UNIVERSIDAD
PANAMERICANA

Universidad Panamericana

Escuela de Ingeniería

Matemáticas de la Computación

Descomposición LU para sistemas lineales y Método Gauss-Seidel Acelerado

Profesor

- David Iván Morales Huerta

Integrantes

- Ain Bolaños Cortés
- Diego Castañeda Ladrón de Guevara
- Santiago Espinosa Ollivier
- Cecilia Gaona Vidales
- Santiago Medina Domínguez

Entrega

- 2 de Diciembre, 2025

1. Resumen General del Reporte

Este reporte desarrolla, explica, justifica e implementa los métodos de descomposición LU y el método Gauss-Seidel para la resolución de sistemas de ecuaciones lineales. Con un enfoque en su resolución automatizada computable. Dando así la justificación teórico mátemática y su implementación algorítmica en Python.

2. Introducción

Los sistemas de ecuaciones lineales, así como su representación matricial son un problema típico de la computación, en áreas relevantes como la Visión de Computadora, la criptografía, los gráficos y el machine learning, así como en las simulaciones físicas y el análisis matemático.

Y de la misma manera en que los conceptos del álgebra lineal son relevantes para la expansión de los límites de la computación, el poder computacional extiende las posibilidades de procesamiento y resolución de sistemas lineales extensos.

Es por esto que, en este reporte analizaremos dos métodos para resolver estos problemas de matrices donde la computación puede fallar, debido a la complejidad y logitud de los sistemas de eciaciones lineales.

3. Primer Método: Descomposición LU

3.1. Introducción del Método

Como ya vimos, el problema reside en el análisis numérico y la computación científica, y se expresa en forma matricial $Ax = b$.

Este tipo de sistemas aparece en casi todos los ámbitos de la ingeniería y las ciencias aplicadas, desde el análisis de circuitos eléctricos, el diseño estructural, la modelización financiera, el aprendizaje automático, etc. Si bien es cierto que existen métodos iterativos (por ejemplo, el método de Gauss-Seidel que trabajará tu compañero) que encuentran una solución aproximada, los métodos directos se limitan a encontrar una solución exacta (dentro de la precisión de la máquina) en un número finito de pasos.

El método de Descomposición LU es uno de los métodos directos más fundamentales y eficientes. No es simplemente un algoritmo, sino una factorización de la matriz A que transforma el problema $Ax = b$ en un proceso de dos pasos mucho más simple de resolver. Esta investigación se centrará en la fundamentación teórica, la formulación matemática, el análisis de estabilidad y las ventajas de este crucial método.

3.2 Revisión de la Literatura

La factorización LU construye la formalización directa del método de eliminación gaussiana, que es un algoritmo muy antiguo con una historia de más de dos mil años. Sin embargo, hoy en día vista como una factorización de matrices resultó crucial para el desarrollo de las computadoras digitales. Alan Turing llegó a utilizar esta factorización como una de las herramientas centrales tratando de encontrar el análisis de errores en las computaciones basados en matrices, esto visto en su trabajo de 1948 "Rounding-Off Errors in Matrix Processes".

Los textos considerados canónicos de álgebra lineal numérica, los de Golub & Van Loan (2013) o los de Trefethen & Bau (1997), consideran la descomposición LU como la herramienta ideal para resolver sistemas lineales densos. Es precisamente la base sobre la cual se construyen algoritmos más complejos y es el método de facto que se encuentra implementado en paquetes de software de alto rendimiento como LAPACK y bibliotecas como NumPy (SciPy) o MATLAB (Trefethen & Bau, 1997).

La literatura no sólo se esfuerza por describir el algoritmo básico, sino que lo hace por describir, y exclusivamente por el mismo, las versiones estables que tienen pivoteo, indispensable para la aplicación práctica de estas ideas.

3.3 Metodología

Este método numérico requiere primero, para su justificación, las siguientes herramientas y conceptos matemáticos.

3.3.1 Definiciones Matemáticas

Matriz Cuadrada:
Sea $A \in M_{n\times m}(F)$, con $M_{n\times m}$ el conjunto de matrices del campo F con dimensiones $n \times m$. En el caso en que $n = m$ decimos que A es una matriz cuadrada. Para el futuro de este reporte, se tomará $F = \mathbb{R}$.

