Análisis Numérico Matricial. Curso 2022-2023.
Grado en Matemáticas. Universidad de Murcia.
Práctica computacional 4. (22/marzo/2023)

- **No utilizar caracteres latinos, tildes o $\tilde{n}$ ni usar espacios en blanco en los nombres de los archivos, ni en los de los directorios**.
- **No utilizar caracteres latinos, tildes o $\tilde{n}$ ni usar espacios en blanco en los nombres en los comentarios dentro de los códigos.**
- En Octave (no es necesario en MATLAB) entrar en preferencias y cambiar el símbolo # por el símbolo % para escribir comentarios. Esto garantiza compatibilidad con Matlab.
- El uso indiscriminado de las órdenes $.*$, $./$ o $\hat{.}$ en casos donde no sean necesarios se penalizará como error ya que indican su desconocimiento y forma de uso.
- Guardar figuras con formato *.pdf y nombre del ejercicio en cuestión. Por ejemplo **Ejercicio4Figura1.pdf**.
- Crear una carpeta que tenga como nombre **ApellidoNombrePractica4** donde guardar todos los archivos que tengan relación con esta práctica, los principales y las funciones. No usar la estructura de biblioteca para entregar los códigos.
- Comprimir estar carpeta en formato *.zip para generar un archivo de nombre **ApellidoNombrePractica4.zip** para subirlo a la tarea de la práctica.

Mostramos algunos ejemplos de códigos y planteamos ejercicios computacionales.

Factorizacion A=LU

1. *a*) Escribe una función `solveU(U,b)` que reciba como dato de entrada una matriz triangular superior $U \in \mathcal{K}_n$ y un vector columna $b \in \mathcal{K}_{n,1}$ que devuelva el vector columna x solución del sistema $Ux = b$, utilizando el método ascendente.

   b) Repite la construcción para obtener una función `solveL(L,b)` relativa a sistemas triangulares inferiores utilizando el método descendente.

   c) Comprueba su funcionamiento con los siguientes sistemas:

   1) $Ux = b$, con $n = 4, 5, 6$, A=rand(n,n), U=triu(A), b=rand(n,1) (observa el funcionamiento de triu(A))
   2) $Lx = b$ L=tril(A) (observa el funcionamiento de tril(A)).

   Una vez se tiene ambas funciones, la función **solveLU(L,U,b)** obtiene la solución del sistema lineal $LUx = b$

```
function X = solveLU (L, U, b)
 [m,n]=size(L);
 [p,c]=size(U);
 [q,r]=size(b);
 %Numero de columnas de L debe ser igual al numero de filas de U
  if p~=n
    error('Dimensiones incompatibles de L y U.');
  end
  %Numero de columnas de L debe ser igual al numero de filas de b
  if q~=n
    error('Dimensiones incompatibles de L y b.');
  end
  X = solveU(U, solveL(L,b));
end
```

Algoritmo 2.1 **Método Ascendente (flujo en Octave/MatLab).**

Datos de entrada:
U (Matriz triangular superior de los coeficientes del sistema, sin ceros en la diagonal);
B (vector o matriz -término independiente.);
Variables:
n (dimensión de U y número de filas de B)
c (número de columnas de B) // Ver que las dimensiones de A y B son compatibles.
x; // un vector o matriz con el mismo número de columnas que B.
Fujo del programa:

```
%% Resolvemos el sistema por el método ascendente.
  x(n,:)=b(n,:)/U(n,n);
  for k=n-1:-1:1
```
$//fila\_x_k = (fila\_B_k - \sum_{j=k+1}^{n}(U_{k,j}.*fila\_x_j))/U_{k,k}$
```
  x(k,:)=x(k,:)=(B(k,:)-U(k,k+1:n)*x(k+1:n,:))/U(k,k);
  end
```

Datos de salida: Solución x del sistema $Ux = B$.

Algoritmo 2.2 **Método Descendente (flujo en Octave/MatLab).**

Datos de entrada:
L (Matriz triangular inferior de los coeficientes del sistema, sin ceros en la diagonal);
B (vector o matriz -término independiente.);
Variables:
n (dimensión de L y número de filas de B)
c (número de columnas de B) // Ver que las dimensiones de A y B son compatibles.
x; // un vector o matriz con el mismo número de columnas que B.
Fujo del programa:

```
%% Resolvemos el sistema por el método descendente.
  x(1,:)=B(1,:)/L(1,1);
  for k=2:n
```
$//fila\_x_k = (fila\_B_k - \sum_{j=1}^{k-1}(L_{k,j}.*fila\_x_j))/L_{k,k}$
```
  x(k,:)=x(k,:)=(B(k,:)-L(k,1:k-1)*x(1:k-1,:))/L(k,k);
  end
```

Datos de salida: Solución x del sistema $Lx = B$.

