DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Métodos iterativos

Métodos Numéricos

Autores

Integrante	LU	Correo electrónico
Battolla, Gianfranco	17/20	gianfrancobtl@gmail.com
Orfanos, Santiago	51/21	s.orfanos@hotmail.com
Vidal, Julian	681/21	juli.vidal14@gmail.com

Resumen

En este informe se hace una comparación experimental entre métodos exactos e iterativos. Reutilizamos Eliminación Gaussiana e implementamos los métodos de Jacobi y Gauss Seidel. Analizamos su convergencia y su error. Luego, comparamos los tiempos de corrida de todas las implementaciones. Finalmente, se observa el impacto en los tiempos al aumentar las densidades del problema, especialmente con redes sumidero de n nodos.

Palabras clave

Iterative Methods, Gauss Seidel, Jacobi, Gaussian Elimination

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

Durante este trabajo, haremos experimentación en lo que refiere a resolver sistemas de ecuaciones lineales para comparar el uso de métodos exactos con métodos iterativos. Esto resulta interesante si se desea estudiar la eficacia de unos sobre los otros en cuanto al tiempo.

Los métodos exactos o directos son aquellos en los que se llega a una solución exacta después de una cantidad determinada de pasos u operaciones.

Por otro lado, los iterativos justamente realizan iteraciones, cuya cantidad es indeterminada, para llegar a una solución. En otras palabras, generan una sucesión que converge a la solución del sistema a resolver (bajo ciertas hipótesis) en una cantidad finita de pasos. Si bien a simple vista, uno podría pensar que usar un método exacto es mejor siempre, para matrices muy grandes por ejemplo, podríamos lograr una menor complejidad y (si conocemos ciertas propiedades de esas matrices) asegurar su convergencia a la solución.

Estaremos trabajando nuevamente con PageRank por lo que reutilizaremos una implementación de Eliminación Gaussiana para el método exacto. Sin embargo, como esta implementación tiene tiempos de cómputo altos, también compararemos con una implementación provista por la librería Eigen

En cuanto a los métodos iterativos, haremos una implementación del método de Jacobi y del método de Gauss-Seidel vistos en la teórica. Explicaremos más adelante cómo funcionan. Como los métodos iterativos llegan a la solución después de una cierta cantidad de iteraciones que desconocemos, graficaremos la convergencia para los casos de la cátedra y analizaremos el error respecto de las soluciones provistas.

Finalmente, analizaremos la diferencia entre las implementaciones. Esto lo haremos comparando los tiempos de ejecución y observando como estos responden al cambio de densidad de la matriz. Para esto implementamos también una manera de sumarle densidad a los grafos de una familia de matrices. Esta familia es la red sumidero, donde N nodos apuntan todos a uno central.

2. Desarrollo

Procedemos a desarrollar los algoritmos referidos a los métodos iterativos de Jacobi y Gauss-Seidel.

2.1. Los algoritmos

A continuación, expresamos ambos con un pseudocódigo que permita una primera aproximación a los mismos

Algorithm 1 Jacobi(A, b, reps, x_ini, x_direct, eps)

$D \leftarrow diagonal(A)$
$L \leftarrow D - TI(A)$
$U \leftarrow D - TS(A)$

$T \leftarrow D^{-1} * (L + U)$
$c \leftarrow D^{-1} * b$

$xi \leftarrow x\_ini$
$error \leftarrow []$
$err \leftarrow 0$

$i \leftarrow 0$
while $i \leqslant repsorerr \leqslant eps$ **do**
    $xi \leftarrow T * xi + c$
    $error.push\_back(norma\_2(x\_i, x\_direct))$
    $i \leftarrow i + 1$
end while

return x_i, error

Algorithm 2 Gauss Seidel(A, b, reps, x_ini, x_direct, eps)

$D \leftarrow diagonal(A)$
$L \leftarrow D - TI(A)$
$U \leftarrow D - TS(A)$

$T \leftarrow (D - L)^{-1} * U$
$c \leftarrow (D - L)^{-1} * b$

$xi \leftarrow x\_ini$
$error \leftarrow []$
$err \leftarrow 0$

$i \leftarrow 0$
while $i \leqslant repsorerr \leqslant eps$ **do**
    $xi \leftarrow T * xi + c$
    $err \leftarrow norma\_2(x\_i, x\_direct)$
    $error.push\_back(err)$
    $i \leftarrow i + 1$
end while

return x_i, error

En los algoritmos expuestos TI(A) hace referencia a la matriz que tiene solo los elementos con $i \leqslant j$ de A. Por otro lado, TS(A) se corresponde con la matriz que tiene solo los elementos con $i \geq j$ de A.

2.2. Los parámetros involucrados

Analizaremos cada uno de los parámetros de las funciones para el problema de PageRank

En primer lugar, $A = I - pWD$, donde I es la matriz identidad, p es un valor entre 0 y 1 (no inclusive), W es una matriz de conectividad de N x N (siendo N la cantidad de nodos en el caso elegido) y D una matriz de puntajes en función de cuántas conexiones hay en cada columna de W.

