DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Métodos iterativos para resolución de sistemas de ecuaciones

Métodos Numéricos

Joaquín Carrasco: joaquin.e.carrasco@gmail.com
Santiago Pensotti: santipensotti@gmail.com
Valentina Vul: valenvul@gmail.com

Resumen:

El objetivo de este trabajo es el análisis del método de Jacobi y el de Gauss-Seidel para la resolución iterativa de sistemas de ecuaciones. Se analizó además las convergencias de los métodos y se implementaron tanto en su forma matricial como mediante una sumatoria. Además se consideraron las ventajas que pueden llegar a tener unos sobre otros, as'i como con respecto a un método exacto como lo es la Eliminación Gaussiana.
Mediante python, usando numpy, y C++, usando la librería Eigen, se implementaron estos algoritmos y se diseñaron tests para medir y comparar, en distintos tipos de matrices, su distancia a la solución, el tiempo de cómputo y la velocidad de convergencia de los métodos y sus variantes matriciales y en formato de sumatoria.
A partir de estos se concluyó que, para matrices de alta dimensión en las que se pueda asegurar la convergencia, el método de Gauss-Seidel implementado por sumatoria presenta las mayores ventajas.

Palabras clave:

Métodos iterativos, Jacobi, Gauss-Seidel, sistemas de ecuaciones

Índice

1. Introducción Teórica

El objetivo de este trabajo es la implementación y experimentación con métodos iterativos para la resolución de sistemas de ecuaciones y la comparación con los métodos de eliminación gaussiana.

Un sistema de ecuaciones puede ser expresado de froma matricial mediante la ecuación

$$A \times x = b$$

Siendo A la matriz de $m \times n$ cuyos elementos representan los coeficientes de las ecuaciones, $x \in \mathbb{R}^n$ el vector columna con las incógnitas buscadas y $b \in \mathbb{R}^m$ el vector columna con los terminos independientes.

Hay diversos mecanismos mediante los cuales un sistema de ecuaciones puede ser resuelto. Los métodos exactos, como la eliminación gaussiana, aseguran que la solución puede ser obtenida en una cantidad finita de pasos. Usualmente, el costo computacional de estos suele ser elevado para entradas de alto tamaño por lo que en muchos casos resulta preferible la utilización de métodos iterativos. Estos procedimientos plantean una sucesión de ecuaciones que, bajo ciertas circunstancias, convergen a la solución del sistema en una menor cantidad de pasos.

Estos métodos siguen la forma

$$x^{k+1} = Tx^k + c$$

En la que x^k representa el resultado de la k-esima iteración de la sucesión y T es la denominada matriz dominante de la iteración. De esta froma, al evaluar esta ecuación cuando k tiende a infinito, la x debería converger a la solución del sistema.

Dado un sistema de ecuaciones, la convergencia de cierta sucesión en particular puede ser garantizada para cualquier punto de inicio si el radio espectral de la matriz dominante de la iteración es menor a 1. Se denomina radio espectral de una matriz al valor del máximo módulo de los autovalores de esta.

$$\rho(T) = max\{|\lambda| : \lambda \text{ autovalor de } T\} \tag{1}$$

Esto se debe a unas propiedades que establecen que si el radio espectral de una matriz es menor a 1, se cumple que $\sum_{k=0}^{\infty} T^k = (I - T)^{-1}$, y dicha matriz es una matriz convergente, es decir $\lim_{k\to\infty} T^k = 0$. De esta forma, al expandir la ecuación de la iteración y tomar su limite

$$x_{k+1} = T^{k+1}x_0 + T^k c + \cdots + Tc + c$$

$$\xrightarrow[k\to\infty]{} x = 0 + (I - T)^{-1}c$$

$$x = Tx + c$$

La ecuación converge a la solución del sistema.

En este trabajo se van a evaluar dos métodos especificos, el de Jacobi y el de Gauss-Seidel. Ambos métodos pueden ser expresados de forma matricial, siguiendo la ecuación antes presentada, asi como en forma de sumatorias. Estas segundas se efectúan sobre cada coordenada particular del vector solución.

El metodo de jacobi está definido como

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1,\, j\neq i}^{n} a_{ij} x_j^k \right) \tag{2}$$

Este método toma cada coordenada por separado y la aproxima a la solución del sistema. Como se basa en divisiones por el elemento de la diagonal, esta sucesión solo está definida para matrices cuyos elementos en la diagonal principal sean no nulos.

Para definir su forma matricial, es necesario descomponer la matriz de coeficientes del sistema de la siguiente forma

$$A = D - L - U \tag{3}$$

Con D diagonal, L triangular inferior con ceros en la diagonal, U triangular superior con ceros en la diagonal

$$D = \begin{bmatrix} a_{11} & & & & 0 \\ & a_{22} & & & \\ & & \ddots & & \\ & & & \ddots & \\ 0 & & & & a_{n\,n} \end{bmatrix}, \; L = \begin{bmatrix} 0 & & & & 0 \\ -a_{21} & 0 & & & \\ \vdots & \vdots & \ddots & & \\ -a_{n-11} & -a_{n-12} & \dots & 0 & \\ -a_{n1} & -a_{n2} & \dots & -a_{nn-1} & 0 \end{bmatrix},$$

$$U = \begin{bmatrix} 0 & -a_{12} & \dots & -a_{1n-1} & -a_{1n} \\ & 0 & \dots & -a_{23} & -a_{2n} \\ & & \ddots & \vdots & \vdots \\ & & & 0 & -a_{n-1n} \\ 0 & & & & 0 \end{bmatrix},$$

Luego, al reemplazar A en la ecuación original del sistema y operar se llega a la expresión matricial de la ecuación antes dada

$$\begin{aligned} Ax &= b \\ (D - L - U)x &= b \\ Dx &= (L + U)x + b \\ x &= (D)^{-1}(L + U)x + (D)^{-1}b \end{aligned} \tag{4}$$

Al requerir que la matriz original no tenga ceros en la diagonal, D siempre es inversible y estas operaciones están definidas.