2. La función **LUDoolittle.m** que se muestra intenta obtener la factorización LU de una matriz A sin hacer permutaciones.

```
function [L,U]=LUDoolittle(A)
  [m,n]=size(A);
  if m ~= n
    error('dimensiones incompatibles');
  end
 %
  for j=1:n-1
    piv=A(j,j);
    if (abs( piv) <1e-10 )
      warning('Gauss sin permutaciones: elemento nulo en diagonal');
    end
    for i=j+1:n
      A(i,j)=A(i,j)/piv;
      A(i,j+1:n)=A(i,j+1:n)-(A(i,j)*A(j,j+1:n));
    end
  end
  L=tril(A,-1)+eye(n);
  U=triu(A);
end
```

Ayudandote de esta función construye la función:

solveLU(A ,b)

que resuelva un sistema de la forma $Ax = b$, descomponiendo $A = LU$ donde L es triangular inferior y U es triangular superior en la forma correspondiente al algoritmo de factorizacion LU de Doolittle. Debe devolver las dos matrices y la solución en un vector [L,U,x].

a) Utiliza las funciones rand(), triu() y tri() para construir matrices aleatorias con factoriazación LU y comprueba con ellas el funcionamiento de tu función.

b) Compara el funcionamiento con las funciones internas de octave [L,U]=lu(A) y [L,U,P]=lu(A).

c) Calcula el determinante de A usando esta función y compáralo con la función interna de Octave.

Método de factorización LU (Dootlittle)

Datos de entrada:
A (Matriz de coeficientes del sistema.);
Variables: n (dimensión de A)
Aux (Matriz axiliar para almacenar los cálculos)
L; U// matrices triangulares superiores e inferiores a devolver como datos de salida.
Fujo del programa:

```
 % Comprobar que A es una matriz cuadrada
[m,n]=size(A);
if (m  = n)
   error('matriz no cuadrada');
end
 % Constuir L y U usando la matriz Aux
Aux=zeros(n,n)
 % Vamos a ir rellenando Aux por filas y columnas de forma alternada
 % Primera fila:
Aux(1,1)=A(1,1); % L(1,1)=1 y U(1,1)=aux(1,1)
if abs(Aux(1,1)) < 100*eps
   error('cero en diagonal de U');
end
Aux(1,2:n)=A(1,2:n); % Coincide con la de U
 % Primera columna:
Aux(2:n,1)=A(2:n,1)/Auz(1,1); % Coincide con la de L
for k=2:n
   Aux(k,k)=A(k,k)-Aux(k,1:k-1)*Aux(1:k-1,k); % U(k,k)=Aux(k,k)
   if abs(Aux(k,k)) < 100*eps
      error('cero en diagonal de U');
   end
    % fila k desde columna k+1 (para U), y columna k desde fila k+1 (para L):
   for r=k+1:n
      Aux(k,r)=A(k,r)-Aux(k,1:k-1)*Aux(1:k-1,r); % U(k,r)=Aux(k,r)
      Aux(r,k)=(A(r,k)-Aux(r,1:k-1)*Aux(1:k-1,k))/Aux(k,k); % L(r,r)=Aux(k,k)
   end
end
U=triu(Aux); % U(i,j)=Aux(i,j) si i≤j
L=tril(Aux,-1)+eye(n); % L(r,r)=1 y L(i,j)=Aux(i,j) si i<j
```

Datos de salida:L y U (Factorización LU)o mensaje de error

3. Aplicar la resolución mediante la factorización LU a la matriz

$$A = \begin{pmatrix} 1 & 1+\frac{1}{2}10^{-15} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{pmatrix}$$

y obtener la matriz residuo $A - LU$.

4. Sea L una matriz triangular inferior con 1 en la diagonal. Escribe una función de Octave/MATLAB que proporcione T matriz triangular inferior con 1 en diagonal tal que $T^2 = L$.

5. Construye una función

```
function [L,D,R]=LDR(A)
```

que admita como entrada una matriz A, y caso de existir devuelva la factorización única $A = LDR$ donde L es una matriz triangular con unos en la diagonal, D es una matriz diagonal y R es triangular superior con unos en la diagonal. La respuesta de la función serán las tres matrices: `L`, `D` y `R`. Comprueba el funcionamiento con una matriz diagonal estrictamente dominante.