Vectores Canónicos:
Sea e_k un vector de dimensión $n \times 1$, decimos que este es k-ésimo vector canónico de $\mathbb{R}^n$ si cumple que con x_i sus entradas con $i \in \{1, \dots, n\}$, entonces $x_k = 1$ y $x_i = 0$ con $i \neq k$. Así pues se puede ver de la siguiente manera:

$$e_k = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Esto es útil también para referirnos a partes de una matriz A lo suficientemente grande. Dado que:

- $Ae_k = k$-ésima columna de A
- $e_k^T A = k$-ésima fila de A

Además, en dado caso de tener un vector v con dimensiones $n \times 1$, la multiplicación ve_k^T será una matriz cuadrada $n \times n$ con todos los valores 0 a excepción de la columna k que contendrá los valores de v.

3.3.2 Justificación Teórica de la Descomposición LU

Sea $A \in M_{n\times n}(\mathbb{R})$, lo que buscamos es la secuencia finita $\{L_k \in M_{n\times n} : 1 \leq k \leq n\}$ tales que para x_k la k-ésima columna de la matriz A ($x_k = Ae_k$):

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1,k} \\ \vdots \\ x_{nk} \end{bmatrix} \xrightarrow{L_k} L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Esto con el objetivo de obtener a partir de esta cantidad finita de L_k una matriz U triangular superior. De manera similar a la eliminación Gaussiana, obtenemos pues, suponiendo que $x_{kk} \neq 0$ los siguientes valores:

$$l_{jk} = \frac{x_{jk}}{x_{kk}} \qquad (k < j \leq m)$$

Esto significa que tenemos que pedirle a A que todos sus elementos en la diagonal **sean no nulos**. Con estas entradas entradas definamos $l_k = [0, \dots, 0, l_{k+1,k}, \dots, l_{nk}]^T$, y así obtenemos que L_k se verá de la siguiente manera:

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{nk} & & 1 \end{bmatrix} = I - l_k e_k^T$$

Con esto, generamos k matrices L_k tales que:

$$L_{n-1} \cdots L_2 L_1 A = U$$

La matriz deseada, y más aún, despejando para A obtenemos:

$$A = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} U$$

Por como está estructurada cada L_k, podemos ver que $L_k^{-1} = I + l_k e_k$, con lo que nos podemos tomar:

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = I + \sum_{k=1}^{n-1} l_k e_k^T = \begin{bmatrix} 1 & & & & \\ l_{21} & \ddots & & & \\ & & 1 & & \\ \vdots & & l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ l_{n1} & & l_{nk} & & 1 \end{bmatrix}$$

Con esto obtuvimos pues $A = LU$, con L matriz triangular inferior y U matriz triangular superior. Ahora, supongamos que existen L_1, L_2 matrices triangulares inferiores y U_1, U_2 matrices triangulares superiores tales que:

$$A = L_1 U_1 = L_2 U_2 \quad \Longrightarrow \quad L_2^{-1} L_1 = U_2 U_1^{-1}$$

Por cerradura de las matrices triangulares, $L_2^{-1} L_1$ es triangular inferior y $U_2 U_1^{-1}$ es triangular superior, así por la igualdad:

$$L_2^{-1} L_1 = U_2 U_1^{-1} = I \quad \Longrightarrow \quad L_1 = L_2 \ \wedge \ U_1 = U_2$$

Así pues, el despeje LU fue único para A. Como observación final, es fácil ver que L es fácil de obtener, al no ser necesario ningún despeje u operación entre matrices. Mientras que U se puede obtener al tiempo que se genera L. Con esto **queda demostrada la existencia y unicidad de** L, U, además de dar una pauta para la obtención de dichas matrices.

3.3.3 Algoritmo de la Descomposición LU

Como se puede apreciar en la deducción matemática, las operaciones de inversa para las L_i son sumamente simples de efectuar, así como el despeje de cada entrada de L se puede obtener de manera directa, lo que hace al algoritmo sumamente efectivo dado que es de matrices. Con esto, Lloyd N. y David Bau, agregan que no es necesario almacenar A, L o U en distintas matrices.

Generamos una matriz partiendo de la identidad, y vamos a encontrar todos los l_{jk}, es decir, los factores de la triangular inferior estricta de L para con ellos actualizar las entradas de U, haciendo la Eliminación Gaussiana.
El pseudo código a continuación supone que la matriz A dada es válida, es decir, $\forall k \leq n$, $A_{kk} \neq 0$. Con n claro el tamaño de la matriz A, es decir $A \in M_{n \times n}(F)$.