El método de Gauss-Seidel se define de forma similar, pero en vez de aproximar cada coordenada de un paso iterativo por separado, se basa en las componentes previamente calcualadas para ese paso. De esta forma su sucesión queda definida de la siguiente manera

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right), \;\; i = 1, 2, \cdots, n \tag{5}$$

Tomando la misma descomposición que para el método anterior, se puede llegar a una ecuacion matricial que cumpla con al estructura de los métodos iterativos

$$\begin{aligned} Ax &= b \\ (D - L - U)x &= b \\ (D - L)x - Ux &= b \\ (D - L)x &= Ux + b \\ x &= (D - L)^{-1}(Ux + b) \\ x &= (D - L)^{-1}Ux + (D - L)^{-1}b \end{aligned} \tag{6}$$

Nuevamente se puede observar por qué para la aplicación de esta técnica también es necesario que la matriz original no presente ceros en su diagonal.

2. Desarrollo

2.1. Consideraciones

La convergencia de los métodos es evaluada en todas las iteraciones mediante el cálculo de la distancia euclidiana entre la iteración anterior y la actual, si esta distancia se encuentra por debajo de un umbral entonces se concluye que se convergió puesto que $x^{k+1} \approx x^k$.

Se consideró la opción de averiguar qué sucede con el radio espectral de la matriz de iteración para sacar conclusiones sobre la convergencia, pero el incumplimiento de la condición $\rho(T) < 1$ no es razón suficiente para asegurar la divergencia. Es decir que se podría determinar que el método diverge cuando para algún x^0 podría converger. De todas maneras esta propiedad fue utilizada en los testeos para la creación de matrices que convergieran.

En las cuatro implementaciones se usaron funciones propias de C++ para poder calcular el tiempo de cada método. Además se guardaron la diferencia de las normas entre cada paso, cuál fue la iteración en la que covergió y si lo hizo.

2.2. Método Jacobi Matricial

Lo primero que se debe hacer para implementar este método es calcular la matriz de iteración de Jacobi. Para esto antes es necesario descomponer la matriz del sistema de ecuaciones en su forma $A = D - L - U$ y luego en base a esto hacer las operaciones para obtener la ecuación 4. Luego es necesario inicializar el vector sobre el que se va a iterar y el que guardará el valor de la iteración previa necesario para verificar la convergencia.

Una vez obtenidos todos los operandos necesarios, se puede comenzar a iterar. Mediante las operaciones matriciales indicadas por la ecuación que define el método, se calcula el valor del vector completo. Por último, en cada iteración se verifica si se llegó a la convergencia y una vez que lo hizo se detiene la ejecución. Si esta no se logra, el ciclo se hará la cantidad de veces indicada por *iter* y se asumirá que, comenzando con $x_0 = 0$, el sistema no converge.

Algorithm 1 Metodo Jacobi Matriz *(in: A matriz, in: b vector, iter, tol, λ)*

```
D ← diagonal(A)
U ← −triangularSuperior(A)
L ← −triangularInferior(A)
T ← D.inversa() ∗ (UL)
c ← D.inversa() ∗ b
x ← vector(0, size(b))
x_ant ← vector(0, size(b)
for i = 0 to iter do
    x =T * x + c;
    norma2 ← ||x − ∗x_ant||_2
    if abs(norma2) < tol then
        break
    end if
    x_ant =x
end for
Return x
```

2.3. Método Jacobi por sumatoria

Al calcular la solución del sistema de ecuaciones mediante la sumatoria, no se opera con matrices, sino que se trabaja componente por componente con los valores de la matriz dada.

Primero se inicializan en cero el vector sobre el que se va a iterar, así como el que va a almacenar el valor de la iteración previa . Luego se realiza la sumatoria correspondiente a la ecuación 1 y, en base a este resultado se hacen el resto de las operaciones de la ecuación. Por último se verifica la convergencia con la técnica antes descripta, y se frena la ejecución si esta fue conseguida.

Algorithm 2 Metodo Jacobi Sumatoria *(in: A matriz, in: b vector, iter, tol, λ)*

```
1: x ← vector(0, size(b)
2: x_ant ← vector(0, size(b)
3: for i = 0 to iter do
4:     for j = 0 to x.size() do
5:         sumatoria =0;
6:         for k = 0 to x.size() do
7:             if k ≠ j then
8:                 sumatoria += A(j,k) * xk[k]
9:             end if
10:        end for
11:        x[k] = 1/A(j, j) ∗ (b[j] − sumatoria
12:    end for
13:    norma1 ← ||x − ∗x_ant||_2
14:    if abs(norma1) < tol then
15:        break
16:    end if
17:    x_ant =x
18: end for
19: Return x
```

2.4. Método Guass-Seidel Matricial

De manera similar a lo realizado en el método de jacobi, lo primero que se debe hacer es la decomposición de la matriz con los coeficientes de las ecuaciones. Luego, a partir de estas, se calcula la matriz de la iteración correspondiente a la ecuación 6, así como el término independiente. Además se inicializan en cero tanto el vector sobre el que se va a iterar, como el que va a almacenar el valor de la iteración previa.

