de dimensión n es definida positiva si se verifica
$$\sum_{i,j=1}^{n} a_{ij}\, u_i\, u_j > 0, \quad \forall u \in \mathbb{R}^n \setminus \{0\}$$

- **Proposición:** Si A es una matriz real definida positiva entonces
  — A es regular y $\det(A) > 0$.
  — Las submatrices principales de A son todas definidas positivas.

- Como consecuencia de lo anterior, si una matriz real A es definida positiva, entonces A admite una factorización LU (que será única si se impone la condición adicional de que la diagonal de L esté formada por unos).

- Si A es una matriz simétrica, las siguientes son condiciones equivalentes:
  — A es definida positiva.
  — Todos los autovalores de A son positivos.
  — Todos los menores principales de A son positivos.

- **Teorema de factorización de Cholesky:** Si A es una matriz simétrica definida positiva, entonces existe una matriz triangular inferior L tal que $A = L\,L^t$. Si se impone la condición adicional de que los elementos diagonales de L sean todos positivos, entonces la factorización es única.

- La factorización de Cholesky también se puede obtener mediante una matriz triangular superior, y entonces se tiene $U^t\,U = A$.

3.2 Métodos directos con MATLAB

3.2.1 Los operadores de división matricial

Sea A una matriz cualquiera y sea B otra matriz con el mismo número de filas que A[1]. Entonces, la "solución" del "sistema" (en realidad un sistema lineal por cada columna de B)

$$A\,X = B$$

se calcula en MATLAB mediante el operador no estándar "\" denominado *backward slash*, (barra inversa en español)

```
X = A \ B
X = mldivide(A, B)     (equivalente a lo anterior)
```

Hay que pensar en él como el operador de "división matricial por la izquierda".

De forma similar, si C es una matriz con el mismo número de columnas que A, entonces la solución del sistema

$$X\,A = C$$

se obtiene en MATLAB mediante el operador"/" ("división matricial por la derecha")

```
X = C / A
X = mrdivide(C, A)     (equivalente a lo anterior)
```

[1] A menos que A sea un escalar, en cuyo caso `A\B` es la operación de división elemento a elemento, es decir `A./B`.

Estos operadores se aplican incluso si A no es una matriz cuadrada, es decir si no hay el mismo número de ecuaciones que de incógnitas.

Lo que se obtiene de estas operaciones es, lógicamente, distinto según sea el caso.

De forma resumida, si A es una matriz cuadrada y b es un vector columna, `x = A\B` es la solución del sistema lineal `Ax=b`, obtenida por diferentes algoritmos, en función de las características de la matriz A (diagonal, triangular, simétrica, etc.). Si A es singular o mal condicionada, se obtendrá un mensaje de alerta (*warning*).

Para una descripción completa del funcionamiento de estos operadores, así como de la elección del algoritmo aplicado, véase la documentación de MATLAB correspondiente, tecleando en la ventana de comandos

```
doc mldivide
```

Ejemplo 3.1 (Uso del operador \)
Calcular la solución del sistema (compatible determinado)

$$\begin{cases} 2x_1 + x_2 - x_3 &= -1 \\ 2x_1 - x_2 + 3x_3 &= -2 \\ 3x_1 - 2x_2 &= 1 \end{cases}$$

```
>> A = [2, 1, -1; 2, -1, 3; 3, -2, 0];
>> b = [-1; -2; 1];
>> x = A\b
x =
   -0.3636
   -1.0455
   -0.7727
```

Como comprobación calculamos el resíduo que, como es de esperar, no es exactamente nulo, debido a los errores de redondeo:

```
>> A*x - b
ans =
   1.0e-15 *
   -0.4441
         0
         0
```

Ejemplo 3.2 (Uso del operador \)
Calcular la solución del sistema (compatible indeterminado)

$$\begin{cases} x_1 + x_2 + x_3 &= 1 \\ 2x_1 - x_2 + x_3 &= 2 \\ x_1 - 2x_2 &= 1 \end{cases}$$

```
>> A = [1, 1, 1; 2, -1, 1; 1, -2, 0];
>> b = [ 1; 2; 1];
>> x = A\b
Warning: Matrix is singular to working precision.
x =
   NaN
   NaN
   NaN
```

