DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP3 - Métodos iterativos para resolución de sistemas de ecuaciones

7 de julio de 2023 Métodos Numericos

Integrante	LU	Correo electrónico
Collasius, Federico	164/20	fede.collasius@gmail.com
Memoli Buffa, Pedro	376/20	demnitth@gmail.com
Schwartzmann, Alejandro Ezequiel	390/20	a.schwartzmann@hotmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
https://exactas.uba.ar

RESUMEN:

Índice

1. Introducción

En este trabajo vamos a implementar dos métodos iterativos de dos formas distintas, para la resolución de sistemas de ecuaciones lineales. Estos métodos ofrecen una alternativa eficiente y flexible para encontrar soluciones **aproximadas** a sistemas de ecuaciones.

Con estos métodos, abordamos el problema de resolver sistemas de ecuaciones lineales de la forma $Ax = b$ con $A \in R^{nxn}$, una matriz cuadrada de coeficientes conocida y $b \in R^n$, un vector de términos independientes.

Además de implementar, buscamos experimentar y testear estos métodos, particularmente nos interesa medir el error que tienen, el tiempo que les lleva encontrar una solución aproximada fehaciente y la convergencia. Esto lo logramos comparando con el resultado esperado, $Ax = b \iff x = A^{-1}b$, y además utilizamos la descomposición LU para resolver directamente el sistema de ecuaciones. Con la solución esperada y la descomposición LU tenemos una referencia para poder calificar a estos métodos.

Luego utilizando los resultados de las experimentaciones buscamos concluir sobre la eficiencia y fidelidad de estos métodos.

2. Métodos

Como previamente definimos buscamos encontrar una solución aproximada $x \in R^n$ tal que $Ax = b$, con A y b conocidos.

Para lograr encontrar la solución buscada utilizamos los métodos iterativos de **Jacobi** y **Gauss-Seidel**.

Estos dos métodos utilizan un $x^0 \in R^{nxn}$ inicial para iniciar el proceso iterativo, que puede ser arbitrario, por ejemplo $x_i^{(0)} = 0$ for $i = 1, 2, \ldots, n$. Aunque también se puede usar una estimación de la posible solución. Ambos métodos pueden ser implementados de dos maneras, usando una expresión matricial o basándose en sumatorias.

Estos cuatro procedimientos que desarrollaremos a continuación, se iteran hasta que los cambios realizados de una iteración a la otra sean inferiores a una tolerancia, o hasta que la cantidad de iteraciones realizadas supere la previamente definida. Lo que ocurra primero.

Previo a explicar la idea detrás de cada método debemos aclarar ciertas notaciones. La forma matricial de ambos métodos, se basa en la igualdad de $A = D - L - U$, con D la diagonal de A, U los elementos de A desde la diagonal hacia arriba y L, los elementos de A desde la diagonal hacia abajo.

$d_{ij} = 0\ \forall j \neq i \wedge d_{ii} = a_{ii}$. $D \in R^{nxn}$ matriz diagonal

$l_{ij} = 0\ \forall i \geq j \wedge l_{ij} = a_{ij}\ \forall i < j$. $D \in R^{nxn}$ matriz triangular inferior.

$u_{ij} = 0\ \forall i \leq j \wedge l_{ij} = a_{ij}\ \forall i > j$. $D \in R^{nxn}$ matriz triangular superior.

El vector $\mathbf{x}^{(0)}$ denota nuestra estimación inicial de $\mathbf{x}$. Notamos a $\mathbf{x}^{(k)}$ como la k-esima aproximación de x o la k-esima iteración de $\mathbf{x}$, y $\mathbf{x}^{(k+1)}$ es la próxima iteración dé $\mathbf{x}$.

2.1. Método Jacobi

2.1.1. Formula Basada en Matrices

Dada la ecuación $Ax = b$, podemos descomponer la matriz A en una matriz diagonal D y las matrices triangular inferior L y triangular superior U:

$$Ax = b$$

$$(D - L - U)x = b.$$

Multiplicando los términos, llegamos a:

$$Dx - Lx - Ux = b.$$

Sumando Lx y Ux a ambos lados, se obtiene:

$$Dx = b + (L + U)x.$$

Multiplicando ambos lados por D^{-1}, se tiene:

$$D^{-1}Dx = D^{-1}(b + (L + U)x).$$

Llegamos a la expresión final:

$$x^{(k+1)} = D^{-1}(b + (L + U)x^{(k)}).$$

Esta igualdad es el método iterativo Jacobi solamente trabajando con la matriz, sin accesos a posiciones de ella.

2.1.2. Formula Basada en Sumatorias

La fórmula Basada en Sumatorias para cada fila i, que representa una ecuación, es la siguiente:

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

Esta expresión define el método iterativo Jacobi, trabajando con elementos de la matriz.

