DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico N°3

Integrante	LU	Correo electrónico
Martinelli, Ezequiel	883/19	ezeaam@gmail.com
Mauro, Andrés	39/17	sebaastian_mauro@live.com
Tasat, Dylan	29/17	dylantasat11@gmail.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

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

Keywords: Metodos iterativos, Método de Jacobi, Método de Gauss-Seidel, Eliminación Gaussiana ,Page-rank

Índice

1. Introducción

En muchas situaciones se busca resolver sistemas de ecuaciones lineales, en este trabajo vamos a comparar varias formas de resolver el sistema para intentar encontrar cuál es más eficiente, en concreto vamos a comparar el método de eliminación gaussiana visto en el primer trabajo práctico de la materia con dos métodos iterativos. Para esto vamos a utilizar las instancias de un problema ya explicado en un trabajo práctico previo llamado Page Ranking, dado que estas se resuelven con un sistema de ecuaciones lineales. Entonces podemos utilizar instancias de Page Ranking a la hora de correr los distintos algoritmos para poder comparar sus tiempos.

Un método iterativo busca resolver un sistema generando una iteración $x^{(k)}$ que converge a una solución del sistema que debemos resolver. Entonces buscamos una sucesión $x^{k+1} = Tx^k + c$ que si converge lo hace a una una solución del sistema. Donde T es la matriz de iteración del método y c es el término independiente del mismo.

Existe una gran variedad de métodos iterativos que resuelven un sistema de ecuaciones lineales ($Ax = b$), pero en concreto vamos a utilizar dos, el método de Jacobi y el método de Gauss-Seidel.

Antes de explicar cada uno de los métodos iterativos analizaremos la matriz A del sistema de ecuaciones $Ax = b$. Podemos reescribir A como $A = D - U - L$, donde D es una matriz diagonal con los elementos de la diagonal de A, L es una matriz triangular inferior con los elementos de A que están bajo la diagonal cambiados de signo y U es una matriz triangular superior con los elementos de A que están por arriba de la diagonal cambiados de signo; tanto L como U tienen todos los elementos de la diagonal nulos. Entonces $Ax = b$ es un sistema equivalente a $(D - L - U)x = b$.

Nos interesa poder definir A de esta forma porque los dos métodos que vamos a estudiar utilizan esta descomposición de A para llegar a su respectivo sistema iterativo. Lo que se busca en estos métodos iterativos es partir de una instancia $Ax = b$ y llegar a una $x = Tx + c$ donde el x siendo multiplicado por la matriz del método es el de la iteración anterior a la actual.

Antes de explicarlos de forma individual cabe aclarar que para ambos métodos si la matriz A del sistema $Ax = b$ es estrictamente diagonal dominante tanto el método de Jacobi como el de Gauss-Seidel convergen a partir de cualquier x^0 inicial[1]. Como en las instancias de Page ranking la matriz $A = I - pWD$ del sistema es diagonal dominante, probado en el primer trabajo práctico, ambos métodos convergen.

1.1. Método de Jacobi

Dado $Ax = b$ descomponemos A como los vimos previamente e intentamos llegar a una expresión $x = Tx + c$.

$$Ax = b \Leftrightarrow (D - L - U)x = b \Leftrightarrow Dx - (L + U)x = b$$

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

[1] Clase teórica de métodos iterativos, diapositiva 49.

Ahora si los elementos de la diagonal de la matriz D son todos distintos de 0 es no singular. Para esto los elementos de la diagonal de A tienen que ser no nulos y esto hace que el método de Jacobi no se pueda expresar si este no es el caso.