Ejemplo 3.3 (Uso del operador \)
Calcular la solución del sistema (incompatible)

$$\begin{cases} 2x + 2y + t &= 1 \\ 2x - 2y + z &= -2 \\ x - z + t &= 0 \\ -4x + 4y - 2z &= 1 \end{cases}$$

```
>> A = [2, 2, 0, 1; 2, -2, 1, 0; 1, 0, -1, 1; -4, 4, -2, 0];
>> b = [ 1; -2; 0; 1];
>> x = A\b
Warning: Matrix is singular to working precision.
x =
   NaN
   NaN
  -Inf
  -Inf
```

3.2.2 Determinante. ¿Cómo decidir si una matriz es singular?

El determinante de una matriz cuadrada A se calcula con la orden

```
det(A)
```

Este cálculo se hace a partir de la factorización LU de la matriz A, ya que se tiene $\det(L) = 1$, y $\det(A) = \det(U)$, que es el producto de sus elementos diagonales.

El uso de `det(A) == 0` para testar si la matriz A es singular sólo es aconsejable para matrices de pequeño tamaño y elementos enteros también de pequeña magnitud.

El uso de `abs(det(A)) < epsilon` tampoco es recomendable ya que es muy difícil elegir el `epsilon` adecuado (véase el Ejemplo 3.4).

Lo aconsejable para testar la singularidad de una matriz es usar su **número de condición** `cond(A)`.

Ejemplo 3.4 (Matriz no singular, con determinante muy pequeño, bien condicionada)
Se considera la matriz 10×10 siguiente, que no es singular, ya que es múltiplo de la identidad,

```
>> A = 0.0001 * eye(10);
>> det(A)
ans =
   1.0000e-40
```

Vemos que su determinante es muy pequeño. De hecho, el test `abs(det(A)) < epsilon` etiquetaría esta matriz como singular, a menos que se eligiera `epsilon` extremadamente pequeño. Sin embargo, esta matriz no está mal condicionada:

```
>> cond(A)
ans =
     1
```

Ejemplo 3.5 (Matriz singular, con `det(A)` muy grande)
Se considera la matriz 13×13 construida como sigue, que es singular y de diagonal dominante:

```
>> A = diag([24, 46, 64, 78, 88, 94, 96, 94, 88, 78, 64, 46, 24]);
>> S = diag([-13, -24, -33, -40, -45, -48, -49, -48, -45, -40, -33, -24], 1);
>> A = A + S + rot90(S,2);
```

La matriz que se obtiene es

```
    24   -13     0     0     0     0     0     0     0     0     0     0     0
   -24    46   -24     0     0     0     0     0     0     0     0     0     0
     0   -33    64   -33     0     0     0     0     0     0     0     0     0
     0     0   -40    78   -40     0     0     0     0     0     0     0     0
     0     0     0   -45    88   -45     0     0     0     0     0     0     0
     0     0     0     0   -48    94   -48     0     0     0     0     0     0
     0     0     0     0     0   -49    96   -49     0     0     0     0     0
     0     0     0     0     0     0   -48    94   -48     0     0     0     0
     0     0     0     0     0     0     0   -45    88   -45     0     0     0
     0     0     0     0     0     0     0     0   -40    78   -40     0     0
     0     0     0     0     0     0     0     0     0   -33    64   -33     0
     0     0     0     0     0     0     0     0     0     0   -24    46   -24
     0     0     0     0     0     0     0     0     0     0     0   -13    24
```

A es singular (la suma de todas sus filas da el vector nulo). Sin embargo, el cálculo de su determinante con la función `det` da

```
>> det(A)
ans =
   1.0597e+05
```

cuando debería dar como resultado cero! Esta (enorme) falta de precisión es debida a los errores de redondeo que se cometen en la implementación del método LU, que es el que MATLAB usa para calcular el determinante. De hecho, vemos que el número de condición de A es muy grande:

```
>> cond(A)
ans =
   2.5703e+16
```

3.2.3 La factorización LU

La factorización LU de una matriz se calcula con MATLAB con la orden

```
[L, U, P] = lu(A)
```

El significado de los distintos argumentos es el siguiente:

`L` es una matriz triangular inferior con unos en la diagonal.
`U` es una matriz triangular superior.
`P` es una matriz de permutaciones, que refleja los intercambios de filas realizados durante el proceso de eliminación gaussiana sobre las filas de la matriz A, al aplicar la técina del pivot.
`LU=PA` es la factorización obtenida.