Una vez hechos todos los pasos previos, se puede comenzar a iterar. Esto se hace mediante operaciones matriciales, calculando el vector completo en cada iteración según lo indica la ecuación correspondiente. Al final de cada iteración se verifica si se llegó a la convergencia y, de ser así, detiene la iteración y se devuelve la solución encontrada.

Algorithm 3 Metodo Gauss Matriz *(in: A matriz, in: b vector, iter, tol, λ)*

1: $D \leftarrow diagonal(A)$
2: $U \leftarrow -triangularSuperior(A)$
3: $L \leftarrow -triangularInferior(A)$
4: $DL \leftarrow D - L$
5: $T \leftarrow DL.inversa() * U$
6: $c \leftarrow DL.inversa() * b$
7: $x \leftarrow vector(0, size(b)$
8: $x_{ant} \leftarrow vector(0, size(b)$
9: **for** $i = 0$ **to** $iter$ **do**
10: $x =$T * x + c;
11: $norma1 \leftarrow ||x - *x_{ant}||_2$
12: **if** $abs(norma1) <$ tol **then**
13: **break**
14: **end if**
15: $x_{ant} =$x
16: **end for**
17: **Return** x

2.5. Método Gauss-Seidel Sumatoria

Al igual que en el método anterior, la diferencia principal entre la forma matricial y la de la sumatoria para el método de Gauss-Seidel es que en esta segunda la solución se calcula componente a componente, en vez de operar matricial y vectorialmente sobre todo el dato. Esto permite que no sea necesario descomponer la matriz original y se pueda operar directamente con sus elementos.

De la misma forma que en los algoritmos descriptos previamente, se comienza inicializando el vector sobre el que se iterará y el que se usa para la comparación que define la convergencia o no de la sucesión. Luego, siguiendo la ecuación 4, se calcula a sumatoria. Esta se calcula en base a la iteración previa para los elementos de indice mayor al que se esta calculando y en base a lo calculado en dicha iteración para los de indice menor. El elemento de la diagonal no forma parte de la sumatoria por lo que este se saltea. Después se realizan las operaciones restantes como lo indica la definición de la sucesión en base a dicha sumatoria.

Por último, se verifica la convergencia tal como fue descripto previamente, mediante la comparación de la iteración actual con la previa, y en base a eso se detiene o no la ejecución.

Algorithm 4 Gauss-Seidel Sumatoria *(in: A matriz, in: b vector, iter, tol, λ)*

```
1: x ← vector(0, size(b)
2: x_ant ← vector(0, size(b)
3: for i = 0 to iter do
4:     for j = 0 to x.size() do
5:         sumatoria =0;
6:         for k = 0 to x.size() do
7:             if k < j then
8:                 sumatoria += A(j,k) * x[k]
9:             end if
10:            if k > j then
11:                sumatoria += A(j,k) * x_ant[k]
12:            end if
13:        end for
14:        x[k] = 1/A(j, j) ∗ (b[j] − sumatoria
15:    end for
16:    norma1 ← ||x − ∗x_ant||_2
17:    if abs(norma1) < tol then
18:        break
19:    end if
20:    x_ant =x
21: end for
22: Return x
```

2.6. Testeo

Para comprobar el correcto funcionamiento de los algoritmos se tomaron distintos sistemas de ecuaciones con resultados conocidos y se verificó que la solución calculada por las implementaciones coincida con estos. Se realizaron varios tests con matrices estrictamente diagonal dominante ya que en estos casos ambos métodos deben converger. Para esto, se generaron matrices aleatorias de distintos tamaños que presentaran dicha estructura y se las multiplicó por un vector aleatorio que sería la solución, generando asi el vector con los terminos independientes. Luego se comparó esta solución con el resultado parcial de los algoritmos.

Además se analizaron casos en los que solo convergería Gauss-Seidel, y en los que ninguno converge. De esta forma también se pudo verificar el comportamiento correcto de los indicadores de convergencia y de divergencia implementados en los algoritmos.

3. Resultados y Discusión

Una vez implementados los algoritmos se realizaron diversos experimentos para analizar y comparar el error generado y el tiempo de ejecución de los métodos. Los resultados de estos fueron ilustrados en gráficos, denotando el promedio de los resultados dados por 50 matrices diferentes. Los algoritmos se corrieron usando una tolerancia de 1e-8 y con una cantidad máxima de iteraciones de 10000.
Los tiempos se midieron en C++ utilizando std::chrono::high_resolution_clock Y el error se evalúo midiendo la distancia euclideana.
Los gráficos 1 y 2 permiten ver cómo se comparan las complejidades temporales de los algoritmos implementados, así como en contraste con un método exacto como la descomposición LU. El primero de estos muestra el comportamiento del tiempo de ejecución de todos estos algoritmos en el caso que las matrices sean estrictamente diagonal dominante.

Esta figura permite observar cómo el método exacto aumenta exponencialmente su tiempo de ejecución a medida que el tamaño de la entrada es más grande. Es por esto que resulta conveniente usar métodos iterativos para sistemas de mayor dimensión, siempre y cuando se pueda garantizar su convergencia. De todas formas, al tener que hacer operaciones matriciales todas las iteraciones, la forma matricial de ambas técnicas resulta poco ventajosa en comparación al algoritmo implementado por sumatorias .

