Análisis Númerico: Taller 2

23 de enero del 2025

Universidad Nacional de Colombia

Jorge Mauricio Ruiz Vera

Andrés David Cadena Simons acadenas@unal.edu.co
Sandra Natalia Florez Garcia sflorezga@unal.edu.co

Problema 1:

Sea $A \in \mathbb{R}^{m\times n}$. Entonces se satisface:

1. $\|A\|_2 = \|A^T\|_2 \leq \|A\|_F = \|A^T\|_F$,
2. $\|A\|_\infty \leq \sqrt{n}\|A\|_2$,
3. $\|A\|_2 \leq \sqrt{m}\|A\|_\infty$,
4. $\|A\|_2 \leq \sqrt{\|A\|_1\|A\|_\infty}$.

Solución:

1. $\|A\|_2 = \|A^T\|_2 \leq \|A\|_F = \|A^T\|_F$.
   Veamos primero que $\|A\|_2 = \|A^T\|_2$, para esto recordemos que:

$$\|A\|_2 = \sqrt{\lambda_{máx}(A^TA)}$$

   Por lo que sabemos que $\|A\|_2$ depende totalmente de sus valores propios, por lo que será suficiente para concluir el resultado con ver que los valores propios de A^TA son los mismos valores propios de $(A^T)^TA^T = AA^T$, para esto será suficiente con verificar que si tomamos $\lambda \neq 0$ valor propio de A^TA, entonces:

$$\begin{aligned} A^TAx &= \lambda x \\ AA^TAx &= \lambda Ax \\ AA^Ty &= \lambda y \end{aligned}$$

   Por lo que podríamos decir que λ también es un valor propio de AA^T, note que de forma análoga se puede repetir el mismo procedimiento para ver que los valores propios de AA^T son valores propios de A^TA y por ende estas 2 matrices comparten valores propios distintos de 0, luego se puede concluir que $\sqrt{\lambda_{máx}(A^TA)} = \sqrt{\lambda_{máx}(AA^T)}$, es decir, $\|A\|_2 = \|A^T\|_2$. Ahora, veamos que $\|A\|_F = \|A^T\|_F$, para esto recordemos que:

$$\|A\|_F = \left(\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2\right)^{\frac{1}{2}}$$

Problema 1 continúa en la siguiente página...

por lo que si tomamos $a_{ij} \in A$ y $\tilde{a}_{ji} \in A^T$, debería ser fácil seguir el siguiente cálculo:

$$\begin{aligned}
\|A\|_F &= \left(\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2 \right)^{\frac{1}{2}} \\
&= \left(\sum_{i=1}^{n} \sum_{j=1}^{m} |\tilde{a}_{ji}|^2 \right)^{\frac{1}{2}} \\
&= \left(\sum_{j=1}^{m} \sum_{i=1}^{n} |\tilde{a}_{ji}|^2 \right)^{\frac{1}{2}} \\
&= \|A^T\|_F
\end{aligned}$$

Ahora veamos que $\|A\|_2 \leq \|A\|_F$.
Primero note que si tomamos la matriz $A^T A \in R^{n \times n}$ se cumple que:

$$[A^T A]_{ij} = \sum_{k=1}^{m} a_{ki} a_{kj}$$

Luego:

$$\begin{aligned}
tr(A^T A) &= \sum_{i=1}^{n} [A^T A]_{ii} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ji} a_{ji} \\
&= \|A\|_F
\end{aligned}$$

además sabemos que $tr(A^T A) = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2$ y que $\|A\|_2^2 = \sigma_1^2$, por lo que podemos asegurar que:

$$\begin{aligned}
\|A\|_2 &\leq \sigma_1^2 \\
&\leq \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2 \\
&\leq \|A\|_F^2
\end{aligned}$$

lo que implica que $\|A\|_2 \leq \|A\|_F$, lo que concluye el ejercicio.

♥

2. $\|A\|_\infty \leq \sqrt{n}\|A\|_2$.
Note que:

$$\|A\|_\infty = \max_{i=1,\cdots,m} \sum_{j=1}^{n} |a_{ij}|$$

Ahora, suponga $A_i = (a_{i1}, a_{i2}, \cdots, a_{in}) \in \mathbb{R}^n$, note que:

$$\begin{aligned}\sum_{j=1}^{n} |a_{ij}| &= (|a_{i1}|, |a_{i2}|, \cdots, |a_{in}|) \cdot (1, 1, \cdots, 1) \qquad \text{Luego por Cauchy-Schwarz.} \\ &\leq \|(|a_{i1}|, |a_{i2}|, \cdots, |a_{in}|)\|_2 \|(1, 1, \cdots, 1)\|_2 \\ &\leq \sqrt{\sum_{j=1}^{n} |a_{ij}|^2} \times \sqrt{\sum_{j=1}^{n} |1|^2} \\ &\leq \sqrt{n} \|A_i\|_2\end{aligned}$$

Ahora, como $\|A_i\|_2 \leq \|A\|_2$ para todo $i = 1, \cdots, m$ al ser consistente, al tomar máximo en ambos lados de la desigualdad se cumple que:

$$\begin{aligned}\|A\|_1 &\leq \max_{i=1,\cdots,m} \sum_{j=1}^{n} |a_{ij}| \\ &\leq \sqrt{n} \|A\|_2\end{aligned}$$

lo que concluye el resultado esperado.

♥

3. $\|A\|_2 \leq \sqrt{m} \|A\|_\infty$.
Sea $x \in \mathbb{R}^n$ tal que $\|x\|_2 = 1$, y denotemos:

$$(Ax)_i = \sum_{j=1}^{n} a_{ij} x_j$$

entonces:

$$\begin{aligned}|(Ax)_i| &= \left| \sum_{j=1}^{n} a_{ij} x_j \right| \\ &\leq \sum_{j=1}^{n} |a_{ij}||x_j| \\ &\leq \sqrt{\sum_{j=1}^{n} |a_{ij}|^2} \|x\|_2 \qquad \text{Por Cauchy-Schawrz.} \\ &\leq \sqrt{\sum_{j=1}^{n} |a_{ij}|^2}\end{aligned}$$

luego como:

$$\begin{aligned}\|Ax\|_2^2 &= \sum_{i=1}^{m} |(Ax)_i|^2 \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}^2| \\ &\leq \sum_{i=1}^{m} \|A\|_\infty^2 \\ &\leq m\|A\|_\infty^2\end{aligned}$$

luego tomando raiz cuadrada en ambos lados y el máximo de los $\|Ax\|_2$ con x de norma 1, se puede concluir que $\|A\|_2 \leq \sqrt{m}\|A\|_\infty$, lo que concluye el resultado esperado.