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

Ahora definimos $D^{-1}(L + U) = T$, $D^{-1}b = c$. De tal forma que nuestra sucesión $x^{k+1} = Tx^k + c$ es:

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

1.2. Método de Gauss-Seidel

Al igual que con el método de Jacobi vamos a utilizar la misma descomposición de A y llegar a otra sucesión $x^{k+1} = Tx^k + c$ con una T y una c distintas:

$$Ax = b \Leftrightarrow (D - L - U)x = b \Leftrightarrow (D - L)x - Ux = b$$

$$(D - L)x = Ux + b$$

$(D - L)$ es una matriz triangular inferior y su determinante es la multiplicación de los elementos de la diagonal, entonces, de la misma forma que antes, si ningún elemento de la diagonal es nulo la matriz es inversible. Así tenemos:

$$(D - L)x = Uxb \Leftrightarrow x = (D - L)^{-1}Ux + (D - L)^{-1}b$$

Entonces al definir $T = (D - L)^{-1}U$, $c = (D - L)^{-1}b$ llegamos a:

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

2. Desarrollo

En esta sección vamos a implementar los distintos algoritmos que necesitamos para este trabajo, estos son:

- Generación de la matriz $I - pWD$ para PageRank.
- Algoritmo de Eliminación Gaussiana.
- Algoritmo de Jacobi.
- Algoritmo de Gauss-Saidel.

Los parámetros del proyecto van a ser los mismos que en el TP previo (la matriz de la red y el p), sumado a un parámetro que te deja elegir cual algoritmo vas a ejecutar.

Para la lectura de la matriz y escritura del resultado vamos a reutilizar la implementación que ya realizamos, sin embargo en este trabajo vamos a utilizar una librería llamada Eigen para almacenar la matriz de forma sparse.

La librería Eigen es una implementación bastante utilizada en la comunidad C++ para operaciones con matrices y cuenta con un módulo dedicado a matrices sparse, sin embargo como en el TP 1 hicimos una clase $SparseMatrix$ con operaciones generales, la migración va a ser tan simple como buscar las funciones equivalentes en Eigen y reemplazarlas, podríamos haber dejado todo el código igual y reemplazar el código interno de la clase $SparseMatrix$ para que use Eigen internamente pero decidimos no hacerlo por prolijidad y en cambio reemplazamos las matrices por el objeto $SparseMatrix < double >$ de Eigen (puede sonar confuso porque ambas tienen el mismo nombre) y los vectores por $VectorXd$ (estabamos usando la librería *vector* en el TP 1).

2.1. Eliminación Gaussiana

Lo primero que hicimos fue migrar la implementación de la Eliminación Gaussiana a la librería Eigen, luego de un ligero análisis de la documentación de la misma fue bastante simple hacer un mapeo de las operaciones básicas: inserción, modificación, suma, resta, multiplicación, etc.

Incluso encontramos una función equivalente a la función *blind_set* que habíamos definido en el TP 1 para insertar elementos con precondición que el valor previo en esa posición sea cero, esto es muy útil a nivel complejidad ya que evita realizar la búsqueda binaria para obtener el valor previo de ese elemento.

Luego de completado pudimos observar una mejora bastante significativa en los tiempos de los ejemplos de 15 y 30 segundos pero seguían dentro del mismo orden de complejidad aunque con una constante bastante inferior.

2.2. Resolución de algoritmos iterativos

Debido a que ambos algoritmos iterativos se pueden resolver de una forma genérica realizando $x^{k+1} = Tx^k + c$, vamos a implementar un método que reciba la T y c y calcule la solución.

Para esto definimos una cantidad máxima de iteraciones para la convergencia, si no se detecta convergencia en 1,000,000 de iteraciones consideraremos como que falló la convergencia, sin embargo en los ejemplos de este trabajo no debería suceder debido a la estructura de la matriz como explicamos previamente.

Hay muchas alternativas para comprobar la convergencia, decidimos utilizar el coseno al igual que en el TP 2 por simplicidad, se realiza el coseno entre el x^k y x^{k-1} con el producto escalar, luego utilizamos un delta para ver si está suficientemente cerca de 1 de la siguiente forma: $||(x^{\{k\}})^t x^{\{k-1\}}| - 1| < \delta$.