2.2. Método Gauss-Seidel

2.2.1. Formula Basada en Matrices

Nuevamente, queremos ver una expresión de Gauss-Seidel, solamente usando matrices. Partimos de:

$$A\mathbf{x} = \mathbf{b}$$

Definimos $L_* = (D - L)$

$$(L_* - U)\mathbf{x} = \mathbf{b}$$

Al distribuir la multiplicación se tiene:

$$L_*\mathbf{x} - U\mathbf{x} = \mathbf{b}$$

Al restar $U\mathbf{x}$ de ambos lados de la ecuación, se obtiene:

$$L_*\mathbf{x} = \mathbf{b} - U\mathbf{x}$$

La fórmula basada en matrices del método de Gauss-Seidel resuelve el lado izquierdo de esta expresión para $\mathbf{x}$, utilizando el valor anterior de $\mathbf{x}$ en el lado derecho. Analíticamente, esto se puede escribir como:

$$\mathbf{x}^{(k+1)} = L_*^{-1}(\mathbf{b} - U\mathbf{x}^{(k)})$$

2.2.2. Formula Basada en Sumatorias

La fórmula Basada en Sumatorias establece que los elementos de $\mathbf{x}^{(k+1)}$ pueden calcularse secuencialmente para cada fila i utilizando la sustitución hacia adelante:

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

Esta fórmula utiliza para actualizar el valor, todos los valores estimados x_i más recientes.

3. Implementación

Para simplificar el pseudocódigo, y como los dos métodos difieren en solamente una línea de código, desarrollamos los dos métodos en una sola función para cada caso. El algoritmo sabe cuál método usar a partir del parámetro $Metodo$.

Para ambas funciones los parámetros son los mismos. Reciben una matriz, A un vector b, un vector x^0 que será nuestra primera suposición para él x. El parámetro que limita las iteraciones del algoritmo es $nIter$. Este define cuantas iteraciones haremos como máximo en caso de no ser terminado por alguna de las condiciones definidas por $threshold$, $check$ y $divThreshold$. La verificación de norma, divergencia y análisis de terminación del algoritmo, ya que $x^k \approx x^{x+1}$, ocurren cada $check$ iteraciones.

En caso de que la norma 2 de la diferencia entre el x actual y el anterior sea menor que $threshold$, la iteración acaba, ya que no requerimos mayor precisión.

Por otro lado, nos interesa saber si la iteración está divergiendo. Para eso verificamos que si sucesivas iteraciones, consecutivas, tienen una estimación cada vez con mayor norma, esto implica que el algoritmo está divergiendo y, por lo tanto, terminamos la ejecución. Esta verificación la logramos utilizando $divThreshold$ como cota máxima de la cantidad de iteraciones que pueden aumentar la norma de x^k sin terminar la ejecución.

3.1. Jacobi/Gauss-Seidel - Matricial

Algorithm 1 Resuelve el sistema Ax = b de forma iterativa matricial.

procedure MetodoIterativoMatricial(A, b, x^0, $nIter$, $threshold$, $check$, $divThreshold$, $Metodo$)
    $D \leftarrow Diagonal(A)$
    $L \leftarrow EstrictamenteTriangularInferior(A) * (-1)$
    $U \leftarrow EstrictamenteTriangularSuperior(A) * (-1)$
    if $Metodo == Jacobi$ **then**
        $R \leftarrow D^{-1} \cdot (L + U)$
        $c \leftarrow D^{-1} \cdot b$
    else
        $R \leftarrow (D - L)^{-1} \cdot U$
        $c \leftarrow (D - L)^{-1} \cdot b$
    end if
    $x^k \leftarrow x^0$
    $normaPrevia \leftarrow ||x^k||_2$
    $divChecker \leftarrow 0$
    for $k = 0$ to $nIter$ **do**
        $x^{k+1} \leftarrow R \cdot x^k + c$
        if $||x^{k+1} - x^k||_2 < threshold$ **then**
            break
        end if
        if $k \,\% check = 0$ **then**
            $normaActual \leftarrow ||x^{k+1}||_2$
            if $normaActual > normaPrevia$ **then**
                $divChecker \leftarrow divChecker + 1$
                if $divChecker > divThreshold$ **then**
                    “ALERTA: El método parece estar divergiendo.”
                    break
                end if
            else
                $divChecker \leftarrow 0$
            end if
            $normaPrevia \leftarrow normaActual$
        end if
        $x^k \leftarrow x^{k+1}$
    end for
    return x^{k+1}

3.2. Jacobi/Gauss-Seidel - Basada en Sumatorias

Algorithm 2 Resuelve el sistema Ax = b de forma iterativa como sumatoria.

procedure METODOITERATIVOSUMATORIA(A, b, x^0, $nIter$, $threshold$, $check$, $divThreshold$, $Metodo$)
  $n \leftarrow filas(A)$
  $x^k \leftarrow x^0$
  $normaPrevia \leftarrow ||x^k||_2$
  $divChecker \leftarrow 0$
  for $k = 0$ to $nIter$ **do**
    for $i = 0$ to n **do**
      if Metodo == Jacobi **then**
        $x_{k+1}[i] \leftarrow (b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} \cdot x_j^k)/a_{ii}$
      else
        $x_{k+1}[i] \leftarrow (b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} \cdot x_j^k)/a_{ii}$
      end if
    end for
    if $||x^{k+1} - x^k||_2 < threshold$ **then**
      break
    end if
    if $k \% check = 0$ **then**
      $normaActual \leftarrow ||x^{k+1}||_2$
      if $normaActual > normaPrevia$ **then**
        $divChecker \leftarrow divChecker + 1$
        if $divChecker > divThreshold$ **then**
          "ALERTA: El Método parece estar divergiendo."
          break
        end if
      else
        $divChecker \leftarrow 0$
      end if
      $normaPrevia \leftarrow normaActual$
    end if
    $x^k \leftarrow x^{k+1}$
  end for
  return x^{k+1}