6. Construye una función

```
function S=symmetricMat(n))
```

que tenga como entrada un entero positivo n y devuelva una matriz aleatoria simétrica S de dimensión n. Puedes usar la función rand(n,n) que devuelve una matriz aleatoria, luego quedarte con la parte triangular inferior con tril() y acaba sumando esta con su traspuesta.

7. Construye una función

```
function SPD=spdMat(n))
```

que tenga como entrada un entero positivo n y devuelva una matriz aleatoria simétrica definida positiva SPD de dimensión n. Con el siguiente código puedes crear la matriz simétrica definida positiva

```
% codigo para generar una matriz simetrica definida positiva
% desde una simétrica A
A=SymmetricMat(n);
[P,D]=eig(A)
D=abs(D)
D=D+norm(D)*eye(size(D))
SPD= P*D*P'
```

8. Construye una función

```
function [L,x]= solveCholeski(A,b)
```

que tenga como entrada una matriz simétrica definida positiva A y un vetor b y devuelva una matriz triangular inferior L que da la factorización de Choleski $A = L\,L^t$ junto con la solución del sistema lineal $Ax = b$. Si A no es definida positiva lo comprobaremos en la construcción de L y mandaremos un mensaje de error.

9. Una matriz cuadrada A de dimensión n se dice que es una **matriz banda** cuando existen enteros p y q tales que $a_{ij} = 0$ si $i+p \leq j$ o $j+q \leq i$. El ancho de banda de este tipo se define como $w = p+q-1$. Las matrices banda que más suelen aparecer en la práctica tienen la forma $p = q = 2$ y $p = q = 4$. Las matrices de ancho de banda 3 con $p = q = 2$ se llaman **matrices tridiagonales** porque su forma es

$$A = \begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{pmatrix}.$$

Si se define la sucesión

$$\delta_0 = 1,\ \delta_1 = b_1,\ \delta_k = b_k\delta_{k-1} - a_k c_{k-1}\delta_{k-2} \quad k = 2, ..., n \tag{1}$$

entonces, $\delta_k = det(A_k)$ (A_k el menor principal de orden k). Observar que si se pone

$$\gamma_k = \frac{\delta_k}{\delta_{k-1}} \quad k = 1, ..., n$$

La recurrencia para δ_k se reescribe en términos de γ_k como

$$\gamma_1 = b_1, \quad \gamma_k = b_k - a_k c_{k-1}\gamma_{k-1}^{-1} \quad k = 2, ..., n. \tag{2}$$

La factorización LU de la matriz A es

$$A = LU = \begin{pmatrix} 1 & & & & \\ a_2\gamma_1^{-1} & 1 & & & \\ & \ddots & \ddots & & \\ & & a_{n-1}\gamma_{n-2}^{-1} & 1 & \\ & & & a_n\gamma_{n-1}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \gamma_1 & c_1 & & & \\ & \gamma_2 & c_2 & & \\ & & \ddots & \ddots & \\ & & & \gamma_{n-1} & c_{n-1} \\ & & & & \gamma_n \end{pmatrix}$$

Cuando A es una matriz tridiagonal simétrica definida positiva esta factorización coincide también con la factorización de Choleski $A = GG^t$. Este algoritmo se conoce como el algoritmo de Thomas, introducido por Llewellyn Thomas en 1949.

Dada las tres diagonales de una matriz tridiagonal construir dos funciones

[aL,bU]=thomasLUfact1(a,b,c) y **[aL,bU]=thomasLUfact2(a,b,c)**

que devuelvan la diagonal inferior **aL** de L y la diagonal principal **bU** de U para la descomposición $A = LU$. Usar (1) en la función **thomasLUfact1(a,b,c)** y (2) en la función **thomasLUfact2(a,b,c)**.

Explicar razonadamente porqué hay que desechar la version generada por (1).

10. Usando el algoritmo de Thomas y los algoritmos de subida y bajada resolver el siguiente sistema con $n = 1000$ y comparar con la obvia solución exacta:

$$\begin{array}{ccccccl} x_1 & + & 0.5x_2 & & & = & 1.5 \\ 0.5x_{i-1} & + & x_i & + & 0.5x_{i+1} & = & 2 \quad (2 \leq i \leq n-1) \\ & & 0.5x_{n-1} & + & x_n & = & 1.5 \end{array}$$

11. Usando el algoritmo de Thomas y los algoritmos de subida y bajada resolver el siguiente sistema con $n = 1000$

$$\begin{array}{ccccccl} 4x_1 & - & x_2 & & & = & -20 \\ x_{i-1} & - & 4x_j & + & x_{j+1} & = & 40 \quad (2 \leq i \leq n-1) \\ & & -x_{n-1} & + & 4x_n & = & -20 \end{array}$$