Entonces, para calcular la solución del sistema lineal de ecuaciones

$$A\,x = b$$

utilizando la factorización anterior sólo hay que plantear el sistema, equivalente al anterior,

$$P\,A\,x = P\,b \iff L\,U\,x = P\,b \iff \begin{cases} L\,v = P\,b \\ U\,x = v \end{cases}$$

El primero de estos dos sistemas se resuelve fácilmente mediante un algoritmo de bajada, ya que la matriz del mismo es triangular inferior. Una vez calculada su solución, v, se calcula x como la solución del sistema $U\,x = v$, que se puede calcular mediante un algoritmo de subida, al ser su matriz triangular superior.

3.2.4 Ejercicios sobre la factorización LU

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.

3.2.5 La factorización de Cholesky

La factorización de Cholesky de una matriz simétrica se calcula en MATLAB con alguna de las órdenes

```
U = chol(A)              % se obtiene una matriz triang. superior
L = chol(A, 'lower')     % se obtiene una matriz triang. inferior
```

de manera que se tiene $U^t U = A$ con la primera opción y $L L^t = A$ con la segunda.

Cuando se usa en la forma `chol(A)`, la función `chol` sólo usa la parte triangular superior de la matriz A para sus cálculos, y asume que la parte inferior es la simétrica. Es decir, que si se le pasa una matriz A que no sea simétrica no se obtendrá ningún error. Por el contrario, si se usa en la forma `chol(A,'lower')`, sólo se usará la parte triangular inferior de A, asumiendo que la parte superior es la simétrica.

La matriz A tiene que ser definida positiva. Si no lo es, se obtendrá un error.

Una vez calculada la matriz L, para calcular la solución del sistema lineal de ecuaciones

$$A\,x = b$$

utilizando la factorización anterior sólo hay que plantear el sistema, equivalente al anterior,

$$A\,x = b \iff L\,L^t\,x = b \iff \begin{cases} L\,v = b \\ L^t\,x = v \end{cases}$$

que, de nuevo, se resuelven fácilmente utilizando sendos algoritmos de bajada + subida.

Si, por el contrario, se calcula la factorización en la forma $U^t U = A$, entonces se tiene

$$A\,x = b \iff U^t\,U x = b \iff \begin{cases} U^t\,v = b \\ U\,x = v \end{cases}$$

que se resuelve aplicando en primer lugar el algoritmo de subida y a continuación el de bajada.

3.2.6 Ejercicios sobre la factorización de Cholesky

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)`

3.3 Métodos iterativos: resumen de resultados teóricos

Sea $u \in \mathbb{R}^n$ la solución (única) del sistema lineal $Au = b$, $A \in \mathbb{R}^{n\times n}$ y $b \in \mathbb{R}^n$

- **Definición:** Un **método iterativo** para resolver el sistema lineal $Au = b$ es una iteración de la forma

$$\text{(MI)} \quad \begin{cases} u_0 \in \mathbb{R}^n \text{ arbitrario} \\ u^{k+1} = Bu^k + c, \quad k \geq 0 \end{cases}$$

  donde $c \in \mathbb{R}^n$ y B es una matriz $n \times n$ verificando que $I - B$ es invertible y que el sistema de ecuaciones $u = Bu + c$ (i.e. $(I - B)u = c$) es equivalente a $Au = b$.

- El método (MI) se dice que es **convergente** si

$$\lim_{k\to+\infty} e^k = \lim_{k\to+\infty} (u^k - u) = 0$$

- **Teorema:** Las siguientes proposiciones son equivalentes

  (a) El método (MI) es convergente.

  (b) $\rho(B) < 1$.

  (c) Existe una norma matricial subordinada para la cual se tiene $\|B\| < 1$.

- El método (MI) converge más rápido cuanto más pequeño sea $\rho(B)$.

- Los métodos iterativos que se ven aquí provienen de descomponer la matriz A en la forma $A = M - N$ (de manera diferente en cada caso) siendo M una matriz regular y *fácil de invertir*, es decir, tal que el sistema $Mx = d$ es fácil de resolver. Entonces

$$Au = b \iff Mu = Nu + b \iff u = (M^{-1}N)u + (M^{-1}b), \quad \text{i.e.} \begin{cases} B & = M^{-1}N \\ c & = M^{-1}b \end{cases}$$

  y, de $A = M - N$, se tiene $M^{-1}A = I - M^{-1}N = I - B$, luego $I - B$ es invertible. Del Teorema anterior se tiene que el método iterativo será convergente sii $\rho(M^{-1}N) < 1$.