Al comparar ambos métodos iterativos, se ve que el método de Gauss-Seidel requiere menos tiempo que el de Jacobi en las dos formas presentadas. Es así que se podría concluír que, cuando se trate de una entrada de gran magnitud y ambos métodos lleguen a la solución adecuada, el algoritmo más conveniente será el de Gauss-Seidel. Y de tratarse de matrices estrictamente diagonal dominantes, el método de Gauss-Seidel implementado con sumatoria es el más adecuado.

El segundo gráfico fue ilustrado de la misma manera que el anterior, pero en este caso las matrices evaluadas eran simétricas definidas postivas. Estas matrices tienen la particularidad de que el método de Jacobi no converge a la solución del sistema.

Esta segunda figura exhibe la supremacía de Gauss-Seidel en matrices simétricas definidas positivas. Además muestra una superioridad del método matricial por sobre el de sumatorias, diferenciándose también en ese aspecto con la primera figura.

Por trabajar con matrices aleatorias, la distribución de los tiempos de ejecución en cada dimensión toma interés. Para exhibir las distribuciones en cada dimensión se utilizaron boxplots. Estos gráficos permiten visualizar la variabilidad y la tendencia central de los tiempos de ejecución. La figura 3 está alineada a lo concluído previamente, pues las sumatorias son mejor que la contraparte matricial, y Gauss-Seidel, aunque por poco, logra marcarse mejor.

Se evaluó también la distribución de los tiempos en simétricas definidas positivas. En la figura 4 está eso. La divergencia asegurada de jacobi justifica la poca varianza en su distribución. Como en todas las operaciones va a llegar hasta la máxima iteración, tiene sentido que su tiempo no varíe mucho. En cambio el método de Gauss-Seidel está muy distribuído en el eje y, que tiene sentido considerando que converge siempre.

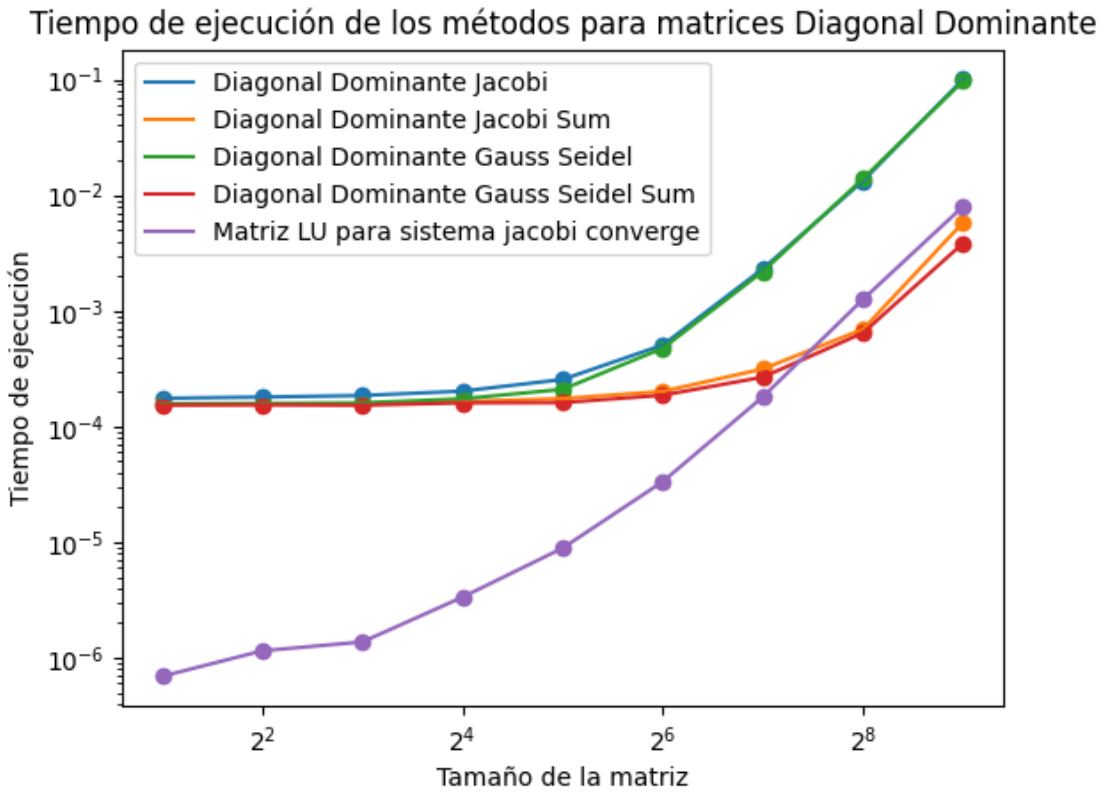

Figura 1: Tiempo de ejecución para matrices EDD en función del tamaño de la entrada