3.3.3.1 Pseudocódigo

```
tome U, L matrices n x n

U = A, L = I

para k = 1 hasta n - 1

   para i = k + 1 hasta n
      l_ik = u_ik/u_kk
      ik = factor

      para j = k hasta n
         u_ij -= l_ik * u_kj
```

```
      fin para
   fin para
fin para

retornar L, U
```

> Es importante notar que este algoritmo tiene **complejidad** $O(n^3)$, lo que implica que su funcionamiento en menos de 1 segundo, estará limitado a matrices de a lo más $n = 10^3$, suponiendo una cantidad de operaciones $\approx 10^9$ por segundo.

3.4. Aplicación de la Descomposición LU

3.4.1 Problema 1: Un Sistema 3×3 Básico

Se propone resolver el siguiente sistema simple para demostrar la trazabilidad manual del algoritmo:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 9 \\ 10 \end{bmatrix}$$

3.4.1.1 Desarrollo Explícito

Aplicando el algoritmo descrito en 3.3.3:$k = 1$: Los multiplicadores son $l_{21} = 1/1 = 1$ y $l_{31} = 1/1 = 1$. Restamos la fila 1 a las filas 2 y 3.$k = 2$: El nuevo pivote es $u_{22} = 1$. El multiplicador es $l_{32} = 1/1 = 1$. Restamos la fila 2 a la fila 3.Obtenemos las matrices:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

3.4.1.2 Resultado del Código

Al ejecutar el script en Python implementado con la metodología descrita, se obtuvieron los siguientes resultados en la consola, confirmando la exactitud de la factorización y la solución del sistema:

```
=== MATRIZ INICIAL A ===
A =
  [1, 1, 1]
  [1, 2, 2]
  [1, 2, 3]

... [Proceso de eliminación omitido por brevedad] ...

===== MATRIZ FINAL L =====
L =
  [1.0, 0.0, 0.0]
  [1.0, 1.0, 0.0]
  [1.0, 1.0, 1.0]

===== MATRIZ FINAL U =====
U =
  [1, 1, 1]
  [0.0, 1.0, 1.0]
  [0.0, 0.0, 1.0]
```

```
>>> SOLUCIÓN PROBLEMA 1 (x): [3.0, 2.0, 1.0]
```

3.4.2 Problema 2: El Análisis de un Circuito Resistivo

Se plantea un circuito con 3 mallas donde A representa las resistencias (Matriz de Impedancia) y b las fuentes de voltaje. El sistema está dado por las Leyes de Kirchhoff:

$$\begin{bmatrix} 10 & -5 & 0 \\ -5 & 15 & -5 \\ 0 & -5 & 10 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}$$

3.4.2.1 Desarrollo Explícito

Para resolver el sistema de circuitos presentado, realizamos la descomposición manual de la matriz A.

Paso 1 (Columna 1):
El pivote es $a_{11} = 10$. Buscamos eliminar el -5 de la fila 2 (a_{21}).
El multiplicador es $l_{21} = \frac{-5}{10} = -0.5$.
Operación elemental: $F_2 \leftarrow F_2 - (-0.5)F_1$.

$$\begin{bmatrix} 10 & -5 & 0 \\ -5 & 15 & -5 \\ 0 & -5 & 10 \end{bmatrix} \xrightarrow{F_2-(-0.5)F_1} \begin{bmatrix} 10 & -5 & 0 \\ 0 & 12.5 & -5 \\ 0 & -5 & 10 \end{bmatrix}$$

La fila 3 ya tiene un cero en la primera columna, por lo que $l_{31} = 0$.