Para facilitar los cálculos y no tener necesidad de pasar un nuevo parámetro, seteamos un p fijo de 0.75 para todos los casos puesto que consideramos que es justo en función a lo analizado en el informe donde analizamos el problema del Ranking de Page con mayor profundidad.

Por otro lado, b es un vector de unos; reps es la cantidad de repeticiones que se desea que se ejecuten en los métodos iterativos; x_ini es un vector aleatorio generado, en nuestro algoritmo, con la función $fillRandomVector$; x_direct es la solución directa al problema, calculada con el método de Eliminación Gaussiana.

2.3. La implementación

2.3.1. Ejecutable

Implementamos un programa que recibe dos parámetros:
En primer lugar, un archivo txt que contenía: en la primera linea la cantidad de nodos de la matriz, en la segunda linea la cantidad de conexiones y en las siguientes lineas dos nodos refiriéndose a la conexión desde el primero hasta el segundo.
Y como segundo parámetro la cantidad de repeticiones que se quiere ejecutar los métodos iterativos.

2.3.2. Creación de Matriz de Cálculo

Para la creación de la matriz A = I - p*W*D, necesaria para realizar la Eliminación Gaussiana, hicimos uso de la librería Eigen, que provee funciones para la creación de matrices ralas (mayormente conformada por elementos nulos) y de vectores.

2.3.3. Eliminación Gaussiana - Eigen

Haciendo uso de la función "SparseLU"de la librería Eigen, resolvimos el sistema lineal Ax=b, para luego utilizar el x resultante para la comparación de los próximos dos métodos iterativos. Nos referiremos a este como "x"próximamente.

2.3.4. Eliminación Gaussiana - Nuestro método

Para implementar la Eliminación Gaussiana triangulamos la matriz A y luego resolvimos el sistema Ax=b con A ya triangulada. Como se verá más adelante, es notable que esta implementación demora mucho más de lo esperado en casos más densos (con más conexiones), y creemos que esto se puede mejorar con el uso del iterador que provee la librería Eigen.

Particularmente, detectamos que la mayor parte del tiempo se pierde al tener que triangular las filas debajo de la fila pivot, ya que a medida que la matriz se va triangulando, el tiempo requerido para triangular debajo decrece. Esto se verá con mayor profundidad al analizar los tiempos de corrida de los tests de sumidero.

2.3.5. Gauss Seidel

Para el método de Gauss Seidel implementamos una función que devuelva una tupla de vectores de la librería Eigen, el cual contenía el valor x_i que iba convergiendo en el primer elemento, y el error (x_i - x) en el segundo elemento.

Además, pasamos por parámetro la matriz A, el vector b (lleno de unos) necesario para el calculo de el Ranking de Page, la cantidad de repeticiones deseadas para los métodos iterativos, un vector inicial calculado aleatoriamente que luego convergería al ultimo parámetro pasado, el vector x resultante de la Eliminación Gaussiana. Todos estos previamente calculados.

Creamos dos vectores del tamaño de las repeticiones pasadas por parámetro, para luego ir calculando el x_i con la fórmula a continuación.

Sea $x^k \in \mathbb{R}^n$. Definimos un próximo punto x^{k+1} de la siguiente manera:

$$x_1^{k+1} = (b_1 - \sum_{j=1,j\neq 1}^{n} a_{1j}x_j^k)/a_{11}$$

$$x_2^{k+1} = (b_2 - a_{21}x_1^{k+1} - \sum_{j=3}^{n} a_{2j}x_j^k)/a_{22}$$

$$\vdots$$

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

$$\vdots$$

$$x_n^{k+1} = (b_n - \sum_{j=1}^{n-1} a_{nj}x_j^{k+1})/a_{nn}$$

y calculamos el error mediante la función *norm*() incluida en la librería Eigen.

Para finalizar, normalizamos los valores de los xi resultantes utilizando una implementación de la norma 2 hecha por nosotros.

2.3.6. Jacobi

Para este método implementamos una función parecida a la del método de Gauss Seidel, utilizando mismos parámetros de entrada y de salida, pero utilizando otra manera para el cálculo de los x_i:

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

Para esto usamos nuevamente la librería Eigen:

- Para el calculo de L y U, utilizamos la función "TriangularView", para obtener la matriz triangular superior o inferior de A.
- Para el calculo de la inversa de D resolvimos el sistema Dx = I con la función "solve". Una manera más directa de hacerlo, puesto que se trata de una matriz diagonal, hubiera sido hacer $d_{ii} = 1/d_{ii}$
- Calculamos T = D^{-1}(L + U) y c = D^{-1}b
- Por último, iteramos las repeticiones pasadas por parámetro siguiendo la formula de la figura **??**

2.3.7. Salida

El algoritmo realizado tiene seis archivos de texto de salida, los cuales son: 1) la solución de la implementación de la Eliminación Gaussiana de la librería Eigen junto con los tiempos de corrida; 2) la solución de nuestra implementación de la Eliminación Gaussiana junto con los tiempos de corrida; 3) la solución de la implementación de Jacobi junto con los tiempos de corrida; 4) la solución de la implementación de Gauss Seidel junto con los tiempos de corrida; 5) los errores de Jacobi por cada iteración; 6) los errores de Gauss Seidel por cada iteración.

