

MÓDULO 8: SISTEMA DE ECUACIONES LINEALES.

A- MÉTODOS DIRECTOS

1. Sistemas fáciles de resolver

Ejercicio 1: Escribe una función MATLAB llamada $x=sp(A,b)$ que admita como parámetros de entrada una matriz triangular inferior A y un vector columna b , con la parte derecha del sistema de ecuaciones, y devuelva, como parámetro de salida x , la solución del sistema obtenida aplicando el algoritmo de sustitución progresiva.

Aplicar este algoritmo para resolver el sistema de ecuaciones:

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

Ejercicio 2: Escribe una función MATLAB llamada $x=sr(A,b)$ que admita como parámetros de entrada una matriz triangular superior A y un vector columna b con la parte derecha del sistema de ecuaciones y devuelva, como parámetro de salida x , la solución del sistema.

Aplicar este algoritmo para resolver el sistema de ecuaciones:

$$\begin{bmatrix} 1 & 4 & 5 \\ 0 & 3 & 2 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$

2. Factorizaciones LU. Método de eliminación Gaussiana.

Ejercicio 3: Escribe una función MATLAB llamada $[A,b]=rp(A,b)$ que admita como parámetros de entrada una matriz A y un vector columna b con la parte derecha del sistema de ecuaciones y devuelva, como parámetro de salida x , las matriz A y el vector b , tras haber aplicado el algoritmo de reducción progresiva. Resolver el sistema de ecuaciones $Ax=b$.

Aplicar este algoritmo sobre el siguiente sistema de ecuaciones:

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix}$$

Ejercicio 4: Escribe una función MATLAB llamada `[L,A]=mi_lu(A)` que admita como parámetros de entrada una matriz A y devuelva, como parámetro de salida, su descomposición LU. ¿Cómo se resolvería un sistema de ecuaciones $Ax=b$?

Comparar los resultados con la función `lu` de MATLAB. ¿Cuándo habría diferencias? ¿Cómo se resolvería en ese caso el sistema de ecuaciones utilizando la función `lu` de MATLAB?

Aplicar este algoritmo para obtener las matrices LU de la factorización de la siguiente matriz:

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -3 \end{bmatrix}$$

3. Mejoras en el método de eliminación Gaussiana

3.1 Pivoteo parcial o de filas

El sistema MATLAB resuelve directamente un sistema de ecuaciones con la operación $x=A \setminus b$. En función de las características de la matriz aplica un método u otro. En caso de que la matriz no sea simétrica y definida positiva aplica el método de eliminación Gaussiana con pivoteo.

Ejercicio adicional: Escribir una función MATLAB `x=eg(A,b)` que resuelva un sistema de ecuaciones introducido como parámetro de entrada por medio del método de eliminación Gaussiana con pivoteo. Para ello implementar la función reducción progresiva con pivoteo y emplear, a continuación, el algoritmo de sustitución regresiva.

Sugerencia:

reducción progresiva con pivoteo

for $k = 1: n-1$

Encontrar el elemento pivote m ; $|a_{mk}| = \max\{|a_{ik}|: i \geq k\}$

Si $a_{mk} = 0$

El sistema no tiene solución

Caso contrario

Intercambiar filas m y k

Reduccion progresiva estándar
end

Para intercambiar dos filas (m y k) podemos copiar la fila k (incluido b_k) en un vector auxiliar, copiar la fila m en la fila k y, finalmente, copiar el vector auxiliar en la fila m . Si el número de ecuaciones es grande, otra solución más rápida consiste en utilizar un vector de índices que indique en cada posición la fila de la matriz que ocupa ese lugar. De este modo, para intercambiar filas solo hay que intercambiar dos índices, los correspondientes a las posiciones k y m .

4. Factorización de Cholesky

El sistema MATLAB dispone de la función `chol` para realizar esta descomposición.

Ejercicio 5:¿Cómo se resuelve un sistema de ecuaciones utilizando esta factorización?

Ejercicio opcional: Escribir un programa MATLAB que realice la descomposición de Cholesky de una matriz. Para comprobar que es correcta se puede comparar con la función de MATLAB `chol`

5. Factorización ortogonal o QR

Ejercicio 6:¿Cómo se resuelve un sistema de ecuaciones utilizando esta factorización?

6. Números de condición y errores en la solución

Hay casos en los que a pesar de haber aplicado el algoritmo con pivoteo, la solución numérica obtenida no es fiable. Estudiar el siguiente sistema:

$$\begin{aligned}0.832 x_1 - 0.448 x_2 &= 1 \\0.784 x_1 + 0.421 x_2 &= 0\end{aligned}$$

Ahora resolver el mismo sistema pero con el coeficiente a_{22} incrementado en 0.001:

Ejercicio 7:¿Qué ha ocurrido? ¿Por qué? ¿Cómo podemos detectarlo? ¿Cómo se puede acotar el error?

Ejercicio opcional: Dar un ejemplo de un sistema de ecuaciones en el que difiera la solución en función de si se aplica o no el pivoteo parcial.