Paso 2 (Columna 2):
El nuevo pivote es $a_{22} = 12.5$. Buscamos eliminar el -5 de la fila 3 (a_{32}).
El multiplicador es $l_{32} = \frac{-5}{12.5} = -0.4$.
Operación elemental: $F_3 \leftarrow F_3 - (-0.4)F_2$.

$$\begin{bmatrix} 10 & -5 & 0 \\ 0 & 12.5 & -5 \\ 0 & -5 & 10 \end{bmatrix} \xrightarrow{F_3-(-0.4)F_2} \begin{bmatrix} 10 & -5 & 0 \\ 0 & 12.5 & -5 \\ 0 & 0 & 8 \end{bmatrix} = U$$

Finalmente, construimos la matriz L colocando los multiplicadores calculados (-0.5 y -0.4) en sus posiciones correspondientes y unos en la diagonal principal.
El resultado de la descomposición es:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.4 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 10 & -5 & 0 \\ 0 & 12.5 & -5 \\ 0 & 0 & 8 \end{bmatrix}$$

3.4.2.2 Resultado del Código

Para el problema de circuitos, la ejecución del algoritmo arrojó los siguientes resultados. Se puede observar que las matrices L y U coinciden con el desarrollo manual, y el vector solución corresponde a las corrientes de cada malla en Amperios.

```
=== MATRIZ INICIAL A (CIRCUITOS) ===
A =
  [10, -5, 0]
  [-5, 15, -5]
  [0, -5, 10]

... [Cálculo de factores y actualización de filas] ...

===== MATRIZ FINAL L =====
```

```
L =
  [1.0, 0.0, 0.0]
  [-0.5, 1.0, 0.0]
  [0.0, -0.4, 1.0]

===== MATRIZ FINAL U =====
U =
  [10, -5, 0]
  [0.0, 12.5, -5.0]
  [0.0, 0.0, 8.0]

>>> SOLUCIÓN PROBLEMA 2 (Corrientes): [1.25, 0.5, 0.25]
```

3.5. Resultados

El algoritmo implementado logró descomponer y resolver exitosamente las matrices de prueba.

Precisión:

Al reconstruir la matriz original mediante la operación $A_{calc} = L \cdot U$, el error absoluto medio fue del orden de 10^{-16} (cero de máquina), lo que valida la implementación.

Eficiencia:

Para el Problema 2, una vez obtenidas L y U, cambiar el vector de voltaje b permitió encontrar nuevas corrientes realizando solo $O(n^2)$ operaciones (sustitución hacia adelante y atrás) en lugar de $O(n^3)$ (eliminación completa), confirmando la ventaja teórica del método.

3.6. Discusión y Conclusiones

La descomposición LU ha demostrado ser una herramienta robusta para la resolución de sistemas lineales cuadrados.Ventajas: Su principal fortaleza radica en la separación del procesamiento de la matriz A del vector b. Esto es crucial en simulaciones temporales o procesos iterativos de ingeniería donde la estructura del sistema no cambia.Limitaciones: Como se observó en la teoría, el algoritmo básico falla si algún pivote a_{kk} es cero (división por cero). Esto hace necesario implementar estrategias de pivoteo parcial (intercambio de filas) para garantizar la estabilidad numérica en casos generales, lo cual añade una matriz de permutación P al resultado ($PA = LU$).

En conclusión, dominar la factorización LU es indispensable para cualquier aplicación de cómputo científico, siendo el paso lógico previo al estudio de métodos más avanzados como Cholesky o QR.

4. Método Gauss-Seidel Acelerado (SOR)

4.1 Introducción del Método

Una vez comprendido el funcionamiento de Gauss-Seidel, contamos con un método iterativo que busca una solución aproximada reduciendo el costo computacional de las soluciones exactas.

Sobre esta base, el método **SOR** agrega una variable ω, conocida como **parámetro de relajación**, para controlar y eficientar la aproximación en cada iteración. Para que este método funcione, se debe cumplir una **condición necesaria**: el parámetro ω debe encontrarse en el intervalo abierto $0 < \omega < 2$. Si se elige un valor fuera de este rango, el error crecerá en lugar de disminuir.

Dependiendo del valor elegido, el comportamiento del método cambia:

- **Subrelajación ($0 < \omega < 1$):** Se usa para "frenar" el paso. Es útil para lograr la convergencia en sistemas que con Gauss-Seidel no convergerían.
- **Gauss-Seidel ($\omega = 1$):** Es el punto neutro (el método estándar).
- **Sobrerrelajación ($1 < \omega < 2$):** Se usa para acelerar la llegada a la solución en sistemas que ya son estables.

Planteamiento del Problema

El problema se plantea de la siguiente forma:

$$Ax = b$$

El objetivo es encontrar el vector solución **x** para un sistema de ecuaciones lineales algebraicas.