3. Experimentación y resultados

3.1. Análisis de convergencia

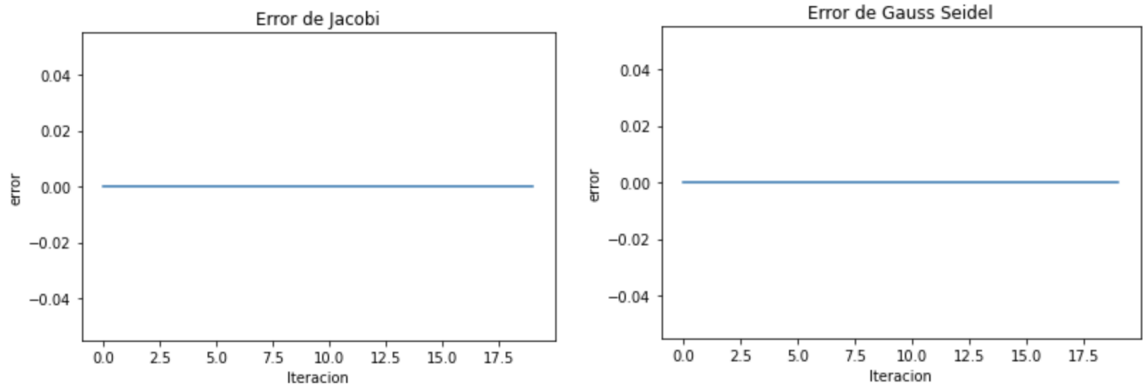
Como mencionamos antes, ambos algoritmos devuelven tanto la solución x a la que arribaron tras iterar n veces como un vector que guarda el error incurrido en cada una de las iteraciones hasta llegar al resultado. En esta sección analizaremos este último y veremos cómo se comporta para los métodos de Gauss Seidel y Jacobi para los cinco tests provistos por la cátedra.

Todos los tests fueron evaluados para 20 repeticiones, a excepción del caso de 30 segundos, el cual fue corrido con un número de repeticiones igual a 30.

Cada uno de los errores será expresado como la norma 2 del vector resultante de restar el x directo (obtenido mediante Eliminación Gaussiana) y el xi de cada iteración. Elegimos tomar la este valor para obtener resultados pequeños y más manipulables. Cuando expresemos que el error "llega a 0"haremos referencia a que llega a un valor menor que 1e-6.

3.1.1. Tests de la cátedra

En primer lugar, analizamos el **test_sin_links**. Como puede verse en la Figura 1, ambos métodos llegan a la solución en la iteración número 1. Esto se debe a que, al no haber links, luego todas las páginas se reparten el puntaje en partes iguales; no existen datos que den lugar a la existencia de error alguno.



(a) Jacobi **(b)** Gauss Seidel

Figura 1: Convergencia de Jacobi y Gauss Seidel luego de 20 iteraciones para el test sin links.

En segundo lugar, vimos qué ocurría con el **test aleatorio (ordenado)**. En este caso, vemos que el método de Gauss Seidel arriba a un error de 0 en la onceava iteración. Por otro lado, el de Jacobi llega al mismo error tras 20 iteraciones. Es decir, vemos que el de Gauss Seidel converge más rápido. Al completar las 20 iteraciones que le impusimos a los métodos iterativos, ambos arriban a un error de 0.

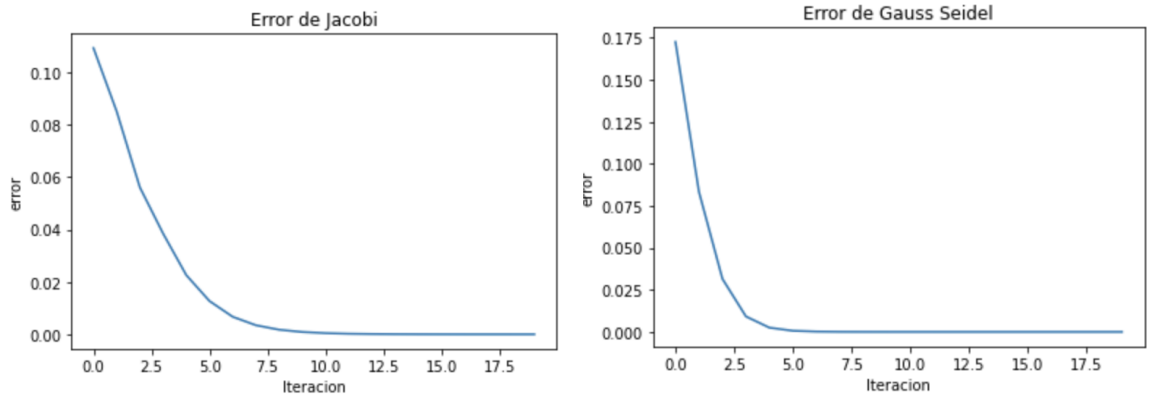


Figura 2: Convergencia de Jacobi y Gauss Seidel luego de 20 iteraciones para el test aleatorio.