4. Experimentación

Para generar casos de test construimos matrices con diagonal dominante, ya que este tipo de matrices son no singulares, con lo cual existe la inversa y podemos obtener el resultado esperado $x = A^{-1}.b$ para comparar con el resultado provisto por los algoritmos y así obtener el error. Realizamos 100 experimentos para cada dimensión de matriz, con el tamaño variando desde 10x10 hasta 2500x2500 con saltos discretos en la dimensión, para así obtener una muestra suficientemente grande de ejecuciones del algoritmo que nos permite realizar un análisis correcto de los tiempos de ejecución.

Además de ejecutar nuestras implementaciones de Gauss-Seidel y Jacobi, ejecutamos un algoritmo de eliminación gaussiana, nativo de la librería Eigen para comparar performance con nuestros algoritmos, ya que la eliminación gaussiana provee una solución directa al problema, a diferencia de Jacobi/Gauss-Seidel que proveen estimaciones.

Los valores de los parámetros que usamos para la experimentación fueron 0.0001 para *threshold*, (*niter* - 1)/10 para *check* y 5 para *divThreshold*.

4.1. Tiempo de Cómputo

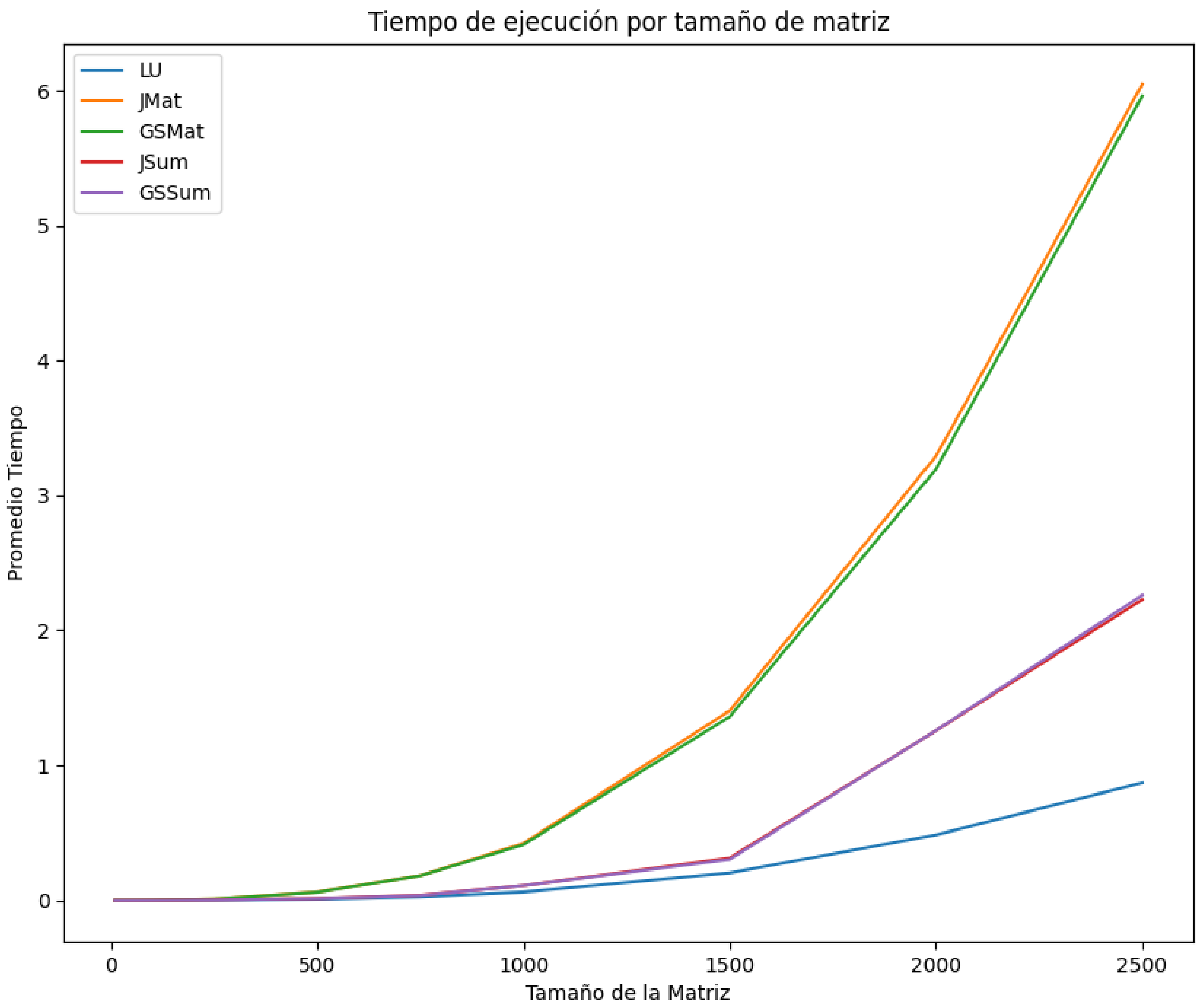

Figura 1: Tiempo de Ejecución Promedio de los Métodos en segundos.