Donde:

- **A:** Es una matriz de coeficientes de tamaño $n \times n$.
- **x:** Es el vector incógnita que se debe encontrar (de tamaño $n \times 1$ para coincidir con **b**).
- **b:** Es el vector de términos independientes de tamaño $n \times 1$.

Las condiciones son:

- Se tiene un punto de partida o aproximación inicial $\mathbf{x}^{(0)}$.
- Un acelerador ω que actúa como parámetro de relajación.

Metodología de Solución

1. Verificación Previa
Antes de empezar, se debe verificar que la matriz **A** no tenga ceros en la diagonal principal ($a_{ii} \neq 0$), ya que estos valores se utilizan como divisores. Además, el parámetro de relajación ω debe elegirse dentro del rango $0 < \omega < 2$.

2. Fórmula Iterativa
Para cada incógnita x_i en la iteración $k+1$, aplicamos la siguiente fórmula:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

La interpretación de la fórmula es un **promedio ponderado**:

- El primer término $(1-\omega)x_i^{(k)}$ considera el valor de la iteración anterior.
- El segundo término (la fracción) representa la corrección calculada mediante el método de Gauss-Seidel.

3. Proceso Algorítmico

- **Inicio:** Se define un vector semilla $\mathbf{x}^{(0)}$ (por defecto suele ser un vector de ceros) y se establece un número máximo de iteraciones (N).
- **Ciclo Iterativo:** Para cada iteración $k = 1$ hasta N, se recorre cada fila i de la matriz (de 1 a n). Se calcula el nuevo valor x_i utilizando los valores más recientes de la iteración actual (para las columnas $j < i$) y los valores de la iteración anterior (para las columnas $j > i$).

4. Criterios de Parada
Al final de cada ciclo completo, se calcula la norma de la diferencia entre el vector nuevo y el anterior:

$$||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}|| < \text{Tolerancia}$$

- **Convergencia:** Si el error es menor que la tolerancia establecida, el proceso se detiene y se entrega el último vector calculado como solución.
- **Divergencia:** Si se alcanza el número máximo de iteraciones sin cumplir la tolerancia, el método se detiene indicando que no convergió.

4.2 Revisión de la Literatura

La fundamentación de este método se basa en una evolución histórica y teórica bien documentada, que abarca desde los métodos manuales del siglo XIX hasta la computación digital moderna.

Contexto Histórico

- **Antecedentes:** Carl Friedrich Gauss, en 1823, introdujo las bases de las técnicas iterativas y transmitió estos conceptos a uno de sus alumnos, Christian Gerling. Tiempo después, Philipp L. von Seidel formalizó el método en 1874 analizando sistemas de mínimos cuadrados. El algoritmo resultante lleva el nombre conjunto de **Gauss-Seidel**; este converge para matrices estrictamente diagonalmente dominantes o simétricas definidas positivas, aunque su velocidad de convergencia suele ser lenta para sistemas grandes.
- **Desarrollo del SOR:** El método SOR fue propuesto en 1950 por **David M. Young Jr.** y **Stanley P. Frankel** con el propósito de resolver sistemas lineales en ordenadores digitales.
  - **Young** estableció la teoría matemática rigurosa, demostrando la relación entre el radio espectral de la matriz y el parámetro óptimo de relajación (ω).
  - **Frankel** se enfocó en la aplicación práctica para computadoras digitales tempranas.

Anteriormente ya existían técnicas como la de Lewis Fry Richardson y los métodos desarrollados por R. V. Southwell. Sin embargo, estos no se podían aplicar eficientemente a computadoras digitales o, en el caso de Southwell, eran ineficientes porque requerían intervención visual humana.

4.3 Metodología

Este método numérico requiere primero, para su justificación, las siguientes herramientas y conceptos matemáticos.

4.3.1 Definiciones Matemáticas

Matriz Diagonalmente Dominante:
Sea $A \in M_{n\times n}(F)$ matriz cuadrada, se dice que es **diagonalmente dominante por filas** si se cumple que:

$$|a_{ii}| \geq \sum_{j\neq i} |a_{ij}| \quad (\forall i)$$

En el caso en que se use ($>$), a esto se le llaman **estrictamente diagonalmente dominante por filas**. Así, se puede generar la **dominancia por columnas** recorriendo la columna asociada al punto en la diagonal, y en caso de cumplir ambas se toma como el caso general de **dominancia diagonal**.