En tercer lugar, analizamos el **test aleatorio (desordenado)**. En este test, ambos métodos se comportan de manera similar al anterior, arribando a errores nulos en la onceava y decimonovena iteración respectivamente.

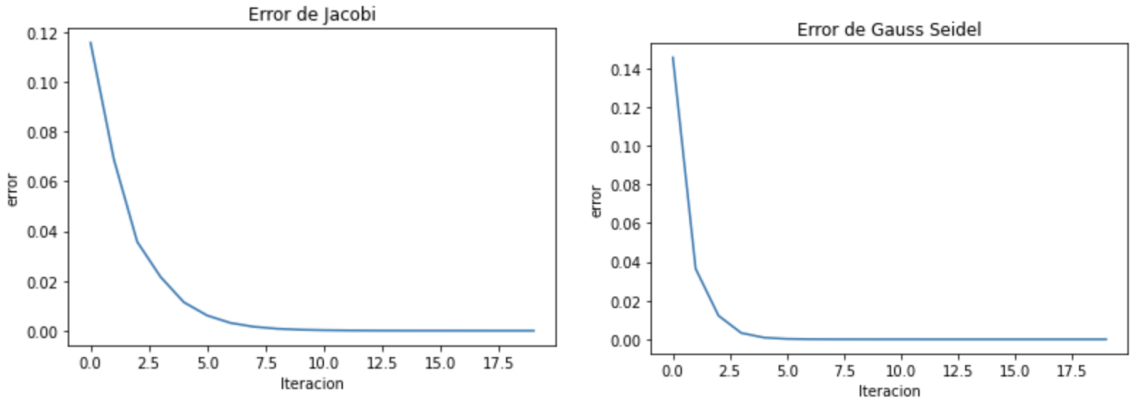


Figura 3: Convergencia de Jacobi y Gauss Seidel luego de 20 iteraciones para el test aleatorio desordenado.

Esto nos permite afirmar que el orden en que se presentan las conexiones entre los nodos no tiene influencia en la convergencia de errores de los métodos. Esta afirmación tiene sentido porque la matriz T y el vector c que conforman ambos métodos iterativos serán iguales al test ordenado tanto para Gauss Seidel como para Jacobi.

Si vemos las soluciones tanto para el ordenado como el desordenado, el método de Gauss Seidel confluye perfectamente a la solución que se desprende directamente de la Eliminación Gaussiana si contabilizamos 10 decimales. Por su parte, la solución de Jacobi es idéntica hasta el sexto decimal. Asimismo, podemos decir que es una aproximación más que aceptable.

Creemos que es probable que con algunas iteraciones más, Jacobi llegue a una solución igual a la de Gauss Seidel. Corrimos el algoritmo con 50 iteraciones y así fue: los 10 decimales resultaron ser iguales.

En cuarto lugar, vimos qué ocurría con el **test completo**. Viendo el gráfico, puede apreciarse que el método de Jacobi converge, en este caso, con mayor velocidad. Y esto es así: tras 8 iteraciones, Jacobi arriba a un error nulo; por su parte, a Gauss Seidel le demanda el total de las 20 iteraciones llegar a un error semejante.

Si vemos las soluciones, ambos métodos confluyen perfectamente a la solución que se desprende directamente de la Eliminación Gaussiana si contabilizamos 10 decimales.

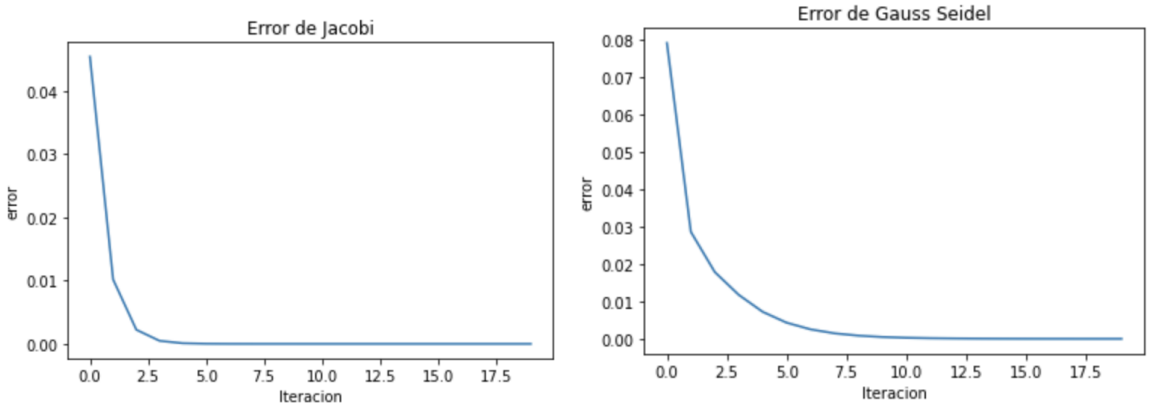


Figura 4: Convergencia de Jacobi y Gauss Seidel luego de 20 iteraciones para el test completo.