Se puede ver una clara mejora de los métodos basados en sumatoria en comparación a los matriciales. La forma matricial implica el cálculo de operaciones matriciales, como multiplicaciones de matrices y vectores, y la inversa de matrices. Estas operaciones son computacionalmente costosas, especialmente para matrices grandes. En el caso de los métodos implementados en forma de sumatoria, el número de operaciones realizadas está relacionado con el número de elementos no nulos en la matriz.

Estadística	**LU**	**JMat**	**GSMat**	**JSum**	**GSSum**
std	0.26838	1.85419	1.81780	0.69868	0.71122
min	1e-06	3e-06	3e-06	0.000000	4e-06
max	0.97099	6.58216	6.8826	2.50674	2.4415

Cuadro 1: Estadísticas de Tiempos en segundos.

4.2. Error

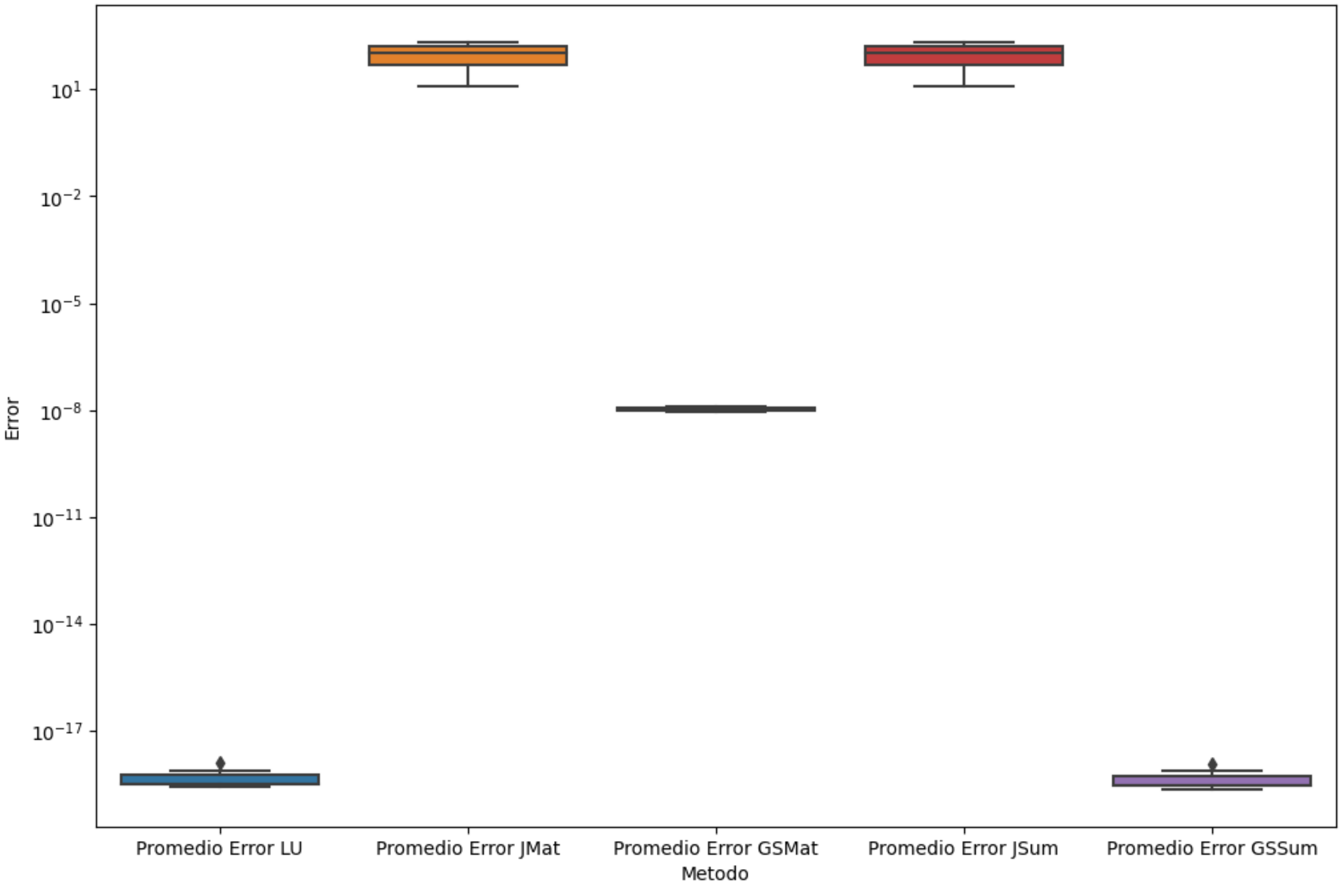

Figura 2: Error promedio de los métodos representado como boxplot.

Se puede ver como el promedio de error entre el valor esperado y el calculado es órdenes de magnitud menor en el caso de Gauss-Seidel contra Jacobi, tanto para matricial como para sumatoria (aunque este último es aún más pequeño). Gauss-Seidel en forma de sumatoria es el que más se le acerca a LU.