Luego la implementación del método es tan simple como defir un x_0 inicial de forma aleatoria y luego realizar un for que en cada iteración modifique x como mencionamos previamente y compruebe si la condición de convergencia fue satisfecha, en caso de ser cierto se detiene y devuelve el valor de x normalizado para que la suma de sus elementos sea 1.

2.3. Algoritmo de Jacobi

Para resolver este algoritmo debemos obtener las matrices T y c de Jacobi que se definen de la forma:

- $T = D^{-1}(L + U)$
- $c = D^{-1}b$

A través de unos métodos que provee la librería para obtener la diagonal, triangular inferior y triangular superior de una matriz, pudimos obtener facilmente la factorización $A = D - L - U$.

Utilizamos un método provisto por la librería para obtener la inversa de la matriz D (aunque es solo invertir los elementos de la diagonal), y luego utilizamos las operaciones matriciales provistas por Eigen para calcular T y c y resolver con el método genérico.

2.4. Algoritmo de Gauss-Seidel

Para resolver este algoritmo debemos obtener las matrices T y c de Gauss-Seidel que se definen de la forma:

- $T = (D - L)^{-1}U$
- $c = (D - L)^{-1}b$

Luego al igual que en Jacobi calculamos la inversa de $D - L$ con la librería y generamos la T y c para luego llamar al algoritmo genérico y obtener el resultado.

3. Experimentación

3.1. Instancias de experimentación

Dado los métodos que estamos estudiando en este trabajo práctico, propusimos las siguientes estructuras de matrices las cuales nos resulta interesante analizar para estos métodos:

- Aleatorias: estas instancias no son más que matrices cuadradas en donde variamos su dimensión y a su vez, la densidad de las mismas (cuanta celdas contienen valores distintos a 0), de esta manera generamos instancias que poseen n filas (con $n \in \{1, 25, 50, 75, 100\}$) y para cada cantidad de filas, variamos la cantidad de celdas variando entre $\{0, 0{,}25n^2, 0{,}5n^2, 0{,}75n^2, n^2\}$

- Árbol: Utilizamos matrices de adyacencia que representan un grafo con forma de árbol completo. De esta forma la matriz de adyacencia W va a tener cada columna representando un nodo con a lo sumo dos elementos distintos de 0, cada uno representando una conexión con un hijo, además tendrá muchas columnas cuyos elementos serán nulos por lo cual la matriz resultante es muy rala. Al traducirla a una matriz de page ranking, ésta conserva una cantidad de elementos no nulos similar, por lo tanto sigue siendo una matriz rala. En la figura 1 se muestra un ejemplo de la forma de la matriz resultante.

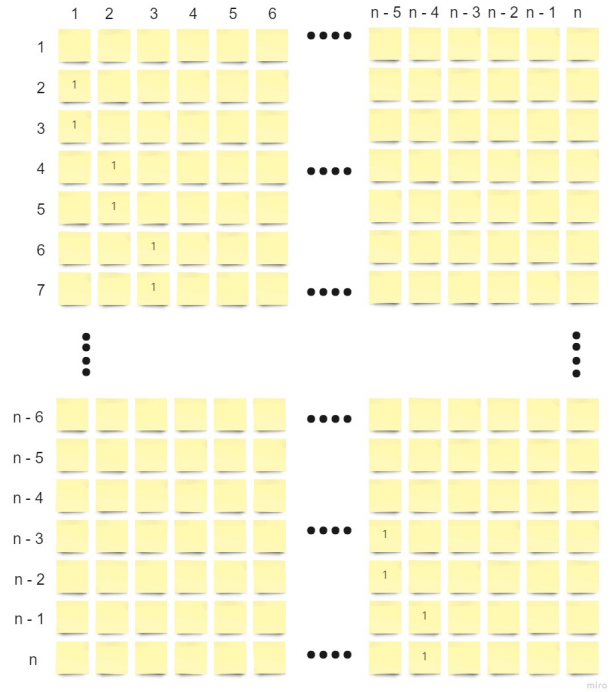

Figura 1: Gráfico de una matriz de adyacencias de un árbol binario

- 3 Principales: las matrices de este dataset fueron generadas en el trabajo práctico 1 para representar un grafo en la cual hay 3 nodos principales. En la construcción de estas instancias se vario determinó la cantidad de filas y columnas de la matriz entre 100 y 1000 (con intervalos de 100) y fijamos la cantidad de elementos no nulos que poseen las columnas 1, 2 y 3 en función a la cantidad total de elementos no nulos en la matriz.