- Suponiendo que se tiene $a_{ii} \neq 0$ $\forall i = 1, \ldots, n$, se considera la siguiente descomposición de:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = D - E - F, \quad \text{con} \quad D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix},$$

$$E = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{pmatrix}, \quad F = \begin{pmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ 0 & 0 & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

- **Método de Jacobi:** Se eligen: $M = D$ y $N = E + F$. Las iteraciones a realizar son:

$$\begin{cases} u^0 \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} \text{ solución del s.l. } Du^{k+1} = b + (E + F)u^k \text{ (matriz diagonal)} \end{cases}$$

  que se pueden escribir también, de forma desarrollada:

$$(\text{MJ})\begin{cases} u^0 = \left(u_1^0, u_2^0, \ldots, u_n^0\right) \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} = \left(u_1^{k+1}, u_2^{k+1}, \ldots, u_n^{k+1}\right) \text{ con} \\ u_i^{k+1} = \dfrac{1}{a_{ii}}\Big[b_i - \sum\limits_{j=1}^{i-1} a_{ij}u_j^k - \sum\limits_{j=i+1}^{n} a_{ij}u_j^k\Big] = u_i^k + \dfrac{1}{a_{ii}}\big[b_i - Au^k\big], \quad i = 1, \ldots, n \end{cases}$$

- **Método de Gauss-Seidel:** Se basa en la idea intuitiva de "aprovechar las componentes" de u^{k+1} recientemente calculadas para calcular la siguiente. Para ello se eligen: $M = D - E$ y $N = F$. Las iteraciones a realizar son:

$$\begin{cases} u^0 \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} \text{ solución del s.l. } (D-E)u^{k+1} = b + Fu^k \text{ (matriz triangular inferior)} \end{cases}$$

que se pueden escribir también:

$$(\text{MGS})\begin{cases} u^0 = \left(u_1^0, u_2^0, \ldots, u_n^0\right) \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} = \left(u_1^{k+1}, u_2^{k+1}, \ldots, u_n^{k+1}\right) \text{ con} \\ u_i^{k+1} = \dfrac{1}{a_{ii}}\Big[b_i - \sum\limits_{j=1}^{i-1} a_{ij}u_j^{k+1} - \sum\limits_{j=i+1}^{n} a_{ij}u_j^k\Big], \quad i = 1, \ldots, n \end{cases}$$

- **Método de relajación:** Dado $\omega \in \mathbb{R}$ (que, de hecho, tendrá que ser $\omega \in [0, 2]$), se eligen

$$M = \frac{1}{\omega}D - E, \qquad N = \left(\frac{1-\omega}{\omega}\right)D + F$$

Las iteraciones a realizar con esta elección son:

$$\begin{cases} u^0 \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} \text{ solución del s.l.} \\ \left(\dfrac{1}{\omega}D - E\right)u^{k+1} = b + \left(\left(\dfrac{1-\omega}{\omega}\right)D + F\right)u^k \text{ (matriz triangular inferior)} \end{cases}$$

que se pueden escribir también:

$$(\text{MR})_\omega\begin{cases} u^0 = \left(u_1^0, u_2^0, \ldots, u_n^0\right) \in \mathbb{R}^n \text{ arbitrario,} \\ u^{k+1} = \left(u_1^{k+1}, u_2^{k+1}, \ldots, u_n^{k+1}\right) \text{ con} \\ u_i^{k+1} = u_i^k + \dfrac{\omega}{a_{ii}}\Big[b_i - \sum\limits_{j=1}^{i-1} a_{ij}u_j^{k+1} - \sum\limits_{j=i}^{n} a_{ij}u_j^k\Big], \quad i = 1, \ldots, n \end{cases}$$

(El método $(\text{MR})_\omega$ para $\omega = 1$ coincide con el método de Gauss-Seidel).

- **Resultados de convergencia:**
  - Para que $(\text{MR})_\omega$ converja es condición necesaria que $0 < \omega < 2$.
  - Si A es hermítica y definida positiva, entonces el método de relajación converge si $0 < \omega < 2$ (en particular, el método de Gauss-Seidel converge).

3.3.1 Ejercicios sobre métodos iterativos

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)` para implementar el método iterativo de Gauss-Seidel.

Utilizar este programa para resolver los sistemas anteriores.

Ejercicio 3.9 Partiendo del programa `MIGaussSeidel`, escribir una M-función
`function [u, flag] = MIRelajacion(A, b, u0, w, tol, Itermax)` para implementar el método de relajacion.

Utilizar este programa para resolver los sistemas anteriores.