En quinto lugar, vimos qué ocurría con el **test de 15 segundos**, el cual tiene 2000 nodos y 12000 conexiones. Siguiendo con la tendencia, el método de Gauss Seidel confluyó con mayor velocidad: llegó a un error del orden de 0 en 16 iteraciones mientras que Jacobi precisó las 20 (y aún así no llegó a un error de 1e-6 esperado sino que alcanzó 1e-4).

Podemos ver que el error es bastante más elevado si lo comparamos con tests anteriores. Esto, creemos, se debe a la mayor cantidad de nodos y conexiones.

Finalmente, analizamos el **test de 30 segundos**, el cual tiene 3000 nodos y 18000 conexiones. Debido a ser un problema con muchas más conexiones, corrimos el algoritmo con treinta en lugar de veinte iteraciones.

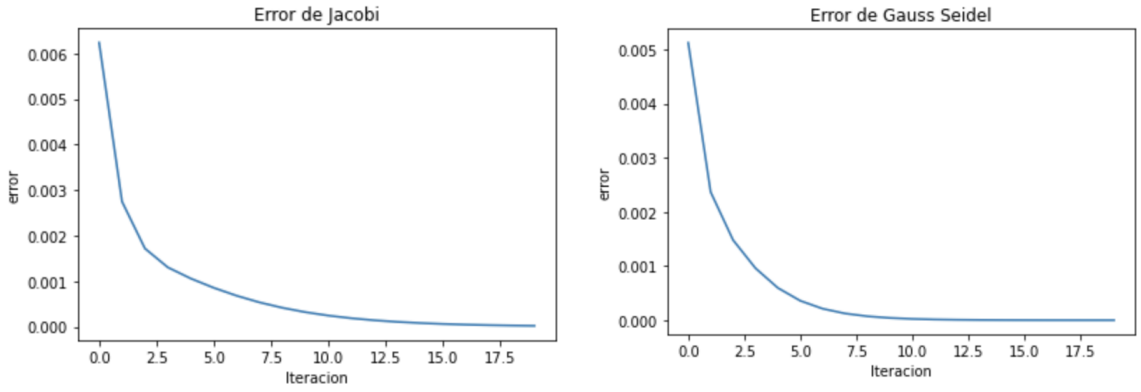


Figura 5: Convergencia de Jacobi y Gauss Seidel luego de 20 iteraciones para el test de 15 segundos.

Con este test, ocurrió lo esperado en función a lo que veníamos analizando. Por un lado, el método de Gauss Seidel confluyó más rápidamente: llegó a un error nulo tras 20 iteraciones mientras Jacobi requirió las 30. Ambos métodos convergieron satisfactoriamente.

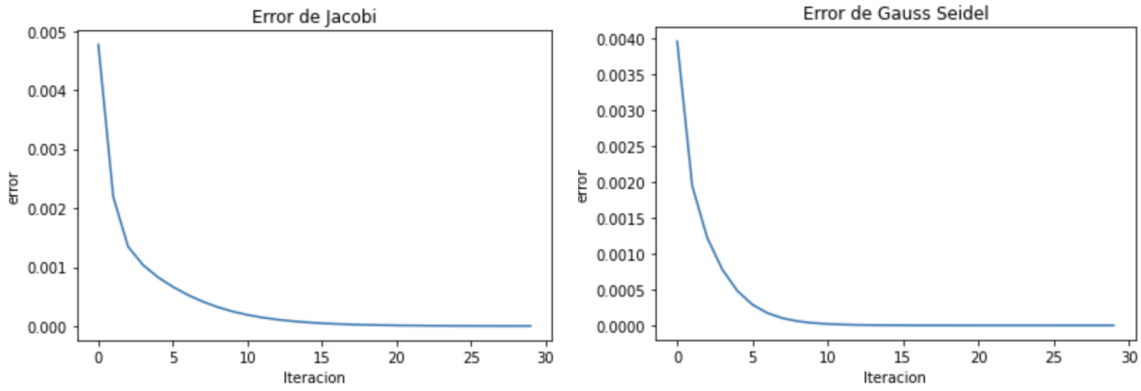


Figura 6: Convergencia de Jacobi y Gauss Seidel luego de 30 iteraciones para el test de 30 segundos.

3.2. Análisis de tiempos

Para analizar cuánto tarda cada método iterativo y compararlos, se programó una función que corriera cinco veces cada uno de ellos para así poder obtener un promedio y el desvío estándar correspondiente. A continuación se muestran cuáles han sido los resultados.

Cada uno de los métodos iterativos fue evaluado para 20 iteraciones para los tests de la cátedra y solo 2 iteraciones para los tests creados para las redes sumidero. ¿Por qué tan poco? Porque al correrlos con más iteraciones, evidenciamos que la solución no se alteraba, luego sería lo mismo probarlos con 2, 5, 100 o 10000 iteraciones. De esta manera, sería además más fácil probar estos métodos en redes con, por ejemplo, 2500 nodos, pues tardaría menos tiempo que si hubiésemos puesto más iteraciones.