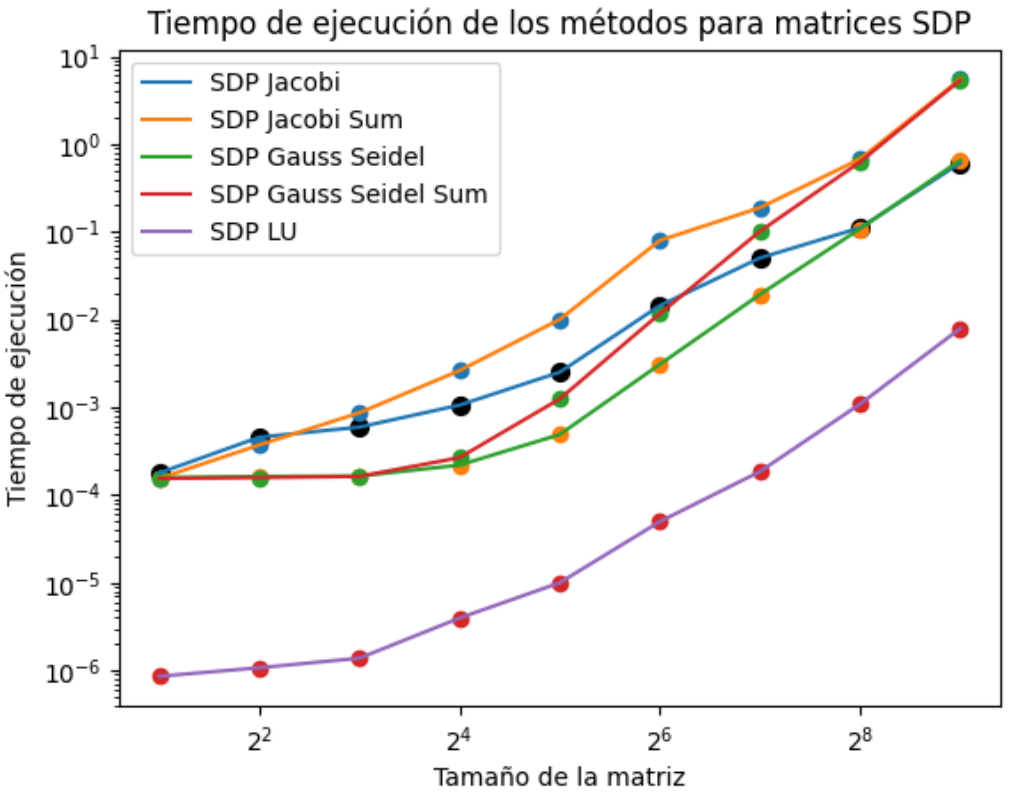

Figura 2: Tiempo de ejecución para matrices SDP en función del tamaño de la entrada