4. $\|A\|_2\sqrt{\|A\|_1\|A\|_\infty}$.
Sea $x \in \mathbb{R}^n$ con $\|x\| = 1$ arbitrario, entonces:

$$(Ax)_i = \sum_{j=1}^{n} a_i j x_j$$

Y utilizando la definición de $\|x\|_1$ tenemos que:

$$\begin{aligned}\|Ax\|_1 &= \sum_{i=1}^{m} |(Ax)_i| \\ &= \sum_{i=1}^{m} \left|\sum_{j=1}^{n} a_{ij}x_j\right|\end{aligned}$$

usando el procedimiento que usamos en el primer ejercicio sabemos que:

$$\begin{aligned}\|Ax\|_2^2 &\leq \sum_{i=1}^{m} \left|\sum_{j=1}^{n} a_{ij}x_j\right|^2 \\ &\leq \sum_{i=1}^{m} \left|\sum_{j=1}^{n} a_{ij}x_j\right| \max_{i=1,\cdots,m} \left|\sum_{j=1}^{n} a_{ij}x_j\right| \\ &\leq \|Ax\|_1\|Ax\|_\infty \\ &\leq \|A\|_1\|A\|_\infty\end{aligned}$$

Luego tomando raíz y luego tomando el máximo variando los $\|x\| = 1$ se concluye que:

$$\|A\|_2 \leq \sqrt{\|A\|_\infty\|A\|_1}$$

Lo que finaliza el ejercicio.

Problema 2:

Sea $\|\cdot\|$ una norma en $\mathbb{R}^n$ y A una matriz invertible de tamaño $n \times n$. Pruebe que:
Si $Ax = b$, $(A + \delta A)(x + \delta x) = b + \delta b$ y $\|A^{-1}\|\|\delta A\| < 1$, entonces $A + \delta A$ es invertible y se cumple que:

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\text{cond}(A)}{1 - \|A^{-1}\|\|\delta A\|}\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right)$$

Solución:

Por hipótesis, $A \in \mathbb{R}^{n\times n}$ es invertible, por lo que $-A^{-1}\delta A \in \mathbb{R}^{n\times n}$. Como la norma matricial inducida por una norma en $\mathbb{R}^n$ es submultiplicativa y $\|A^{-1}\|\|\delta A\| < 1$, se tiene que:

$$\| - A^{-1}\delta A\| \leq |-1|\|A^{-1}\|\|\delta A\| = \|A^{-1}\|\|\delta A\| < 1.$$

Entonces, por la Serie de Neumann, $I - (-A^{-1}\delta A)$ es invertible. Por consiguiente, dado que es producto de matrices invertibles, $A + \delta A = A(I - (-A^{-1}\delta A))$ también es invertible.

Además, sabemos que:

$$(A + \delta A)(x + \delta x) = b + \delta b.$$

Expandiendo,

$$Ax + A\delta x + \delta Ax + \delta A\delta x = b + \delta b.$$

Como $Ax = b$ y A es invertible, se tiene:

$$A\delta x = \delta b - \delta Ax - \delta A\delta x,$$

y despejando δx,

$$\delta x = A^{-1}(\delta b - \delta Ax - \delta A\delta x).$$

Aplicando la norma y usando propiedades de la norma inducida:

$$\|\delta x\| \leq \|A^{-1}\|\|\delta b - \delta Ax - \delta A\delta x\|.$$

Esto implica:

$$\|\delta x\| \leq \|A^{-1}\|(\|\delta b\| + \|\delta A\delta x\| + \|\delta Ax\|),$$

y expandiendo,

$$\|\delta x\| \leq \|A^{-1}\|\|\delta b\| + \|A^{-1}\|\|\delta A\|\|\delta x\| + \|A^{-1}\|\|\delta A\|\|x\|.$$

Por lo tanto,

$$\|\delta x\| - \|A^{-1}\|\|\delta A\|\|\delta x\| \leq \|A^{-1}\|(\|\delta A\|\|x\| + \|\delta b\|),$$

lo que resulta en:

$$(1 - \|A^{-1}\|\|\delta A\|)\|\delta x\| \leq \|A^{-1}\|(\|\delta A\|\|x\| + \|\delta b\|).$$

Problema 2 continúa en la siguiente página...

De aquí,

$$\|\delta x\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\|\|\delta A\|}(\|\delta A\|\|x\| + \|\delta b\|). \tag{1}$$

Además, como $Ax = b$, se cumple:

$$\|b\| \leq \|A\|\|x\| \implies \frac{\|A\|}{\|b\|} \geq \frac{1}{\|x\|}. \tag{2}$$

Finalmente, aplicando (2) en (1), obtenemos:

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\|\|\delta A\|}\left(\frac{\|\delta A\|\|x\|}{\|x\|} + \frac{\|\delta b\|}{\|x\|}\right).$$

Simplificando,

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\|\|\delta A\|}\left(\|\delta A\| + \frac{\|\delta b\|\|A\|}{\|b\|}\right).$$

Finalmente,

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|\|A\|}{1 - \|A^{-1}\|\|\delta A\|}\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right),$$

lo que demuestra la desigualdad deseada.

Problema 3:

Sea

$$A = \begin{pmatrix} a & a \\ a & a+\delta \end{pmatrix}, \qquad a > 0 \text{ fijo, } \delta > 0 \text{ variable.}$$

1. Obtenga el número de condición de A. Para los valores de δ muy pequeños o muy grandes, ¿podemos afirmar que el sistema $Ax = b$ esta mal condicionado? Justifique su respuesta.
2. ¿Existe algún valor de δ que haga óptimo el número de condición de A?¿Cuál es este número de condición?

Solución:

1. Obtenga el número de condición de A. Para los valores de δ muy pequeños o muy grandes, ¿podemos afirmar que el sistema $Ax = b$ esta mal condicionado? Justifique su respuesta.
   Recordemos que el número de condición de una matriz A es:

$$K_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$$

   Así, calculemos estas normas:

$$\|A\|_\infty = 2a + \delta$$
$$\|A^{-1}\|_\infty = \frac{2a+\delta}{a\delta}$$

   Luego:

$$\begin{aligned} K_\infty(A) &= (2a+\delta)\left(\frac{2a+\delta}{a\delta}\right) \\ &= \frac{4a^2 + 4a\delta + \delta^2}{a\delta} \\ &= \frac{4a}{\delta} + 4 + \frac{\delta}{a} \end{aligned}$$

   Ahora, para ver la tendencia del valor de condición para números muy pequeños o muy grandes revisaremos que sucede cuando $\delta \to \infty$ y cuando $\delta \to 0$:

$$\begin{aligned} \lim_{\delta\to 0} K_\infty(A) &= \lim_{\delta\to 0} \frac{4a}{\delta} + 4 + \frac{\delta}{a} \\ &= \infty \\ \lim_{\delta\to \infty} K_\infty(A) &= \lim_{\delta\to \infty} \frac{4a}{\delta} + 4 + \frac{\delta}{a} \\ &= \infty \end{aligned}$$

   Luego vemos que en general para números muy pequeños o muy grandes $K_\infty(A)$ está muy lejana a 1, por lo que la matriz A estaría mal condicionada y por ende el sistema

Problema 3 continúa en la siguiente página...