3.2.1. Tests de la cátedra

En primer lugar, realizamos la medición de los tiempos para cada uno de los tests provistos por la cátedra. Para hacerlo, corrimos cada método cinco veces, registrando cada uno de los tiempos y calculando su promedio y su desvío estándar.

Como puede verse en la figura 7, el promedio de tiempos para los tests aleatorio (ordenado y desordenado), completo y sin links son muy parecidos, ya que ningún caso llega a 1ms, por ende todos los tiempos son muy cercanos a 0.

Como puede verse, las diferencias en los tiempos de ejecución son mínimas y despreciables.

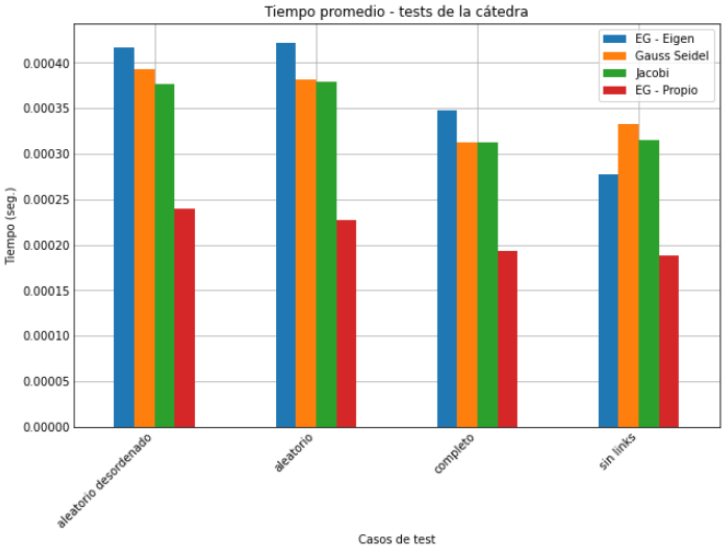


Figura 7: Tiempo promedio - tests de la cátedra (20 iteraciones).

Es apreciable que en estos tests (figura 8), donde hay una cantidad bastante mayor de nodos (nos encontramos en el orden de los miles) el tiempo promedio de la Eliminación Gaussiana (Eigen) es ampliamente superior al de los métodos iterativos, anotando 19 segundos promedio para el test de 15 y 66 segundos promedio para el de 30.

Por su parte, también podemos comparar los tiempos de los métodos iterativos. En este sentido, vemos que el de Jacobi tiene un rendimiento bastante superior. En el de 15 segundos, Jacobi realiza su trabajo en 1.18 segundos mientras que Gauss Seidel lo hace en 5.18s.

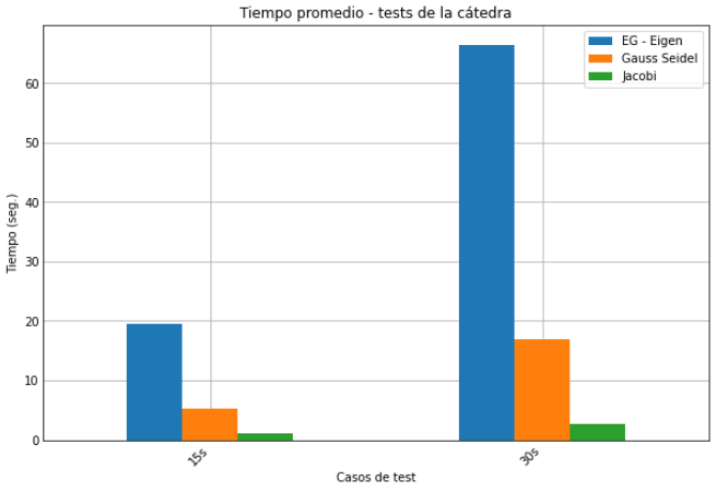


Figura 8: Tiempo promedio - tests de 15 y 30 segundos (20 iteraciones).

Lo mismo sucede en el de 30 segundos: Jacobi corre en 2.6s segundos mientras que Gauss Seidel lo hace en 16.83s.

Además, también evaluamos nuestro método de Eliminación Gaussiana. El mismo nos dio tiempos demasiado elevados, que rondaron los 100 segundos en promedio para el test de 15 segundos y los 400 segundos para el de 30.

Ese resultado se debe, principalmente, a no haber usado iteradores en la implementación del método. De haberlos usado, el mismo solo habría iterado sobre los elementos no nulos de la matriz. Sin embargo, no hemos podido descifrar cómo iterar primero por toda una fila (con sus respectivas columnas) y luego el resto, lo cual nos habría permitido realizarlo de esta manera. Nuestra hipótesis es que, igualmente, un método de E.G. implementado con iteradores habría demorado, de todas maneras, más que ambos métodos iterativos.

Por otro lado, y siguiendo con el análisis, dedicamos un espacio a ver qué sucedía con el desvío estándar

de los tiempos de ejecución. En este sentido, todos los desvíos estándar nos han dado un resultado menor a 1 para todos los métodos y para cada uno de los tests.