Teorema de Convergencia Gauss-Seidel:
Sea el sistema dado por $Ax = b$, con $A \in M_{n\times m}(F)$ y x, b vectores de tamaño $n \times 1$. Si A es estríctamente diagonalmente dominante, entonces el método de **Gauss-Seidel** converge para cualquier aproximación inicial $x^{(0)}$.

4.3.2 Justificación Teórica del Método SOR

Entonces dado un sistema $Ax = b$, con $A \in M_{n\times n}(F)$ matriz y x, b vectores de tamaño $n \times 1$. Generamos así la descomposición estándar de la matriz A:

$$A = D - L - U$$

Donde $D, L, U \in M_{n\times n}(F)$, que cumplen:

- $D =$ matriz diagonal con los elementos a_{ii}.
- $L =$ matriz triangular inferior estricta.
- $U =$ matriz triangular superior estricta.

Sabemos pues que por método Gauss-Seidel, la entrada i de $x_{GS}^{(k+1)}$ tendrá la forma:

$$x_{GS,i}^{(k+1)} = \frac{1}{aii}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\Big)$$

Donde entonces, el cambio se ve como $\Delta x_i = x_{GS,i}^{(k+1)} - x_i^{(k)}$. Ahora, la metodoloía de usar un **parámetro de relajación** ω, requiere que hagamos un cambio ponderado:

$$x_i^{(k+1)} = x_i^{(k)} + \omega\Delta x_i = x_i^{(k)} + \omega\big(x_{GS,i}^{(k+1)} - x_i^{(k)}\big)$$

Luego sustituímos el valor de $x_{GS,i}^{(k+1)}$ con la formula del método Gauss-Seidel:

$$\begin{aligned} x_i^{(k+1)} &= x_i^{(k)} + \omega\left(\frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) - x_i^{(k)}\right) \\ &= (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \end{aligned}$$

Multiplicamos todo por a_{ii} para quitar los denominadores:

$$a_{ii}x_i^{(k+1)} = (1-\omega)a_{ii}x_i^{(k)} + \omega\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

Y podemos reorganizar tal que:

$$a_{ii}x_i^{(k+1)} + \omega\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} = (1-\omega)a_{ii}x_i^{(k)} - \omega\sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + \omega b_i$$

Con lo que podemos obtener de forma matricial:

$$(D - \omega L)x^{(k+1)} = [(1-\omega)D + \omega U]x^{(k)} + \omega b$$

Y despejando finalmente:

$$x^{(k+1)} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)} + \omega(D-\omega L)^{-1}\mathbf{b}$$

En la literatura, generalmente definen:

$$T_{SOR} = (D-\omega L)^{-1}[(1-\omega)D + \omega U], \quad \mathbf{c}_{SOR} = \omega(D-\omega L)^{-1}\mathbf{b}$$

Con lo que el despeje final queda de la manera:

$$x^{(k+1)} = T_{SOR}x^{(k)} + c_{SOR}$$

> Es importante notar que por teorema, para asegurar la **convergencia** del método **SOR**, $0 < \omega < 2$, esto se puede ver en el Teorema 6.4 de *Applied Numerical Linear Algebra* de J. Demmel.

4.3.3 Algoritmo del SOR

Como vamos a hacer una mejora del método Gauss-Seidel, tomaremos como parámetro de relajación a ω, con $1 < \omega < 2$, es decir, una sobrerrelajación.

El pseudocódigo a continuación supone que la entrada corresponde a una matriz diagonalmente dominante, así como un conjunto de n-términos independientes b. Observe además que le damos criterios `max_iter` y `tol` para evitar una sobrecarga de operaciones, es decir, damos parámetros de cuántos pasos máximos y cuál es la tolerancia del error de la solución x encontrada.

4.3.3.1 Pseudocódigo

```
entrada
   A // matriz
   b // vector de terminos independientes
   x // vector de aproximación inicial
   omega // factor de relajación
   tol // tolerancia de alto
   max_iter // máximo de iteraciones

n = filas de A
iter = 0
error = tol + 1

mientras error > tol e iter < max_iter
   error = 0

   para i = 1 hasta n
      sum = 0
      xprev = x_i

      para j = 1 hasta n
         si j != i
            sum += A_ij * x_j
         fin si
      fin para


      x_i = (1-omega)*x_i + omega(b_i - sum)/a_ii

      error = MAX(error, ABS(x_i - xprev))
   fin para

   iter += 1
fin mientras

retorno x, iter, error
```

Este algoritmo dadas n y m = `max_iter` , tiene complejidad $O(mn^2)$. Con lo que así como el algoritmo pasado, suponiendo 10^9 operaciones por segundo, limitamos m, n para estar de acorde con estas. Además, es importante tener en cuenta que `tol` , estará íntimamente relacionada con el tipo de dato que almacene los valores decimales de x.