$Ax = b$ esta mal condicionado, ya que la desigualdad:

$$\frac{\|x - \tilde{x}\|_\infty}{\|x\|_\infty} \leq K_\infty(A)\frac{\|r\|_\infty}{\|b\|_\infty}$$

no tendría una cota uniforme para valores pequeños o grandes de δ y por ende podríamos esperar que las soluciones aproximadas se comporten mal.

2. ¿Existe algún valor de δ que haga óptimo el número de condición de A?¿Cuál es este número de condición?
Para responder la pregunta es necesario estudiar el comportamiento del número de condición como una función de δ, esto es $K_\infty(A)(\delta) = \frac{4a}{\delta} + 4 + \frac{\delta}{a}$.
Primero hallemos los puntos críticos de esta función, para esto usemos la derivada:

$$K_\infty(A)'(\delta) = -\frac{4a}{\delta^2} + \frac{1}{a}$$

igualando a 0 para encontrar puntos críticos:

$$\begin{aligned}
-\frac{4a}{\delta^2} + \frac{1}{a} = 0 && \text{Lo que implica}\\
\frac{1}{a} = \frac{4a}{\delta^2} && \text{Luego}\\
\delta^2 = 4a^2 && \text{así}\\
\delta^2 - 4a^2 = 0 && \text{lo que implica}\\
(\delta - 2a)(\delta + 2a) = 0 &&
\end{aligned}$$

de lo que podemos concluir que, como $\delta > 0$, el único punto crítico es cuando $\delta = 2a$, luego como la tendencia de la función es ∞ cuando δ va para 0 o ∞, entonces podemos asegurar que este es un mínimo, siendo así, calculemos el número de condición suponiendo $\delta = 2a$.

$$\begin{aligned}
K_\infty(A) &= \frac{4a}{2a} + 4 + \frac{2a}{a}\\
&= 2 + 4 + 2\\
&= 8
\end{aligned}$$

luego el valor de δ que hace óptimo el número de condición de A es $2a$, que estaría aún así mal condicionando el problema (al ser mayor que 1).

Problema 4:

aproximar una función continua $f : [0, 1] \longrightarrow \mathbb{R}$ mediante un polinomio $p(t) = a_n t^n + \cdots + a_1 t + a_0$, el error de aproximación E se mide en la norma L^2, es decir:

$$E^2 := \|p - f\|_{L^2}^2 = \int_0^1 [p(t) - f(t)]^2 dt$$

Solución:

a) Muestre que la minimización del error $E = E(a_0, a_1, \cdots, a_n)$ conduce a un sistema de ecuaciones lineales $H_n a = b$, donde:

$$b = [b_0, \ldots, b_n]^T \in \mathbb{R}^{n+1}, \quad b_i = \int_0^1 f(t) t^i \, dt, \qquad i = 0, 1, \ldots, n,$$

H_n es la matriz de Hilbert de orden n, definida como:

$$(H_n)_{i,j} = \frac{1}{i + j + 1}, \qquad i, j = 0, \cdots, n,$$

y a es el vector de coeficientes de p.

Nuestro objetivo es minimizar el error en términos de los coeficientes de $p(t)$. Para ello, simplificamos la expresión de E^2, de manera que pueda escribirse en función de $a_0, \ldots, a_n$.

$$\begin{aligned} E^2 &:= \int_0^1 [p(t) - f(t)]^2 \, dt \\ &= \int_0^1 \left(p(t)^2 - 2p(t)f(t) + f(t)^2 \right) \, dt \end{aligned}$$

Dado que $f(t)^2$ no depende de a_i para $i = 0, \cdots, n$, este término no afecta la minimización y, por lo tanto, no se considera. Así, tenemos:

$$E^2(a_0, a_1, \cdots, a_n) = \int_0^1 p(t)^2 \, dt - 2 \int_0^1 p(t) f(t) \, dt$$

Como $p(t)^2 = \left(\sum_{i=0}^n a_i t^i \right)^2 = \sum_{i=0}^n \sum_{j=0}^n a_i a_j t^{i+j}$, entonces:

$$\int_0^1 p(t)^2 \, dt = \sum_{i=0}^n \sum_{j=0}^n a_i a_j \int_0^1 t^{i+j} \, dt = \sum_{i=0}^n \sum_{j=0}^n a_i a_j \frac{1}{i + j + 1}$$

Además:

$$\int_0^1 p(t) f(t) \, dt = \sum_{i=0}^n a_i \int_0^1 t^i f(t) \, dt = \sum_{i=0}^n a_i b_i, \qquad \text{con } b_i = \int_0^1 t^i f(t) \, dt$$

Problema 4 continúa en la siguiente página...

Sustituyendo las integrales anteriores, obtenemos:

$$E^2(a_0, a_1, \cdots, a_n) = \sum_{i=0}^{n}\sum_{j=0}^{n} a_i a_j \frac{1}{i+j+1} - 2\sum_{i=0}^{n} a_i b_i$$

Para minimizar E, hallamos los puntos críticos resolviendo el sistema $\nabla E^2 = \vec{0}$, lo que implica resolver:

$$\frac{\partial}{\partial a_k}\left(\sum_{i=0}^{n}\sum_{j=0}^{n} a_i a_j \frac{1}{i+j+1} - 2\sum_{i=0}^{n} a_i b_i\right) = 0, \quad k = 0, \cdots, n$$

Calculando las derivadas parciales:

$$2\sum_{j=0}^{n} a_j \frac{1}{k+j+1} - 2b_k = 0$$

Finalmente, el sistema de ecuaciones lineales resultante es:

$$\sum_{j=0}^{n} a_j \frac{1}{k+j+1} = b_k, \quad \text{con } k = 0, \cdots, n$$

De forma matricial, este sistema se escribe como $H_n a = b$:

$$\begin{bmatrix} \frac{1}{1} & \frac{1}{2} & \cdots & \frac{1}{n+1} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n+1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}.$$

b) Muestre que H_n es simétrica y definida positiva. Primero, demostremos que H_n es simétrica. Por definición, los elementos de H_n están dados por:

$$(H_n)_{i,j} = \frac{1}{i+j+1} = (H_n)_{j,i}, \qquad i, j = 0, \cdots, n.$$

Por lo tanto, H_n es simétrica. Ahora, probemos que H_n es definida positiva. Por definición, una matriz es definida positiva si, para todo vector no nulo $\vec{x} \in \mathbb{R}^{n+1}$, se cumple que $\vec{x}^T H_n \vec{x} > 0$. Evaluemos:

$$\begin{aligned} \vec{x}^T H_n \vec{x} &= \begin{bmatrix} x_0 & x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} \frac{1}{1} & \frac{1}{2} & \cdots & \frac{1}{n+1} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n+1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n} \frac{x_i x_j}{i+j+1}. \end{aligned}$$