B-MÉTODOS ITERATIVOS

Ejercicio 1:

Escribe una función MATLAB llamada `xs1=jacobi(A,b,xs)` que admita como parámetros de entrada la matriz del sistema de ecuaciones A , un vector columna b con la parte derecha del sistema de ecuaciones, y una aproximación a la solución x_s y devuelva, como parámetro de salida, el resultado $xs1$ de aplicar una iteración de Jacobi.

Implementar dos versiones de esta función, una que utilice exclusivamente operaciones matriciales y otra haciendo uso de bucles

Ejercicio 2:

Escribe una función MATLAB `x = mjacobi(A,b,x0,tolerancia)` que admita como parámetros de entrada una matriz A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación inicial a la solución x_0 y un valor de *tolerancia* y devuelva, como parámetro de salida, la solución x del sistema de ecuaciones dentro de la tolerancia dada. Además, debe visualizar el error y la solución aproximada en cada iteración.

Usar esta función para obtener la solución del siguiente sistema de ecuaciones dentro de un error ± 0.001 ¿ Cuántas iteraciones son necesarias? ¿ Y para un error de ± 0.00001 ?

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 5 & 1 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 4 \end{bmatrix}$$

Comprobar el resultado con: $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$

Ejercicio 3:

Escribe una función MATLAB llamada `xs1=jacobiw(A,b,xs,w)` que admita como parámetros de entrada la matriz del sistema de ecuaciones A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación x_s a la solución y un peso w , y devuelva, como parámetro de salida, el resultado $xs1$ de aplicar una iteración de Jacobi amortiguado con el peso dado.

Ejercicio 4:

Escribe una función MATLAB `x = mjacobiw(A,b,x0,w,tolerancia)` que admita como parámetros de entrada la matriz del sistema A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación inicial a la solución x_0 , el peso w y un valor de *tolerancia* y devuelva, como parámetro de salida, la solución x del sistema de ecuaciones dentro de la tolerancia dada. Además, debe visualizar el error y la solución aproximada en cada iteración.

a) Usar esta función para obtener la solución del siguiente sistema de ecuaciones dentro de un error ± 0.001 utilizando un peso de 0.5 ¿ Cuántas iteraciones son necesarias?

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 5 & 1 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 4 \end{bmatrix}$$

b) Aplicar la función *jacobiw.m* al sistema de ecuaciones del ejercicio 4, considerando una tolerancia de 0.001 y un peso elegido por el alumno tal que el proceso converga más rápidamente respecto al *método de Jacobi*.

Solución	w =	x =	nº de iteraciones :
----------	-----	-----	---------------------

Ejercicio 5:

a) Escribe una función MATLAB llamada `xs1 = gseidel(A,b,xs)` que realice una iteración de Gauss Seidel sobre el sistema de ecuaciones introducido como argumento de entrada.

b) Implementar la misma función manejando matrices directamente.

Ejercicio 6: Escribe una función MATLAB `x = mgseidel(A,b,x0,tolerancia)` que admita como parámetros de entrada una matriz A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación inicial a la solución final x_0 , y un valor de *tolerancia* y devuelva, como parámetro de salida x , la solución del sistema de ecuaciones dentro de la tolerancia dada utilizando el método de Gauss-Seidel. Además, debe visualizar el error y la solución aproximada en cada iteración.

Ejercicio 7:

a) Escribe una función MATLAB llamada `xs1=sor(A,b,xs,w)` que admita como parámetros de entrada la matriz del sistema de ecuaciones A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación a la solución xs y un peso w , y devuelva, como parámetro de salida $xs1$, el resultado de aplicar una SOR con el peso dado.

b) Implementar la misma función utilizando la expresiones matricial

Ejercicio 8:

Escribe una función MATLAB `x=msor(A,b,x0,tolerancia,w)` que admita como parámetros de entrada la matriz del sistema A , un vector columna b con la parte derecha del sistema de ecuaciones, una aproximación inicial x_0 a la solución final, un valor de *tolerancia* y un peso w , y devuelva, como parámetro de salida x , la solución del sistema de ecuaciones dentro de la tolerancia dada utilizando el método SOR. Además, debe visualizar el error y la solución aproximada en cada iteración.

Ejercicio 9: Supongamos el siguiente sistema de ecuaciones:

$$\begin{bmatrix} -4 & 1 & 1 & 1 \\ 1 & -4 & 1 & 1 \\ 1 & 1 & -4 & 1 \\ 1 & 1 & 1 & -4 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Si queremos alcanzar la solución con una tolerancia igual a ± 0.00001 y con una aproximación inicial a la solución $(0 \ 0 \ 0 \ 0)$. Haz las siguientes pruebas:

- ¿Cuántas iteraciones de Jacobi hacen falta para alcanzar la solución?
- ¿Cuántas iteraciones de Gauss-Seidel hacen falta para alcanzar la solución?
- Variando el peso del método SOR desde 1.0 a 1.9 con un incremento 0.1,

¿Cuántas iteraciones hacen falta para cada peso y qué peso minimiza el número de iteraciones?