4.4. Aplicación del SOR

4.4.1. Problema 1: Sistema Diagonalmente Dominante para SOR

Se propone resolver el siguiente sistema diagonalmente dominante para demostrar la convergencia rápida del **método SOR**:

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix}$$

Solución exacta conocida: $x = [1, 1, 1]^T$

Este sistema es estrictamente diagonalmente dominante, lo que garantiza la convergencia del método SOR para $0 < \omega < 2$.

4.4.1.2 Formulación del Método SOR

Para el método SOR, la iteración para cada componente x_i es:

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

Tomaremos $\omega = 1.2$ (sobrerrelajación óptima estimada) y vector inicial $x^{(0)} = [0, 0, 0]^T$.

4.4.1.3 Desarrollo de las iteraciones

Iteración 1 ($k = 0$):

Para $i = 1$:

$$x_1(1) = (1 - 1.2) \cdot 0 + 1.24(5 - (1 \cdot 0 + 0 \cdot 0)) = 0 + 0.3 \cdot 5 = 1.5$$

Para $i = 2$:

$$x_2(1) = (1 - 1.2) \cdot 0 + 1.24(6 - (1 \cdot 1.5 + 1 \cdot 0)) = 0 + 0.3 \cdot (6 - 1.5) = 0.3 \cdot 4.5 = 1.35$$

Para $i = 3$:

$$x_3(1) = (1 - 1.2) \cdot 0 + 1.24(5 - (0 \cdot 1.5 + 1 \cdot 1.35)) = 0 + 0.3 \cdot (5 - 1.35) = 0.3 \cdot 3.65 = 1.095$$

Resultado: $x^{(1)} = [1.500, 1.350, 1.095]^T$

4.4.1.4 Convergencia e Iteraciones

Iteración	x_1	x_2	x_3	**Error máximo**
0	0.0000	0.0000	0.0000	1.0000
1	1.5000	1.3500	1.0950	0.7050
2	0.7950	0.9630	0.9921	0.2571
3	1.0521	0.9941	1.0033	0.0607

Después de solo 3 iteraciones, la aproximación está muy cerca de la solución exacta $[1, 1, 1]^T$. El error máximo absoluto en la tercera iteración es:

$$\max(|\ 1.0521 - 1\ |, |\ 0.9941 - 1\ |, |\ 1.0033 - 1\ |) = 0.0521$$

Esto demuestra la rápida convergencia del método SOR con un factor de relajación bien elegido en un sistema diagonalmente dominante.

4.4.1.5 Resultado del Código

Al ejecutar el script en Python implementado con la metodología descrita, se obtuvieron los siguientes resultados en la consola, confirmando la exactitud de la factorización y la solución del sistema:

```
=== INICIANDO MÉTODO SOR ===
omega = 1.2, tol = 0.07, max_iter = 1000
x0 = [0.0, 0.0, 0.0]


===== Iteración 0 =====
x[1] previo: 0.000000  →  nuevo: 1.500000   (cambio = 1.500000e+00)
x[2] previo: 0.000000  →  nuevo: 1.350000   (cambio = 1.350000e+00)
x[3] previo: 0.000000  →  nuevo: 1.095000   (cambio = 1.095000e+00)
Error de la iteración: 1.500000e+00
Vector x = [1.5, 1.3499999999999999, 1.095]

===== Iteración 1 =====
x[1] previo: 1.500000  →  nuevo: 0.795000   (cambio = 7.050000e-01)
x[2] previo: 1.350000  →  nuevo: 0.963000   (cambio = 3.870000e-01)
x[3] previo: 1.095000  →  nuevo: 0.992100   (cambio = 1.029000e-01)
Error de la iteración: 7.050000e-01
Vector x = [0.795, 0.963, 0.9920999999999999]

===== Iteración 2 =====
x[1] previo: 0.795000  →  nuevo: 1.052100   (cambio = 2.571000e-01)
x[2] previo: 0.963000  →  nuevo: 0.994140   (cambio = 3.114000e-02)
x[3] previo: 0.992100  →  nuevo: 1.003338   (cambio = 1.123800e-02)
Error de la iteración: 2.571000e-01
Vector x = [1.0520999999999998, 0.9941400000000002, 1.0033380000000003]

===== Iteración 3 =====
x[1] previo: 1.052100  →  nuevo: 0.991338   (cambio = 6.076200e-02)
x[2] previo: 0.994140  →  nuevo: 1.002769   (cambio = 8.629200e-03)
x[3] previo: 1.003338  →  nuevo: 0.998502   (cambio = 4.836360e-03)
Error de la iteración: 6.076200e-02
Vector x = [0.9913380000000002, 1.0027692, 0.99850164]

=== FIN DEL MÉTODO SOR ===
Solución aproximada x: [0.9913380000000002, 1.0027692, 0.99850164]
Iteraciones: 4
Error final: 0.06076199999999965
```