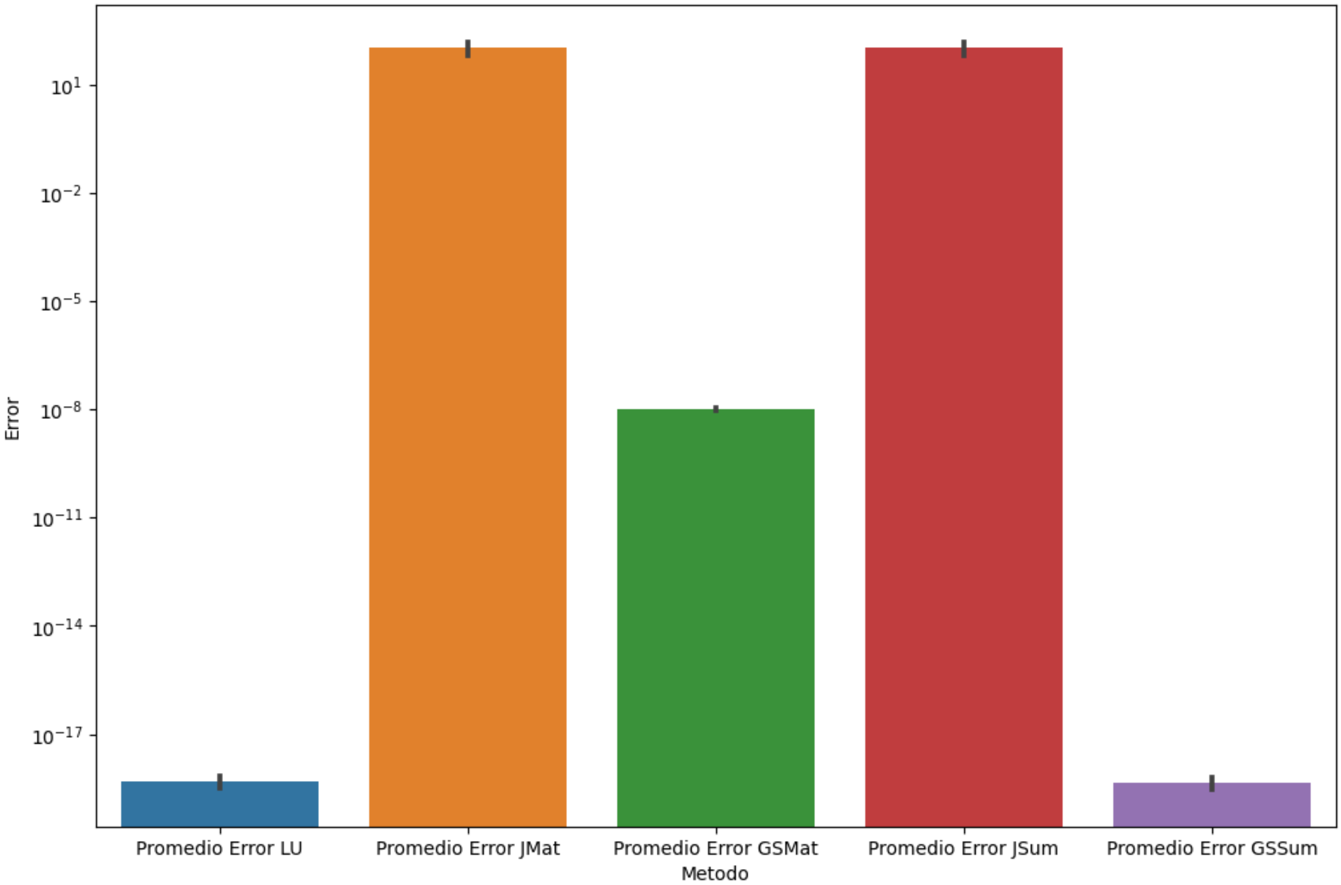

Figura 3: Error promedio de los métodos representado como barplot

Decidimos también representar el error usando un gráfico de barras, ya que nos pareció que este tipo de plot es el más claro para visualizar las diferencias entre el error promedio de los métodos.