Observemos que:

$$\begin{aligned}\sum_{i=0}^{n}\sum_{j=0}^{n}\frac{x_i x_j}{i+j+1} &= \sum_{i=0}^{n}\sum_{j=0}^{n} x_i x_j \int_0^1 t^{i+j}\,dt \\ &= \int_0^1 \left(\sum_{i=0}^{n} x_i t^i\right)^2 dt \\ &= \left\|\sum_{i=0}^{n} x_i t^i\right\|_{L^2}^2 .\end{aligned}$$

Dado que $\left(\sum_{i=0}^{n} x_i t^i\right)^2 > 0$ para cualquier $\vec{x} \neq \vec{0}$, se cumple que:

$$\vec{x}^T H_n \vec{x} > 0.$$

Por lo tanto, H_n es definida positiva.

c) Solucione el sistema $H_n x = b$, donde b tiene componentes $b_i = 1/(n+i-1)$, para $i = 1, \cdots, n$. Para esto, use las factorizaciónes LU `([L,U] = lu(H))` y Cholesky `L = chol(H);` y luego resuelva los dos sistemas triangulares, uno tras otro (es decir, realice las sustituciones hacia adelante y hacia atrás llamando al operador \ con matrices triangulares);
De forma arbitaria tomemos $n = 12$, y realicemos cada uno de los pasos que se describen en el enunciado en MATLAB.

```
% Parametro de tamano
n = 12;  % Puedes cambiar n a cualquier valor mayor

% Construir la matriz de Hilbert H_n
H = zeros(n);
for c = 1:n
  for r = 1:n
    H(r,c) = 1/(r+c-1);
  end
end

% Construir el vector b
b = 1 ./ (n + (1:n) - 1)';  % b(i) = 1 / (n + i - 1)

% Factorizacion LU
[L, U] = lu(H);

% Resolver el sistema utilizando LU (H * x = b)
% Paso 1: Resolver L * y = b usando sustitucion hacia adelante
y = L \ b;
```

```
21
22 % Paso 2: Resolver U * x = y usando sustitucion hacia atras
23 x_LU = U \ y;
24
25 % FactorizaciOn de Cholesky
26 L_chol = chol(H, 'lower');
27
28 % Resolver el sistema utilizando Cholesky (H * x = b)
29 % Paso 1: Resolver L_chol * y = b usando sustitucion hacia adelante
30 y_chol = L_chol \ b;
31
32 % Paso 2: Resolver L_chol' * x = y usando sustitucion hacia atras
33 x_chol = L_chol' \ y_chol;
34
35 % Mostrar los resultados
36 disp('Solucion usando LU:');
37 disp(x_LU);
38
39 disp('Solucion usando Cholesky:');
40 disp(x_chol);
41 disp(cond(H));
```

```
>> untitled
Solución usando LU:
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     1
```

```
Solución usando Cholesky:
    0.0000
   -0.0000
    0.0000
   -0.0000
    0.0001
   -0.0003
    0.0008
   -0.0014
    0.0016
   -0.0012
    0.0005
    0.9999
```

La matriz de Hilbert es conocida por ser mal condicionada, lo que se refleja en el alto valor de su condición ($1{,}7086 \times 10^{16}$), lo que indica que la matriz es muy sensible a pequeños errores numéricos. Este alto valor de condición causa que el método de Cholesky produzca errores significativos en las soluciones, como se observa en los valores de la solución, que

Problema 4 continúa en la siguiente página...

están lejos de los esperados. Estos errores pueden atribuirse a la acumulación de errores de redondeo y a la inestabilidad numérica inherente en la factorización de Cholesky para matrices mal condicionadas.

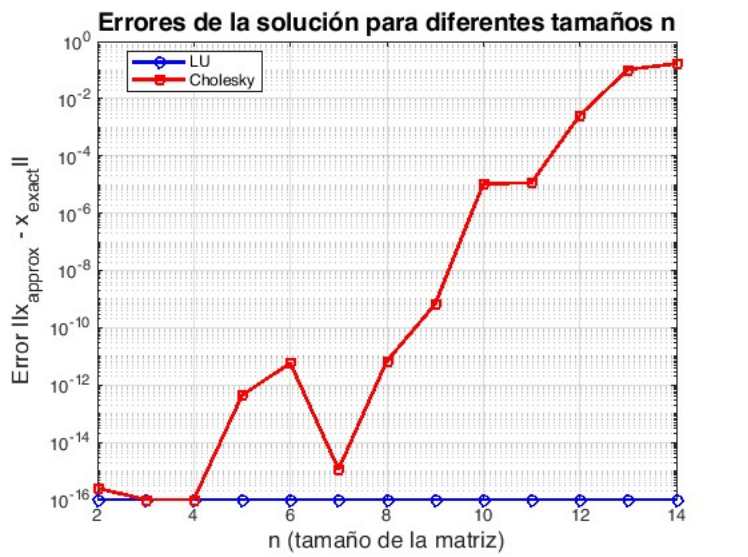
d) Para ambos métodos ¿Qué tan precisas son la soluciones numéricas $\hat{x}_{approx}$? Tabule los errores de la solución:

$$e(n) = \|x_{approx} - x_{exact}\|$$

Como una función de $n = 2, \cdots, 15$. Note que $x_{exact} = (0, \cdots, 1)^T$.Puede graficar los errores en función de n utilizando la función `semilogy` de Matlab. Explique en detalle los resultados.
En este punto, ya habíamos demostrado que la matriz de Hilbert es definida positiva. Nuestro objetivo es analizar su comportamiento numérico al aplicarle métodos de factorización, como LU y Cholesky, para identificar las limitaciones computacionales asociadas a esta matriz.

```
% Rango de tamaños de matriz
n_values = 2:15;
errors_LU = zeros(length(n_values), 1);
errors_Chol = zeros(length(n_values), 1);

% Calcular el error para cada tamaño de matriz
for k = 1:length(n_values)
    n = n_values(k);
    % Solución exacta (x_exact)
    x_exact = zeros(n, 1);
    x_exact(end) = 1;

    % Construir la matriz de Hilbert H_n
    H = zeros(n);
    for c = 1:n
        for r = 1:n
            H(r,c) = 1/(r+c-1);
        end
    end

    % Construir el vector b
    b = 1 ./ (n + (1:n) - 1)';   % b(i) = 1 / (n + i - 1)

    % Solución usando LU
    [L, U] = lu(H);
    y_LU = L \ b;
    x_LU = U \ y_LU;