4.5 Resultados

El método SOR implementado resolvió exitosamente el sistema diagonalmente dominante propuesto, demostrando convergencia rápida hacia la solución exacta.

Precisión:

Tras solo 3 iteraciones, el error máximo absoluto se redujo de 1.0000 a 0.0521, acercándose significativamente a la solución exacta $[1, 1, 1]^T$. La convergencia monótona observada valida la elección del parámetro de relajación $\omega = 1.2$.

Eficiencia:

Para un sistema 3×3, cada iteración requiere solo $O(n)$ operaciones por componente, demostrando la ventaja computacional del método iterativo frente a métodos directos como eliminación Gaussiana ($O(n^3)$), especialmente para aproximaciones iniciales cercanas a la solución.

4.6 Discusión y Conclusiones

El método SOR ha demostrado ser una técnica eficiente para la resolución iterativa de sistemas lineales, *específicamente* cuando se aplica a matrices diagonalmente dominantes.

Ventajas: Su principal fortaleza radica en la flexibilidad del parámetro de relajación ω, que permite acelerar significativamente la convergencia respecto a métodos como Jacobi o Gauss-Seidel. Además, al actualizar cada componente utilizando los valores más recientes disponibles (carácter secuencial), reduce el número de iteraciones necesarias.

Limitaciones: Como se observó en la teoría, la convergencia del método depende críticamente de la elección de ω y de las propiedades espectrales de la matriz. Para matrices que no son diagonalmente dominantes o definidas positivas, puede divergir o requerir un ω subóptimo, lo que disminuye su eficiencia. Además, su naturaleza secuencial dificulta la paralelización completa del algoritmo.

En conclusión, el método SOR representa un equilibrio óptimo entre simplicidad y eficiencia para sistemas bien condicionados, siendo fundamental comprender su comportamiento para aplicarlo adecuadamente en problemas de ingeniería y cómputo científico donde los métodos directos son computacionalmente costosos.

5. Referencias

- Geeks4Geeks - Linear Algebra in Computer Science
- CS 357 Textbook - LU Decomposition for Solving Linear Equations
- MIT OpenCourseWare - LU Decomposition
- Lloyd N., David Bau - Numerical Linear Algebra: Ver **Lecture 20** (Pag. 147 - 151).
- Wikipedia - Diagonally Dominant Matrix
- Wikipedia - Gauss-Seidel Method
- J. Demmel - Applied Numerical Linear Algebra: Ver **Capítulo 6** (Pag 282-286, 290)
- Seidel, L. (1874). Ueber ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt, durch successive Annäherung aufzulösen. Münchener DigitalisierungsZentrum, Digitale Bibliothek, Bayerische Staatsbibliothek. Recuperado de https://www.digitale-sammlungen.de/de/view/bsb11179192?page=5
- Burden, R. L., & Faires, J. D. (2002). Análisis Numérico (7ª ed.). Ediciones Paraninfo, S.A. Recuperado de https://evflores.wordpress.com/wp-content/uploads/2014/02/analisis-numerico-richard-l-burden-7ma.pdf
- Sobrerrelajación sucesiva. (n.d.). En Wikipedia, la enciclopedia libre. Recuperado el 2 de diciembre de 2025, de https://es.wikipedia.org/wiki/Sobrerrelajaci%C3%B3n_sucesiva