Sin embargo, podemos evidenciar que el mismo se agranda cuando trabajamos con casos en los que participan muchos nodos y hay más conexiones. Esto puede verse con mayor precisión en las redes de sumidero que analizaremos a continuación.

3.2.2. Red sumidero

Hemos creado una porción de código que se encarga de generar una red sumidero en función de los n nodos y la cantidad de conexiones que se le proporcione como parámetro. Corrimos el mismo para 2500 nodos. Y además, corrimos cada uno con 3 cantidades de conexiones diferentes: 250, 1250, y 2499 conexiones.

Luego, hicimos gráficos con los 3 casos de test y analizamos como variaban los tiempos de los diferentes métodos según se agregaba densidad a los gráficos. Los resultados se muestran a continuación:

Analicemos que pasa a medida que aumenta su densidad:

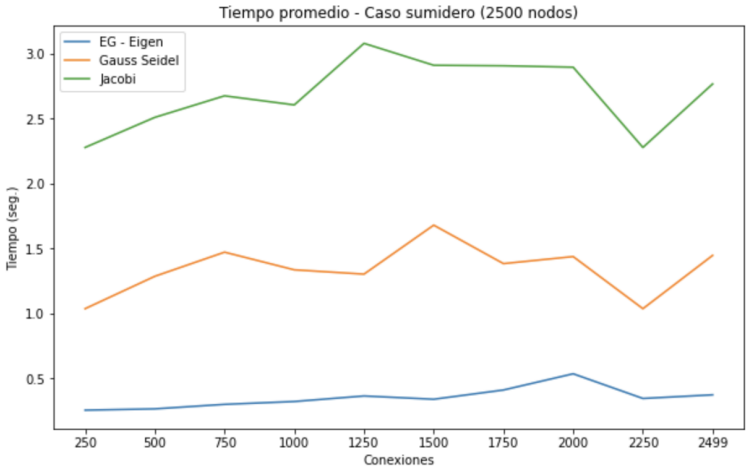


Figura 9: Tiempo promedio para la red sumidero de 2500 nodos.

Analicemos el tiempo de ejecución. En primer lugar, vemos que Gauss Seidel tiene tiempos menores a los presentados por Jacobi, lo cual se condice con gran parte de los tests de la cátedra realizados.

En segundo lugar, vemos que los tiempos no varían demasiado en función de la densidad. El único método que parece demorar un poco más a medida que aumenta la cantidad de conexiones es el de la E.G. Sin embargo, vemos también que es el método que menores tiempos presenta. Esto tiene mucho sentido y se debe a que la matriz sumidero es una matriz que ya se encuentra diagonalizada y lo único que le demanda un tiempo considerable es la búsqueda de la solución x.

Por otro lado, vimos analíticamente que tanto para Jacobi como para Gauss Seidel el desvío estándar de los tiempos es pequeño para todas las densidades. ¿Qué significan estos resultados de desvío estándar tanto para la red sumidero como los obtenidos para los tests de la cátedra? Que la diferencia entre los tiempos de ejecución de cada método es muy poca, por lo tanto se espera que, si queremos repetir alguna corrida del algoritmo para una determinada cantidad de nodos, el mismo tardará un tiempo muy parecido al que tardó cuando lo hicimos con anterioridad.

3.2.3. Más análisis de densidades

Para continuar con el análisis de densidades, creamos una porción de código que crea redes de 250 nodos con 10 densidades diferentes que van desde una densidad del 10 (6000 conexiones) hasta el 90 por ciento (62250 conexiones).

En este caso modificamos uno de los errores que habíamos tenido previamente y agregamos un criterio de parada adicional: ahora no solo frenará al completar las 30 iteraciones sino que lo hará antes si el método alcanza un error menor a 0.0000001.

Veamos qué sucede con esto:

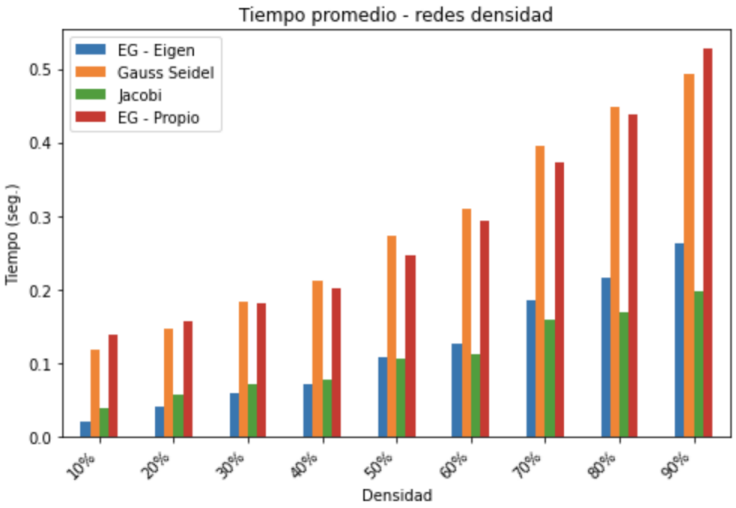


Figura 10: Tiempo para diferentes densidades.