```

```
29
30      % Solución usando Cholesky
31      L_chol = chol(H, 'lower');
32      y_Chol = L_chol \ b;
33      x_Chol = L_chol' \ y_Chol;
34
35      % Calcular el error para LU
36      errors_LU(k) = norm(x_LU - x_exact(1:n));
37
38      % Calcular el error para Cholesky
39      errors_Chol(k) =norm( x_Chol - x_exact(1:n));
40    % Mostrar los errores en la tabla
41      fprintf('%d\t\t%.5e\t%.5e\n', n, errors_LU(k), errors_Chol(k));
42  end
43
44  % Reemplazar ceros con un valor pequeño
45  errors_LU(errors_LU == 0) = 1e-16;
46  errors_Chol(errors_Chol == 0) = 1e-16;
47
48  % Graficar los errores con escala logarítmica
49  figure;
50  semilogy(n_values, errors_LU, 'b-o', 'LineWidth', 2, 'MarkerSize',
        6);
51  hold on;
52  semilogy(n_values, errors_Chol, 'r-s', 'LineWidth', 2,
        'MarkerSize', 6);
53  hold off;
54
55  % Etiquetas y título
56  xlabel('n (tamaño de la matriz)', 'FontSize', 13);
57  ylabel('Error ||x_{approx} - x_{exact}||', 'FontSize', 13);
58  title('Errores de la solución para diferentes tamaños n',
        'FontSize', 14);
59  legend('LU', 'Cholesky', 'Location', 'Best');
60  grid on;
```




```
>> Puntode
2               0.00000e+00     2.48253e-16
3               0.00000e+00     0.00000e+00
4               0.00000e+00     0.00000e+00
5               0.00000e+00     4.37678e-13
6               0.00000e+00     5.79726e-12
7               0.00000e+00     1.13229e-15
8               0.00000e+00     6.76181e-12
9               0.00000e+00     6.50591e-10
10              0.00000e+00     1.06223e-05
11              0.00000e+00     1.16067e-05
12              0.00000e+00     2.65368e-03
13              0.00000e+00     1.03880e-01
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.016600e-17.
> In Puntode (line 27)

14              0.00000e+00     1.66572e-01
Error using chol
Matrix must be positive definite.

Error in Puntode (line 31)
    L_chol = chol(H, 'lower');
             ^^^^^^^^^^^^^^^^
```

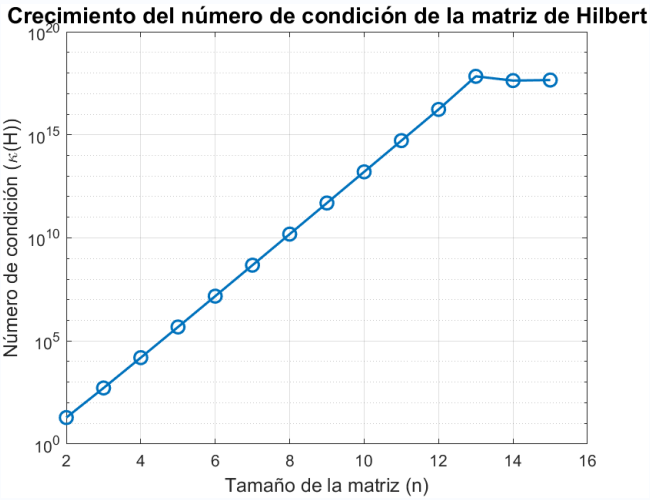
1) **Condición numérica de la matriz de Hilbert**

La matriz de Hilbert H fue construida para diferentes tamaños n. Esta matriz es teóricamente definida positiva y, por lo tanto, debería ser apta para la factorización de Cholesky. Sin embargo, las pruebas en MATLAB revelaron que, para valores mayores de n, la función de factorización de Cholesky no puede completarse, arrojando un error indicando que la matriz no es definida positiva. Esto se debe a que la matriz de Hilbert es conocida por ser extremadamente mal condicionada. Esto quedó evidenciado en los cálculos del número de condición $\kappa(H)$, el cual aumenta rápidamente con n. Esta tendencia se puede observar en el siguiente algoritmo, que calcula los números de

Problema 4 continúa en la siguiente página...

condición para n desde 2 hasta 15 y los grafica:

```
% Cálculo y graficación del número de condición de la matriz de
    Hilbert
n_values = 2:15;  % Valores de n desde 2 hasta 15
cond_numbers = zeros(size(n_values));  % Vector para almacenar
    los números de condición

for i = 1:length(n_values)
  n = n_values(i);            % Tamaño de la matriz actual
  H = hilb(n);                % Generar la matriz de Hilbert
  cond_numbers(i) = cond(H); % Calcular el número de condición
end

% Graficar el número de condición
figure;
semilogy(n_values, cond_numbers, '-o', 'LineWidth', 1.5,
    'MarkerSize', 8);
grid on;
xlabel('Tamaño de la matriz (n)', 'FontSize', 12);
ylabel('Número de condición (\kappa(H))', 'FontSize', 12);
title('Crecimiento del número de condición de la matriz de
    Hilbert', 'FontSize', 14);
```



La gráfica generada por este código muestra cómo el número de condición $\kappa(H)$ crece de manera significativa a medida que n aumenta, destacando la extrema sensibilidad de esta matriz a perturbaciones numéricas.

2) **Impacto en la factorización de Cholesky**

La factorización de Cholesky requiere que la matriz sea definida positiva no solo teóricamente, sino también en la práctica numérica. Durante el procedimiento, el algoritmo comprueba que los elementos diagonales de las submatrices menores sean estrictamente positivos. Sin embargo, en el caso de la matriz de Hilbert, los errores de redondeo acumulados generan perturbaciones en las entradas de H, llevando a la detección de valores numéricamente no positivos en las submatrices. Este fenómeno se vuelve más evidente a medida que el tamaño n de la matriz aumenta, lo cual incrementa la susceptibilidad del cálculo a los errores de precisión. Como resultado, el algoritmo de Cholesky falla al interpretar H como no definida positiva.