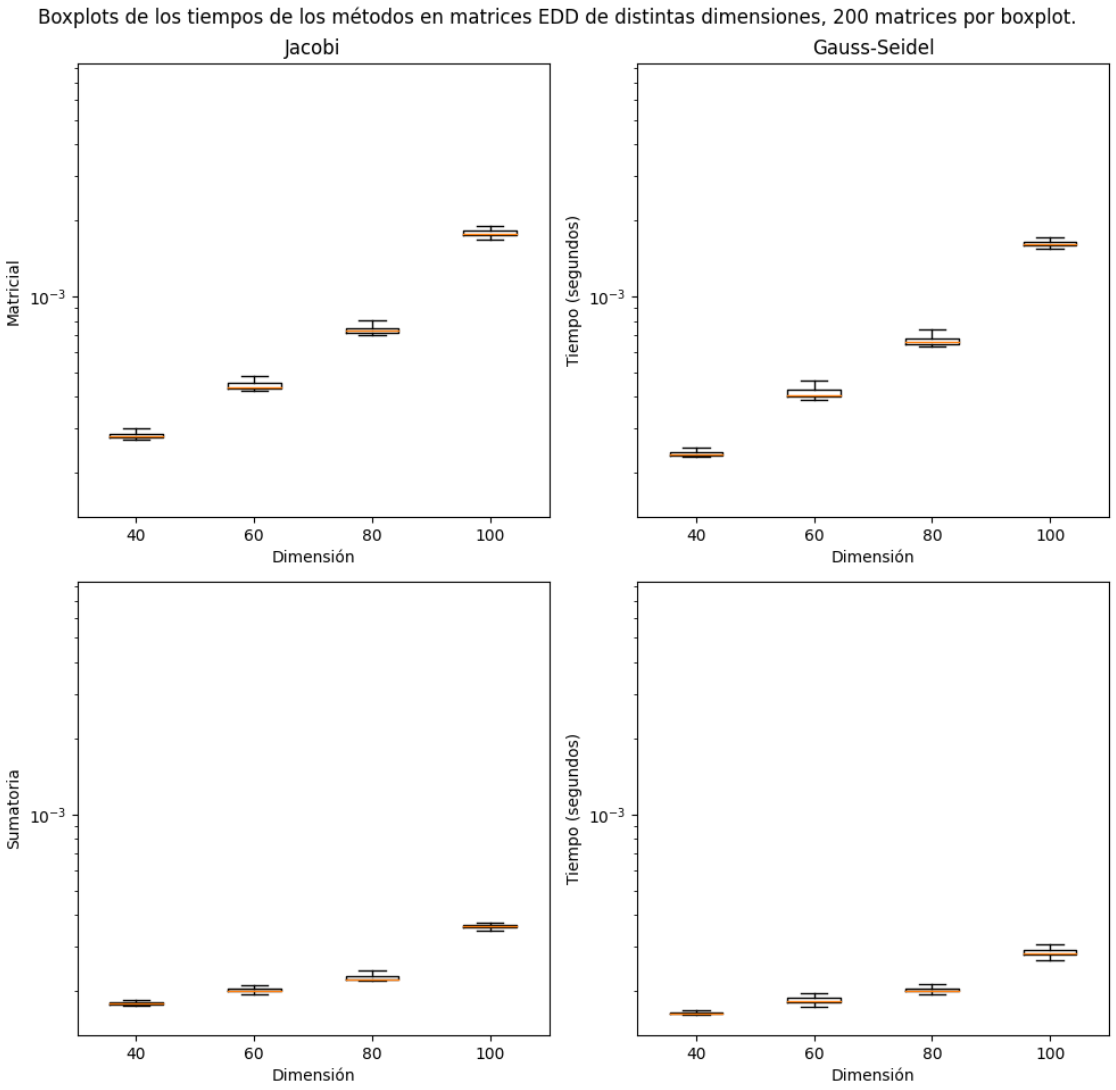

Figura 3: Distribuciones de los tiempos de ejecución sobre matrices estrictamente diagonal dominantes con respecto a su dimensión, método e implementación.

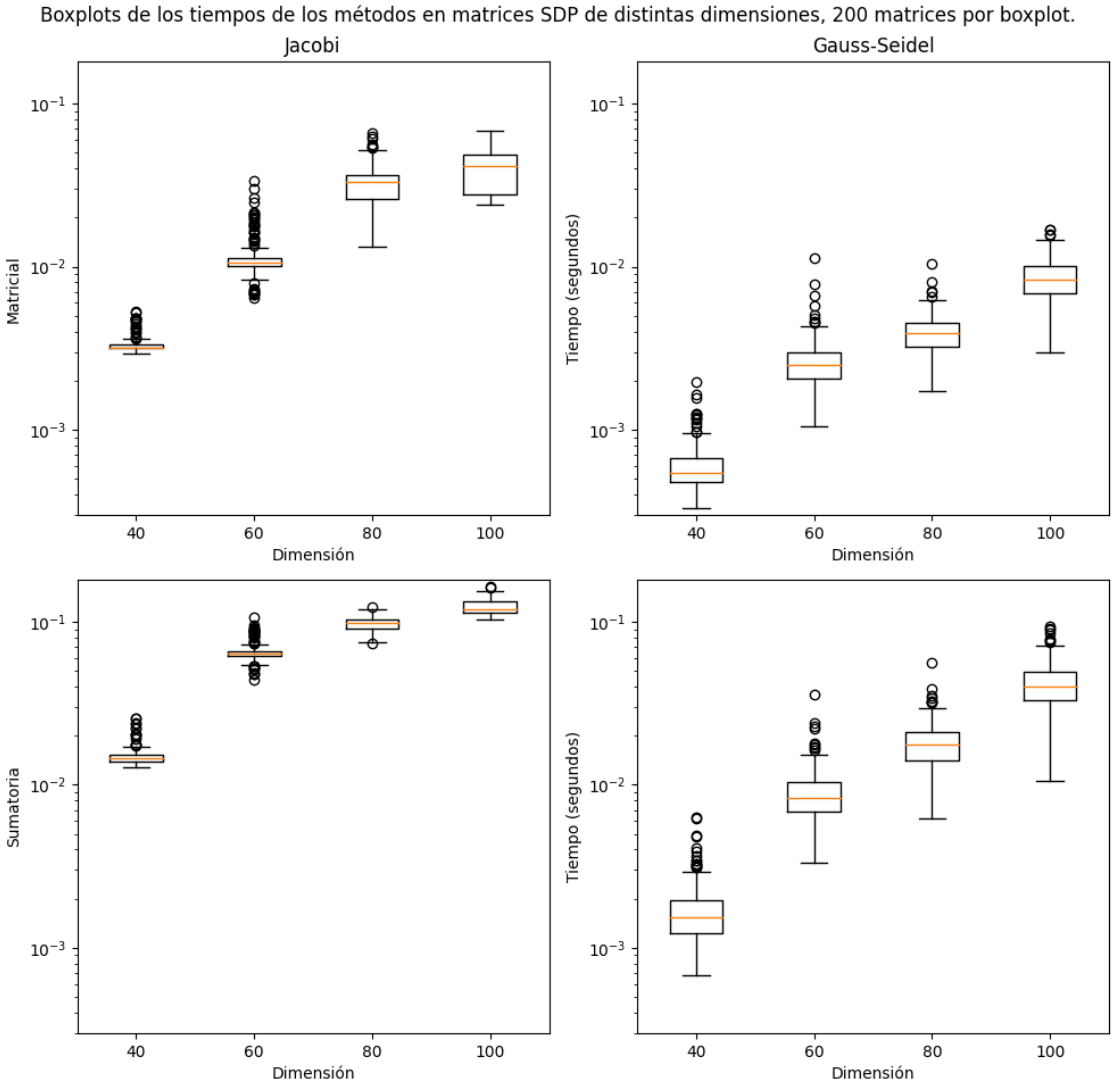

Figura 4: Distribuciones de los tiempos de ejecución sobre matrices simétricas definidas positivas con respecto a su dimensión, método e implementación.

En cuanto al error númerico, solo es interesante evaluarlo en los casos en los que los métodos convergen ya que si no lo hacen este no nos brinda información sobre la cálidad de los resultados de los algoritmos. Es por esto que para generar los gráficos 5 y 6 solo se evaluaron matrices de la forma estrictamente diagonal dominante.