4.2.1. Análisis del error en función de la cantidad de iteraciones.

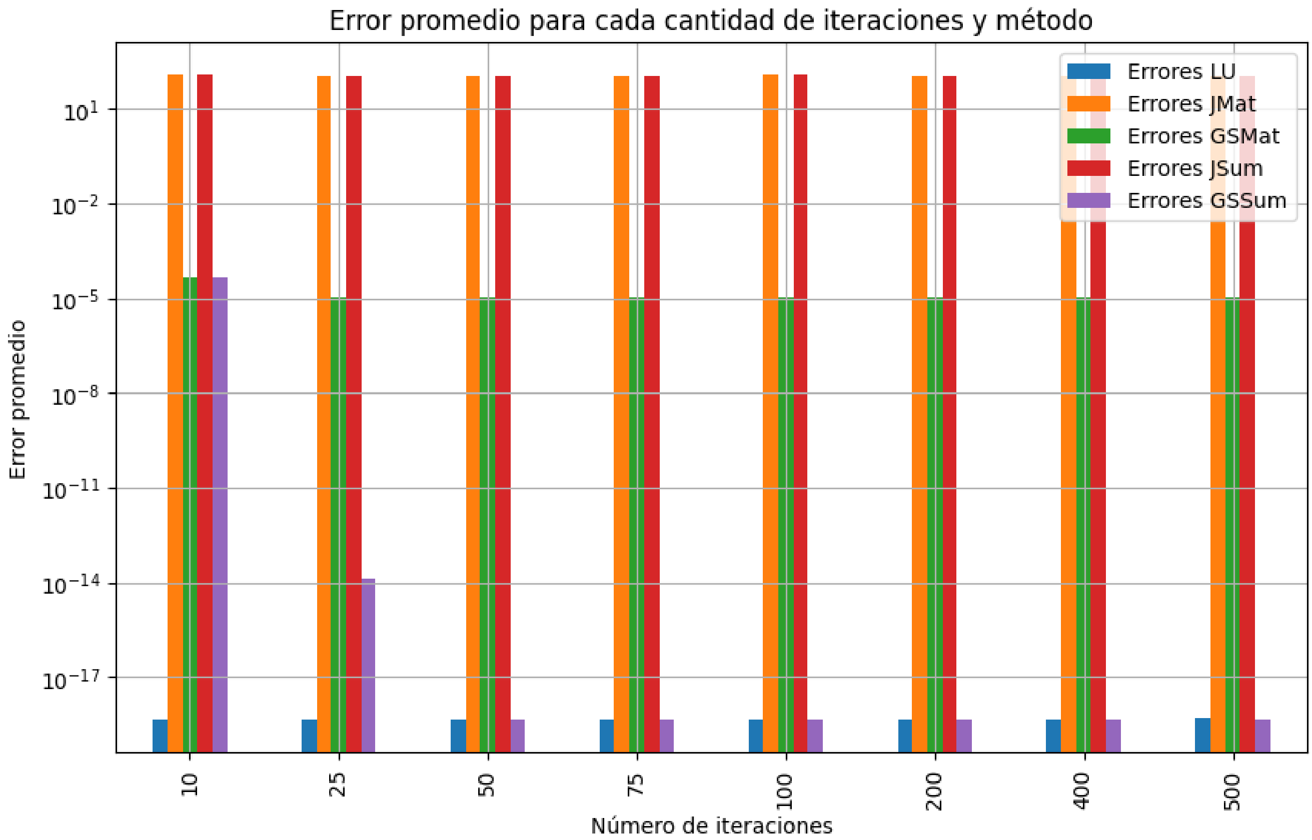

Figura 4: Error promedio para cada cantidad de iteraciones y método

El siguiente gráfico muestra como el error promedio varía a medida que se aumenta la cantidad de iteraciones que permitimos para cada ejecución de los métodos. Se puede ver que el único método beneficiado por una mayor cantidad de iteraciones es el de Gauss-Seidel en forma de sumatoria, ya que este es el único que muestra mejoras de varios órdenes de magnitud con aumentando de forma lineal la cantidad de iteraciones. Los demás tienen mejoras negligentes que no justifican el aumento de la cantidad de iteraciones para mejorar el error.