3) **Comparación con la factorización LU**
A diferencia de la factorización de Cholesky, el método de descomposición LU no tiene restricciones sobre la positividad definida de la matriz. Por esta razón, las pruebas en MATLAB mostraron que la factorización LU pudo completarse exitosamente incluso para matrices de Hilbert de tamaño grande **hasta** $n \leq 25$. Sin embargo, debido a la condición numérica extremadamente alta de H, la solución obtenida mediante LU también está sujeta a errores acumulados **desde** $n > 25$.

```
% Parámetro de tamaño
n = 26;  % Puedes cambiar n a cualquier valor mayor

% Construir la matriz de Hilbert H_n
H = zeros(n);
for c = 1:n
    for r = 1:n
        H(r,c) = 1/(r+c-1);
    end
end

% Construir el vector b
b = 1 ./ (n + (1:n) - 1)';  % b(i) = 1 / (n + i - 1)

% Factorización LU
[L, U] = lu(H);

% Resolver el sistema utilizando LU (H * x = b)
% Paso 1: Resolver L * y = b usando sustitución hacia adelante
y = L \ b;

% Paso 2: Resolver U * x = y usando sustitución hacia atrás
x_LU = U \ y;

% Mostrar los resultados
```

```
disp('Solución usando LU:');
disp(x_LU);

disp(cond(H));
```

```
Solución usando LU:
    0.0000
   -0.0000
    0.0000
   -0.0001
    0.0012
   -0.0066
    0.0210
   -0.0374
    0.0279
    0.0089
   -0.0040
   -0.0577
    0.0530
    0.0249
   -0.0090
   -0.0637
    0.0536
   -0.0653
    0.1303
   -0.0840
   -0.0394
    0.0824
   -0.0402
   -0.0077
    0.0177
    0.9941
```

Problema 5:

a) Simplifique el algoritmo de eliminación de Gauss de manera adecuada para resolver un sistema lineal $Ax = b$ con

$$A = \begin{pmatrix} a_{11} & a_{12} & & & \\ a_{12} & a_{22} & a_{23} & & \\ & a_{23} & \ddots & \ddots & \\ & & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ & & & a_{n-1,n} & a_{n,n} \end{pmatrix} \in \mathbb{R}^{n\times n}$$

matriz tridiagonal.

b) Considere la ecuación de Poisson con término fuente f en el intervalo $(0, 1)$:

$$-T"(x) = f(x), x \in (0, 1),$$

con condiciones de frontera $T(0) = T(1) = 0$. Para resolver numéricamente el problema de valor en la frontera, la segunda derivada de T se aproxima por medio de diferencias finitas con un paso de discretización $h > 0$:

$$T"(x) \approx \frac{T(x-h) - 2T(x) + T(x+h)}{h^2}$$

Tomando $x_i = ih$, $i = 0, 1, \ldots, n$, $n = 1/h$, obtenemos el sistema lineal de ecuaciones

$$-T_{i-1} + 2T_i - T_{i+1} = h^2 f(x_i), \qquad i = 1, \ldots, n-1 \tag{1}$$

para las incognitas $T_i \approx T(x_i)$, $i = 1, \ldots, n-1$ (de las condiciones de frontera se tiene que $T_0 = T_n = 0$). Escriba (1) en la forma $AT = f$ donde A es una matriz tridiagonal. Utilice su algoritmo desarrollado en la parte a) para resolver el sistema lineal de ecuaciones para $n = 1000$ y $f(x) = \text{sen}(2\pi x)/(4\pi^2)$.

Solución:

a) Con el objetivo de simplificar el algoritmo de eliminación de Gauss, tomemos como base el algoritmo presentado en clase y la representación general de la matriz A,
Algoritmo de eliminación de Gauss:

Problema 5 continúa en la siguiente página...

Para $k = 1, 2, \ldots, n-1$:
  Para $i = k+1, k+2, \ldots, n$:
    $m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$
    Para $j = k+1, k+2, \ldots, n$:
      $a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}$
      $b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)}$
Para $i = n, n-1, \ldots, 1$:
  $x_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j \right)$

$$A = \begin{pmatrix} \ddots & \ddots & & & \\ \ddots & a_{kk} & a_{k,k+1} & & \\ & 0 & a_{k+1,k+1}^{(k)} & a_{k+1,k+2}^{(k)} & \\ & & a_{k+2,k+1}^{(k)} & a_{k+2,k+2}^{(k)} & \ddots \\ & & & \ddots & \ddots \end{pmatrix}$$

En primera instancia, eliminaremos el segundo contador ($i = k+1, k+2, \ldots, n$), pues la única componente $a_{ik}^{(k)}$ presuntamente distinta de cero es $i = k+1$. Por ende, el contador es innecesario.

Además, eliminaremos también el tercer contador ($j = k+1, k+2, \ldots, n$), ya que dichas operaciones realizan la eliminación entre filas, necesaria únicamente para las componentes de la fila k-ésima presuntamente distintas de cero, lo cual solo ocurre para $a_{k,k+1}^{(k)}$. Por lo tanto, el contador es descartado. Es importante recalcar que $a_{k+1,k+2}^{(k+1)} = a_{k+1,k+2}^{(k)}$.

Finalmente, conservaremos el último contador, pero modificaremos la sumatoria. Esto se debe a que, después de realizar la eliminación de Gauss, por cada fila solo habrá dos componentes distintas de cero: $a_{ii}^{(i)}$ y $a_{i,i+1}^{(i)}$. En consecuencia, la sumatoria no es necesaria.

Por consiguiente, el **Algoritmo de eliminación de Gauss modificado** queda de la siguiente forma:

Para $k = 1, 2, \ldots, n-1$:
  $m_{k+1,k} = \frac{a_{k+1,k}^{(k)}}{a_{kk}^{(k)}}$
  $a_{k+1,k+1}^{(k+1)} = a_{k+1,k+1}^{(k)} - m_{k+1,k} a_{k,k+1}^{(k)}$
  $a_{k+1,k+2}^{(k+1)} = a_{k+1,k+2}^{(k)}$
  $b_{k+1}^{(k+1)} = b_{k+1}^{(k)} - m_{k+1,k} b_k^{(k)}$
Para $i = n-1, \ldots, 1$:
  $x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}}$
  $x_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - a_{i,i+1}^{(i)} x_{i+1} \right)$

b) Siguiendo las intrucciones enunciadas obtenemos el sistema lineal de ecuaciones.

$$\begin{cases} -T_0 + 2T_1 - T_2 = h^2 f(x_1), \\ -T_1 + 2T_2 - T_3 = h^2 f(x_2), \\ \quad \vdots \qquad\qquad \vdots \\ -T_{n-2} + 2T_{n-1} - T_n = h^2 f(x_{n-1}). \end{cases}$$

donde $x_i = ih$, $i = 0, 1, \ldots, n$, $n = 1/h$ y $T_i \approx T(x_i)$, $i = 1, \ldots, n-1$ (con condiciones de frontera $T_0 = T_n = 0$). Luego con dichas condiciones el sistema lineal queda de la siguiente

Problema 5 continúa en la siguiente página...

forma:

$$\begin{cases} 2T_1 - T_2 = h^2 f(x_1), \\ -T_1 + 2T_2 - T_3 = h^2 f(x_2), \\ \qquad \vdots \qquad\qquad\qquad \vdots \\ -T_{n-2} + 2T_{n-1} = h^2 f(x_{n-1}). \end{cases}$$