En el gráfico 5 se puede ver la evolución del error numérico para los 4 algoritmos en base a la cantidad de iteraciones realizadas. Lo primero que se puede derivar de dicha imágen es el hecho de que, en cuanto a la exactitud de la respuesta, no hay ventaja en usar la forma matricial sobre la sumatoria y viceversa, las dos formas del método converjen con la misma cantidad de iteraciones. Por otro lado, este gráfico refleja que, en los casos en los que ambas técnicas converjen, es preferible utilizar Gauss-Seidel ya que esta lo hace en menos iteraciones.

El otro gráfico (figura 6) ilustra la cercanía de la respuesta calculada por los algoritmos a la verdadera solución del sistema en base al tamaño de la entrada. Como todos los métodos concluyen en la misma respuesta, el error de todos crece al mismo ritmo. De esta forma el error numérico no sirve como parámetro para decidir cual es el método mas conveniente en el caso de una matriz E.D.D ya que no hay diferencia alguna. La distribución del error fue dispuesta en la figura 7. El

error era el mismo para todos y se puede ver a simple vista la distribución uniforme de este.

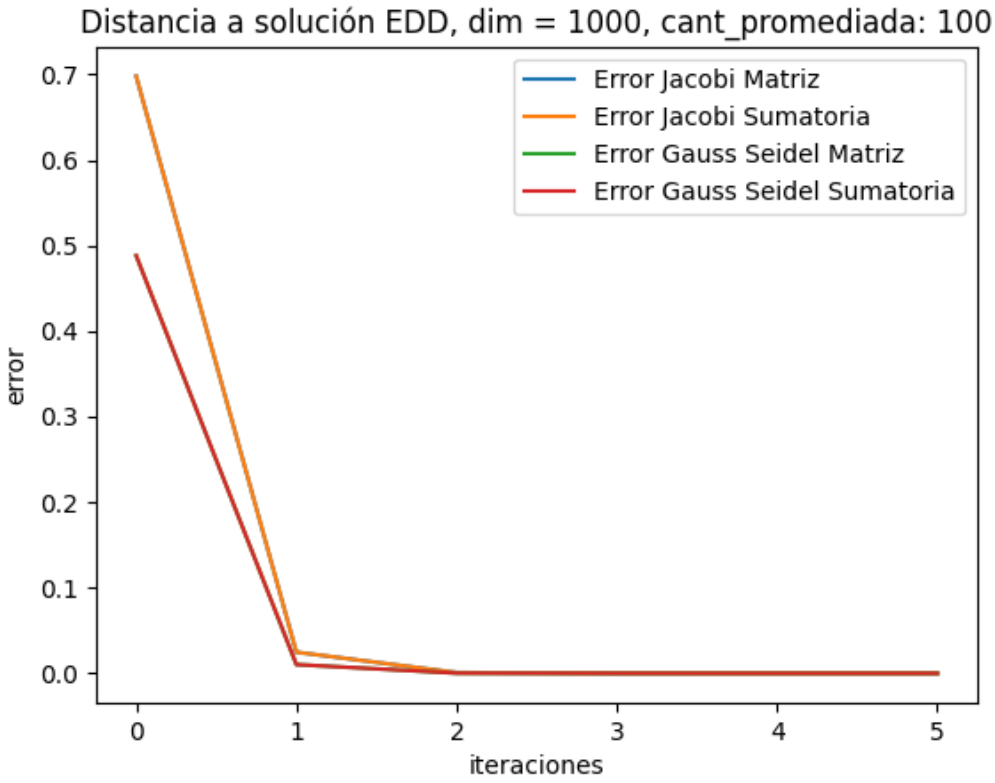

Figura 5: Error numérico en base a la cantidad de iteraciones.

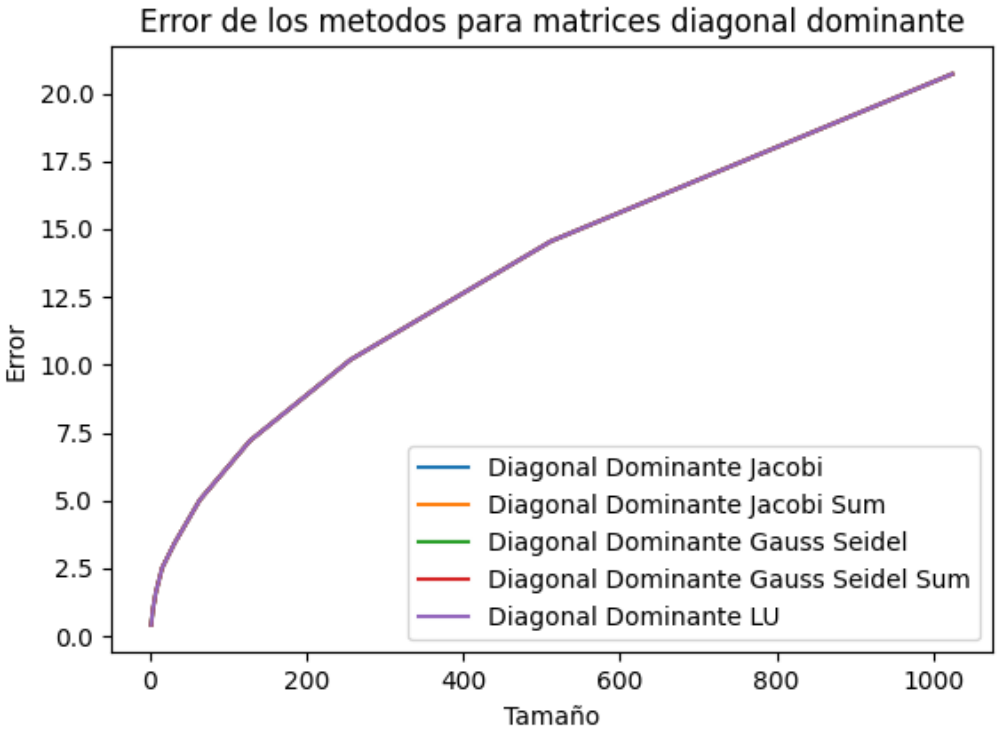

Figura 6: Error en base al tamaño de la entrada.