4.2.2. Análisis del error en función del número de condicionamiento.

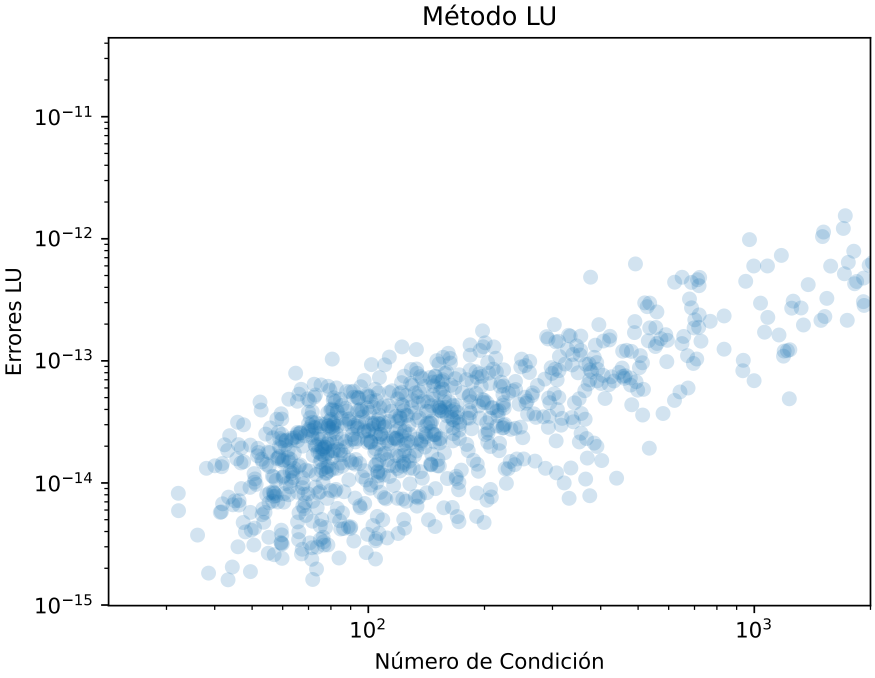

Figura 5: Error y Número de Condición LU

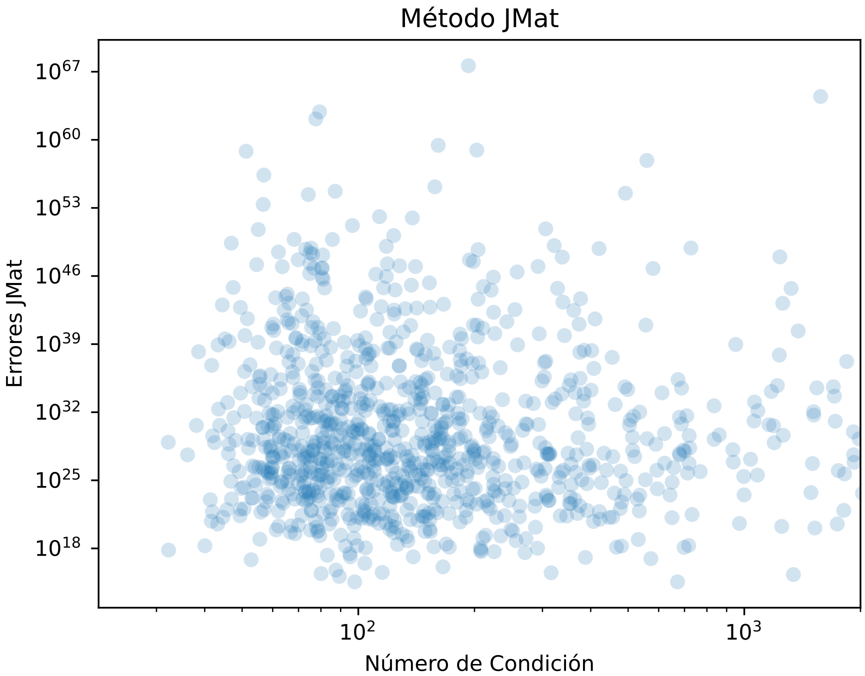

Figura 6: Error y Número de Condición JMat

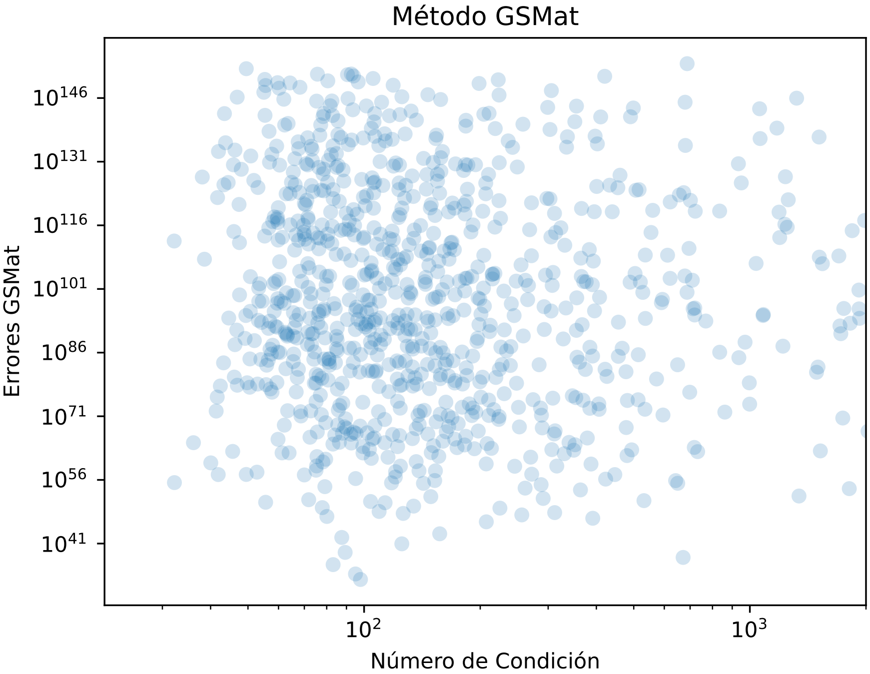

Figura 7: Error y Número de Condición GSMat

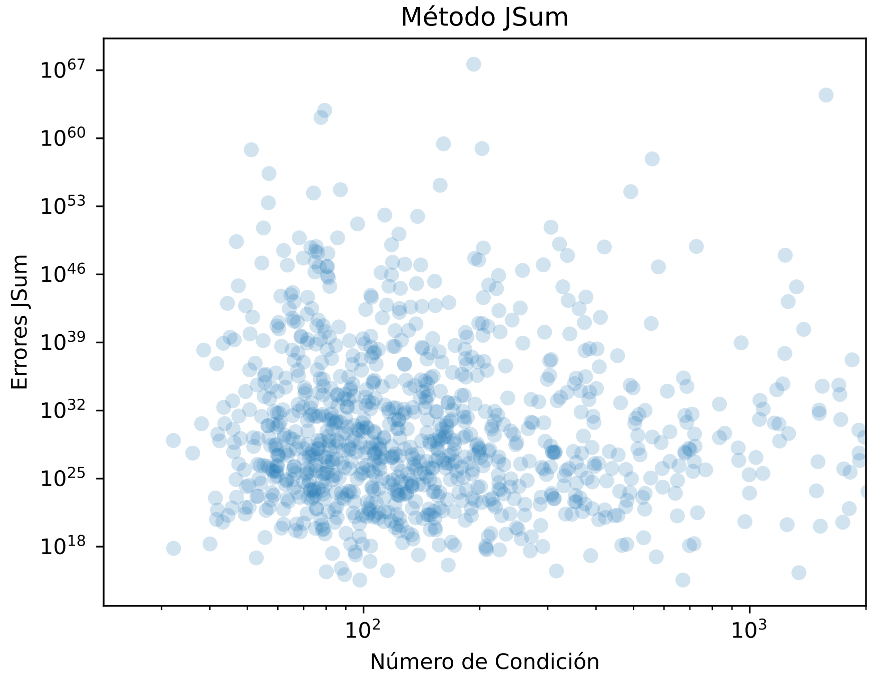

Figura 8: Error y Número de Condición JSum

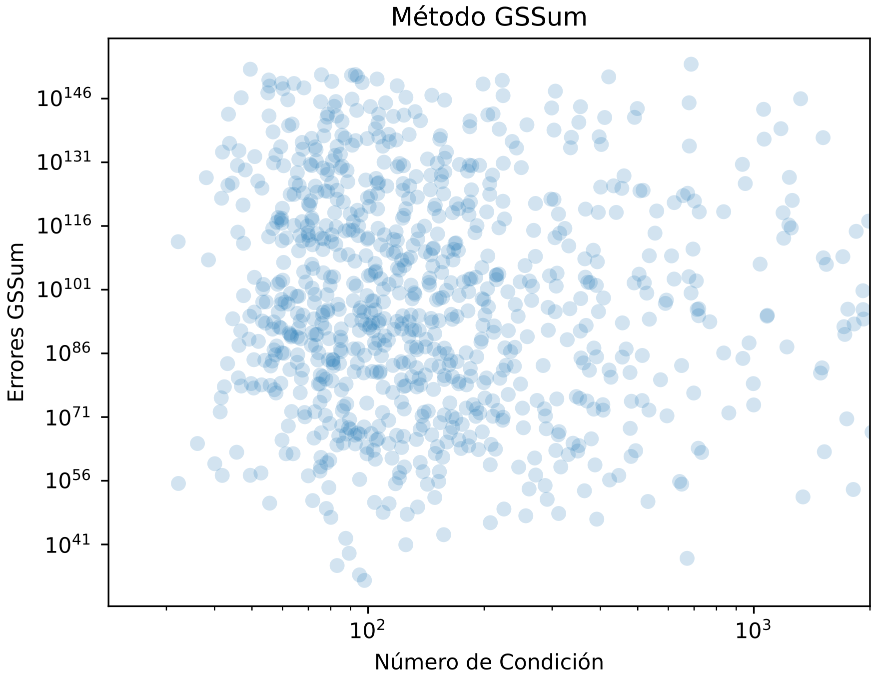

Figura 9: Error y Número de Condición GSSum

Los scatter plots muestran la relación entre el error del método y el número de condición de la matriz. Para generar las muestras fijamos un tamaño de matriz, 16 x 16, y generamos 10000 matrices de manera aleatoria. A partir de eso calculamos el número de condición a cada una de estas matrices y le aplicamos los 5 métodos. Nuestra hipótesis se basaba en que los métodos iban a seguir una tendencia similar a LU, pero es claro que en los métodos iterativos no hay una relación predecible. En cambio, se puede decir que no hay correlación alguna. Creemos que esto se puede deber a que al ser LU un método exacto va a tener una tendencia más predecible que Jacobi o Gauss-Seidel que se rigen por procesos iterativos. Es interesante notar también como el error se ve menos afectado por el número de condición para ambos casos de Jacobi en comparación con Gauss-Seidel.