Como puede verse el tiempo promedio para todos los tests aumenta sostenidamente a medida que la densidad de la red es mayor. Sin embargo, también puede verse cómo los métodos de Gauss Seidel y el de Eliminación Gaussiana propio (sin iteradores) aumentan de mayor manera.

Analizamos si los menores tiempos en Jacobi se debían a que alcanzaba el error esperado antes pero esto no fue así. El método realiza el total de 30 iteraciones.

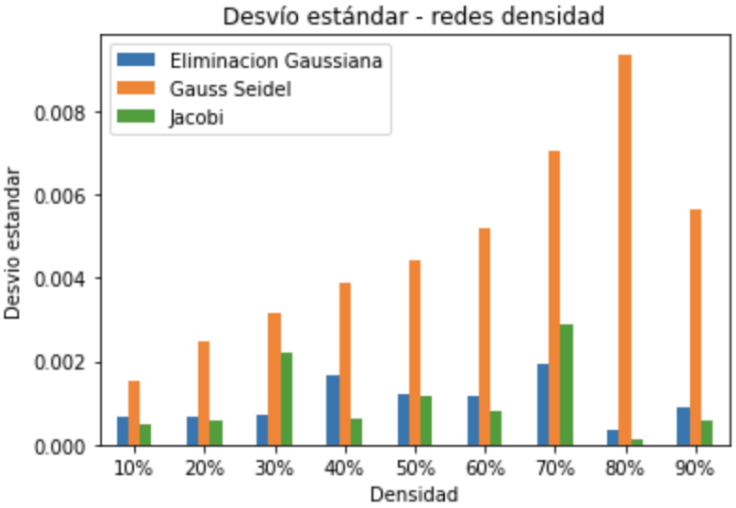


Figura 11: Desvío estándar para diferentes densidades.

¿Qué ocurre con el desvío estándar? Vemos que el método de Gauss Seidel tiene un desvío mucho mayor al de los otros métodos. Por su parte, la E.G. mantiene un error parejo con todas las densidades y no tiene una correlación con la misma. Finalmente, el método de Jacobi también se mantiene estable, por más que tiene aumentos aislados para las densidades de 30 y 70 por ciento. Sin embargo, que para el test de 90 por ciento el desvío sea tan bajo da el parámetro de que el aumento en el desvío fue, quizá debido a una mayor lentitud del procesador.

4. Conclusiones

Durante el transcurso del trabajo, nos ocupamos de analizar los diferentes métodos iterativos y analizamos su error y el tiempo que demora su ejecución.

En primer lugar, podemos concluir que tanto Jacobi como Gauss Seidel confluyen a un error que se encuentra dentro del margen esperado y que se acerca asintóticamente a 0 con repeticiones que tendiesen al infinito. Esto se debe a que el radio espectral de las matrices analizadas es, para todas, menor que uno.

En segundo lugar, podemos analizar la velocidad a la que los métodos confluyen. En este sentido, vemos que Gauss Seidel llega más rápido a la solución que Jacobi, método que precisa en promedio el doble de iteraciones para llegar a la misma. Esto nos hace pensar que, para matrices de dimensiones más grandes, será conveniente usar el método iterativo de Gauss Seidel por sobre el de Jacobi.

Por este mismo camino, comparemos los tiempos con la Eliminación Gaussiana de la librería Eigen para los tests de 15 y 30 segundos. Como se ve en la figura 10, la E.G. tiene tiempos muy superiores a ambos métodos iterativos. Esto es remarcable puesto que el método pertenece a una librería y se encuentra perfeccionado al máximo para tener un rendimiento de tiempos muy bueno.

También podemos realizar una comparación de tiempos con nuestra implementación de Eliminación Gaussiana. La misma, por las razones explicitadas en el transcurso del trabajo, no presenta la mejor performance y posee tiempos de ejecución ampliamente superiores a los métodos analizados anteriormente. Apuntamos a mejorar en un futuro esta implementación para que la inserción sea menos costosa, probablemente mediante la utilización de iteradores sobre los valores no nulos de la matriz esparsa que recorremos.

En tercer lugar, analicemos qué ocurrió con los tiempos de ejecución para la red sumidero. Al tratarse de matrices extremadamente ralas con al menos un 90 por ciento de sus posiciones nulas, la E.G. de la librería Eigen es la más veloz en comparación con los métodos iterativos. También es mucho más rápida que nuestra implementación de E.G., como cabía esperar. Esto se debe a que nuestra implementación, como está dada, itera también sobre los elementos nulos.

Finalmente, damos nuestra impresión sobre el trabajo: sentimos que nos ha servido para entender con mayor profundidad los métodos iterativos y cuándo es conveniente usarlos (en matrices de más dimensiones y no tan ralas) y cuándo no (cuando las matrices no son lo suficientemente grandes o son muy grandes con muchas posiciones en cero).