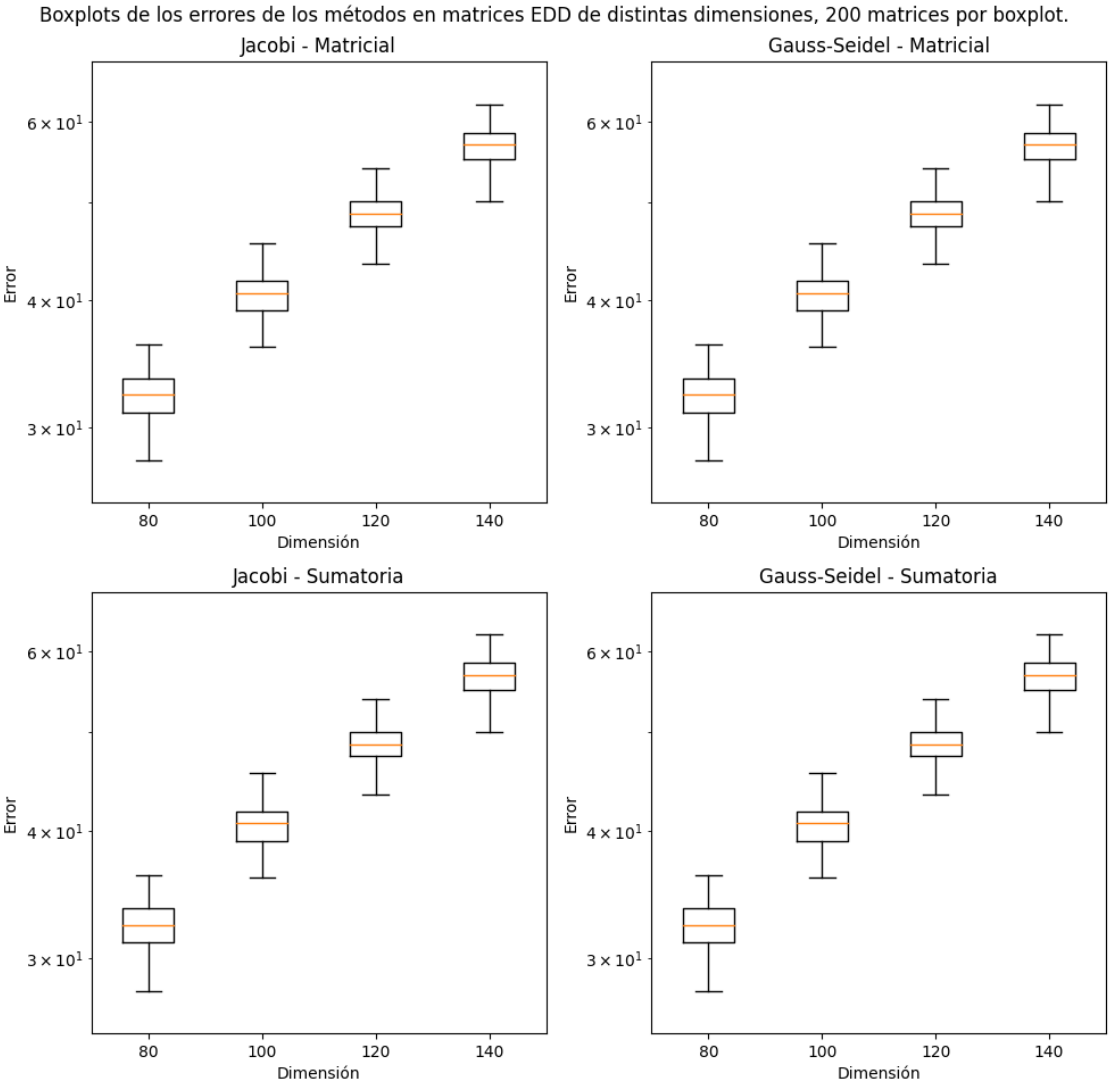

Figura 7: Distribución del error.

4. Conclusiones

Los métodos iterativos presentan una alternativo a los exactos en los casos en los que la complejidad temporal de estos resulta un inconveniente. Mediante experimentos y análisis se concluyo que en los casos donde se pueda asegurar la convergencia, el método de Gauss-Seidel implementado mediante una sumatoria es el que más ventajoso resulta en cuestión temporal. Aún así, si no se puede garantizar la convergencia de este pero sí del de Jacobi, la implementación no matricial de este sigue resuntando mas beneficioso que el método exacto.

Estos beneficios solo se presentan en casos especificos, en los que se requiere que tanto la matriz con los coeficientes del sistema de ecuaciones como la matriz de la iteración cumplan determinadas características. En la mayoría de los usos los métodos exactos siguen siendo preferibles a los presentados en este trabajo.

5. Bibliografía

1. Capítulo 7. Burden, R. L., Faires, J. D., & Burden, A. M. (2017). Análisis numérico (10.ª ed.). Cengage Learning.