Por ende en forma matricial, $AT = f$,

$$\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 2 & -1 \\ 0 & 0 & \dots & -1 & 2 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_{n-1} \end{pmatrix} = \begin{pmatrix} h^2 f(x_1) \\ h^2 f(x_2) \\ h^2 f(x_3) \\ \vdots \\ h^2 f(x_{n-1}) \end{pmatrix}.$$

Ahora en nuestro caso particular tomando $n = 1000$ y $f(x) = \text{sen}(2\pi x)/(4\pi^2)$, el algoritmo queda de la siguiente forma:

```python
import numpy as np

n = 1000 # Tamano de la matriz
c = -1  # Numero que deseas en la diagonal superior e inferior inmediata
a = 2  # Numero que deseas en la diagonal principal


# Crear matriz de ceros
matriz = np.zeros((n, n))

# Llenar la diagonal superior e inferior inmediata
for i in range(n - 1):
  matriz[i, i + 1] = c
  matriz[i + 1, i] = c

# Llenar la diagonal principal
for i in range(n):
  matriz[i, i] = a

print(matriz)
```

[h]

```
[[ 2. -1.  0. ...  0.  0.  0.]
 [-1.  2. -1. ...  0.  0.  0.]
 [ 0. -1.  2. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  2. -1.  0.]
 [ 0.  0.  0. ... -1.  2. -1.]
 [ 0.  0.  0. ...  0. -1.  2.]]
```

Problema 5 continúa en la siguiente página...

```python
# Definimos f(x)
def funcion(x):
  return np.sin(2 * np.pi * x)

# Definimos h para la matriz
h=1/n

# definimos nuestro vactor independiente
valores = []  # Lista con valores especificos
for i in range(1, n + 1):
  valores.append((h ** 2) * funcion(i * h))

# Convertir la lista en un vector columna nx1
b = np.array(valores).reshape(n, 1)  # Convertirlos en vector
    columna nx1
print(b[:5])
```

```
[[6.28314397e-09]
 [1.25660399e-08]
 [1.88484397e-08]
 [2.51300954e-08]
 [3.14107591e-08]]
```

```python
# Eliminacion de Gauss simplificado para tridiagonales
for i in range(n-1):
  multiplicador = matriz[i + 1, i] / matriz[i, i]
                          # Encontrando los multiplicadores para
      la fila i+1
  matriz[i + 1 ,i] = 0
  matriz[i + 1, i + 1] = matriz[i + 1, i + 1] - multiplicador *
      matriz[i, i + 1]  # Actualizando la fila i+1
  b[i + 1] = b[i + 1] - multiplicador * b[i]   # Aplicando la
      operacion al termino independiente
  print("Matriz despues de aplicarle eliminacion de Gauss (primeros
      5 filas):")
print(matriz)
```

```
Matriz despues de aplicarle eliminación de Gauss :
[[ 2.         -1.          0.         ...  0.          0.
   0.        ]
 [ 0.          1.5        -1.         ...  0.          0.
   0.        ]
 [ 0.          0.          1.33333333 ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  1.001002   -1.
   0.        ]
 [ 0.          0.          0.         ...  0.          1.001001
  -1.        ]
 [ 0.          0.          0.         ...  0.          0.
   1.001     ]]
```

```
print("Termino independiente b (primeros 5 valores):")
print(b[:5])
```

```
Término independiente b (primeros 5 valores):
[[6.28314397e-09]
 [1.57076119e-08]
 [2.93201810e-08]
 [4.71202312e-08]
 [6.91069440e-08]]
```

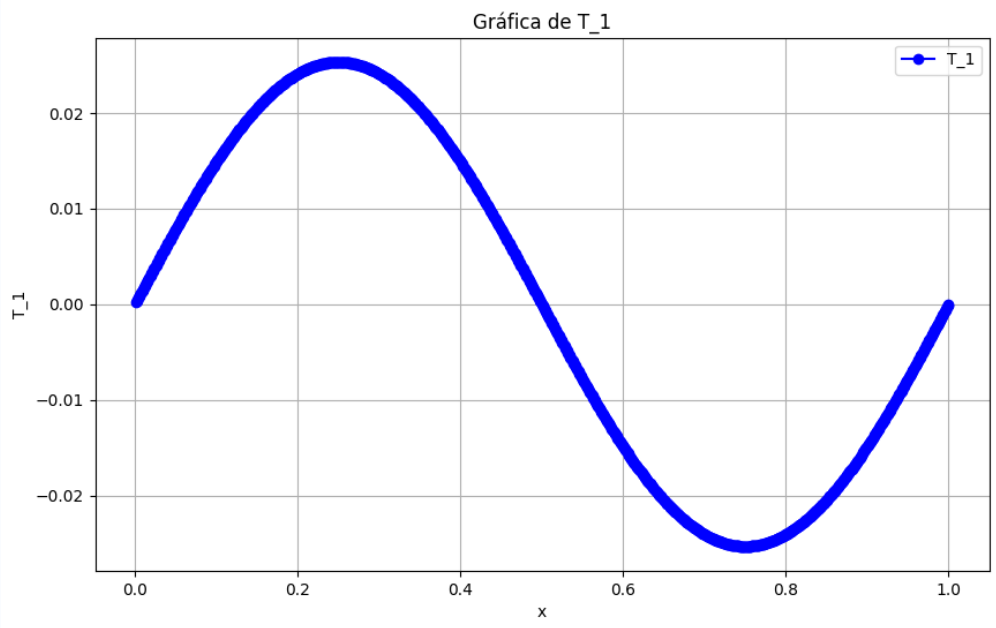
```
# Sustitución regresiva simplificada
x = np.zeros((n, 1))  # Crear el vector solucion

for j in range(n - 1, -1, -1):  # Desde la ultima fila hasta la
    primera
  if j == n - 1:              # Caso base: ultima fila
    x[j] = b[j] / matriz[j, j]
  else:
    x[j] = (b[j] - matriz[j, j + 1] * x[j + 1]) / matriz[j, j]

print("Solucion x (primeros 5 valores):")
print(x[:5])
```

```
Solución x (primeros 5 valores):
[[0.000159  ]
 [0.00031798]
 [0.00047696]
 [0.00063592]
 [0.00079485]]
```

```python
# Definimos la funci0n real
def funcion_real(x):
  return np.sin(2 * np.pi * x) / (4 * (np.pi ** 2))

# Definimos la funcion Error Absoluto
  def error_absoluto(x, h):
    # Creamos una copia para no modificar la original
    errores = np.zeros_like(x)  # Vector de errores
    valores_T1 = np.zeros_like(x)  # Vector para almacenar los
        valores de T_1

    for i in range(len(x)):
      T_1 = funcion_real((i + 1) * h)  # Calcula el valor real
      valores_T1[i] = T_1  # Guarda el valor de T_1 en el vector
      errores[i] = abs(T_1 - x[i])  # Calcula el error absoluto

    return errores, valores_T1, x
  values = np.zeros((n,1))
  for i in range(1000):
    values[i] = (i+1) * h

  # Llamamos a la funcion
  errores, valores_T1, x = error_absoluto(x, h)

  # Mostramos los resultados
  print("Valores reales T_1 (primeros 5 valores):")
  print(valores_T1[:5])
  print("\nError absoluto (primeros 5 valores):")
  print(errores[:5])
```