5. Conclusiones

A partir de la experimentación, notamos que los métodos iterativos son en general inferiores a la hora de resolver sistemas de ecuaciones $Ax = b$, comparados con la factorización LU, ya que proveen estimaciones con un error más alto en comparación con los resultados provistos por el algoritmo de eliminación gaussiana, y además tienen una complejidad temporal más alta que hace que en promedio sean más lentos. A su vez, es interesante notar que temporalmente los métodos matriciales tienden a tener una complejidad temporal más elevada que su contraparte sumatoria. Creemos que se debe a la complejidad superior de los cálculos matriciales en comparación con las sumas directas de los otros. Por otro lado, es importante saber que el aumento en la cantidad de iteraciones no genera un menor error en promedio. Al único método que parece afectarle esa variación es a Gauss-Seidel en formato de sumatoria.

Dicho todo esto, creemos que el método iterativo que más se asemeja a LU tanto en términos de performance como de cálculo de error es el de Gauss-Seidel, en particular en forma de sumatoria. Por lo que creemos que es el más útil a la hora de elegir alguno de los metodos iterativos.

Referencias

[1] Análisis Estadístico de los Métodos. https://colab.research.google.com/drive/13QqRZ46PrgtYBWxAVsYsPdRn1eBRwJVS#scrollTo=IqZDHwsgeWBP