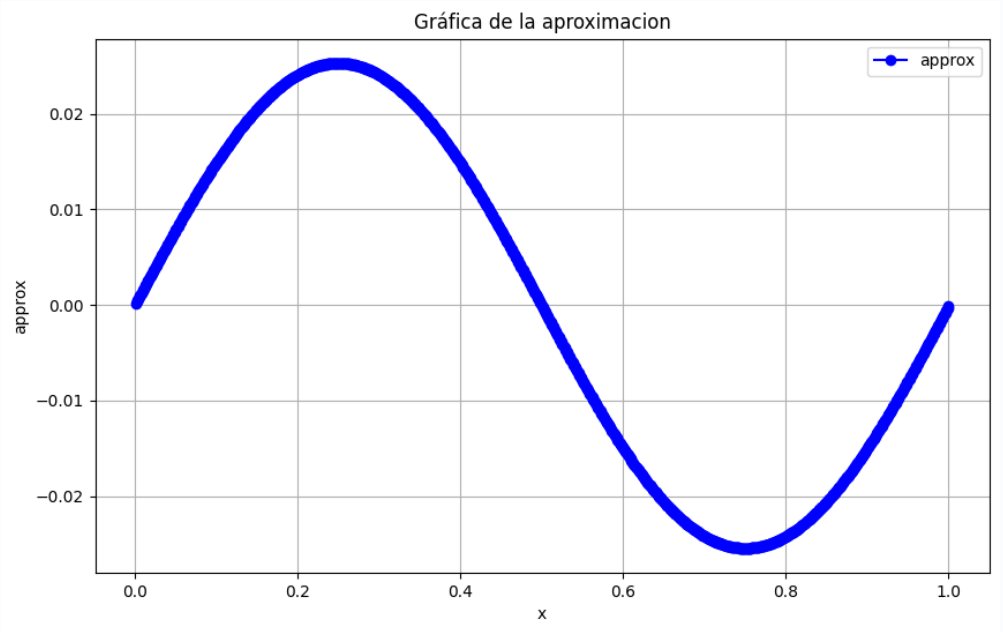
```
Valores reales T_1 (primeros 5 valores):
[[0.00015915]
 [0.0003183 ]
 [0.00047744]
 [0.00063655]
 [0.00079564]]

Error absoluto (primeros 5 valores):
[[1.58471828e-07]
 [3.16943676e-07]
 [4.75415566e-07]
 [6.33887517e-07]
 [7.92359552e-07]]
```

```python
import pandas as pd
import matplotlib.pyplot as plt

tabla = pd.DataFrame({
    "x": values.flatten(),
    "T_1": valores_T1.flatten(),
})
# Supongamos que 'tabla' ya está creada con tus datos
plt.figure(figsize=(10, 6))

# Graficar T_1
plt.plot(tabla['x'], tabla['T_1'], label='T_1', marker='o',
    linestyle='-', color='b')

# Personalizar la gráfica
plt.title('Gráfica de T_1')
plt.xlabel('x')
plt.ylabel('T_1')
plt.grid(True)
plt.legend()
plt.show()
print(tabla.head(1000))
```



	x	T_1
0	0.001	1.591539e-04
1	0.002	3.183015e-04
2	0.003	4.774366e-04
3	0.004	6.365528e-04
4	0.005	7.956438e-04
..	...	...
995	0.996	-6.365528e-04
996	0.997	-4.774366e-04
997	0.998	-3.183015e-04
998	0.999	-1.591539e-04
999	1.000	-6.204133e-18

```python
tabla = pd.DataFrame({
    "x": values.flatten(),
    "approx": x.flatten(),
})
# Supongamos que 'tabla' ya está creada con tus datos
plt.figure(figsize=(10, 6))

# Graficar x
plt.plot(tabla['x'], tabla['approx'], label='approx', marker='o',
    linestyle='-', color='b')

# Personalizar la gráfica
plt.title('Gráfica de la aproximacion')
plt.xlabel('x')
plt.ylabel('approx')
plt.grid(True)
plt.legend()
plt.show()
print(tabla.head(1000))
```



```
          x    approx
0     0.001  0.000159
1     0.002  0.000318
2     0.003  0.000477
3     0.004  0.000636
4     0.005  0.000795
..      ...       ...
995   0.996 -0.000795
996   0.997 -0.000636
997   0.998 -0.000477
998   0.999 -0.000318
999   1.000 -0.000159
```

```python
tabla = pd.DataFrame({
    "x": x.flatten(),
    "T_1": valores_T1.flatten(),
    "Error Absoluto": errores.flatten()
})
print(tabla.head(1000))
```

```
             x           T_1  Error Absoluto
0     0.000159  1.591539e-04    1.584718e-07
1     0.000318  3.183015e-04    3.169437e-07
2     0.000477  4.774366e-04    4.754156e-07
3     0.000636  6.365528e-04    6.338875e-07
4     0.000795  7.956438e-04    7.923596e-07
..         ...           ...             ...
995  -0.000795 -6.365528e-04    1.583615e-04
996  -0.000636 -4.774366e-04    1.585200e-04
997  -0.000477 -3.183015e-04    1.586785e-04
998  -0.000318 -1.591539e-04    1.588370e-04
999  -0.000159 -6.204133e-18    1.589954e-04
```

Problema 5 continúa en la siguiente página...

Usando Python para cálcular el número de condición se tiene que:

```
condicion = np.linalg.cond(matriz)
print("Número de condición:", condicion)
```

en dónde la salida es:
Número de condición: 782,9164494157501.
Cuando se utiliza $n = 1000$, la propagación de errores debido a la eliminación modificada y la sustitución hacia atrás de Gauss es extremadamente alta, lo que provoca que la aproximación tenga un error cercano al 99 %. Esto se debe a que, al ser tan grande el valor de n, el tamaño de paso h es muy pequeño, lo que introduce errores de truncamiento en cada operación del proceso de solución. Estos errores se acumulan a lo largo de las iteraciones, amplificando la inexactitud de la solución final. El comportamiento observado sugiere que, aunque el método parece adecuado para valores más pequeños de n, al aumentar el número de nodos, el método no es suficientemente preciso debido a la acumulación de errores en las operaciones, lo que limita su eficacia a medida que la discretización se afina. Esto destaca la importancia de encontrar un balance adecuado entre la resolución del problema y la precisión numérica para evitar la propagación excesiva de errores.