a estas columnas se les asignaron: el 50 %para la columna 1, el 25 % para la columna 2 y el 10 % para la columna 3. El 15 % restante fue asignado con una distribución uniforme al resto de columnas en la matriz.

- 3 Principales y un Anillo: En este dataset se modelaba el caso en el que la red contiene 3 nodos principales, pero también contiene un sub conjunto de nodos que forman otra componente conexa. Esta otra componente forma anillo de nodos que se enlazan entre si (cada uno con su vecino) y finalmente, estos apuntan a un nodo que es el central del anillo.
  En la construcción de estas instancias, se dejo fija la cantidad de páginas que conforman el anillo (el 50 % de la cantidad total de columnas) y se determinó que sean las últimas de la matriz y en particular, la página central es la última de todas. Se vario la cantidad de filas (y columnas) la matriz con valores entre 100 y 1000 como las instancias anteriormente mencionadas.

  Esta estructura nos deja una matriz la cual las primeras 3 columnas poseen varios elementos distintos de 0, las siguientes columnas pocos elementos distintos de 0 y a partir de la columna $1000 - k$ (siendo k el tamaño del anillo) las columnas poseerían solo 2 elementos no nulos, para finalmente llegar a la ultima columna que poseerá $k - 1$ elementos distintos de uno. Cabe aclarar que la cantidad de elementos no nulos asignados a las columnas que no forman parte del anillo fue distribuida de igual manera al las instancias del tipo 3 Principales. En la figura 2 se muestra un ejemplo de la forma de la matriz resultante.

Figura 2: Gráfico de una matriz de una instancia del tipo 3 principales y un anillo

3.2. Experimento complejidad con instancias Aleatorias

Como primer experimento vamos a realizar un análisis de la complejidad temporal de los distintos métodos implementados, con el fin de comparar la escalabilidad y viabilidad de los mismos. Para esto, vamos a utilizar todas las instancias definidas en la sección instancias de experimentación, con estas instancias, ejecutaremos todos los métodos (5 veces cada una) y tomaremos la media del tiempo total que le llevo cada método[2]. Con estas instancias vamos a graficar los tiempos en función de la cantidad de filas de la matriz y su proporción de elementos no nulos.

Para estas instancias tenemos tres hipótesis:

- Cuando la matriz sea sparse, la complejidad en función de la cantidad de filas del algoritmo eliminación Gaussiana va a tener una complejidad similar a n^3, mientras que los métodos iterativos poseerían una complejidad $n^2 * k$ siendo k la cantidad de iteraciones de los métodos. Además, dependiendo la sparcidad de la matriz, una matriz con muy pocos elementos, eliminación gaussiana podría llegar n^2 y los métodos iterativos a una complejidad $n * k$ Las suposiciones sobre los métodos iterativos se deben a que ambos métodos iterativos realizan una multiplicación Matriz vector k veces.

- Cuando la matriz no sea sparse la complejidad del algoritmo de la eliminación Gaussiana va a tener una complejidad similar a n^3. Mientras que los métodos iterativos poseerían una complejidad $n^2 * k$.

- Además la cantidad de iteraciones que realizan ambos métodos iterativos va a ser relativamente chica comparada con la cantidad de páginas, entonces estos van a ser más eficientes en cuestiones de tiempo que la eliminación gaussiana.

[2]Esto ultimo se realizo en todos los experimentos de complejidad temporal que siguen

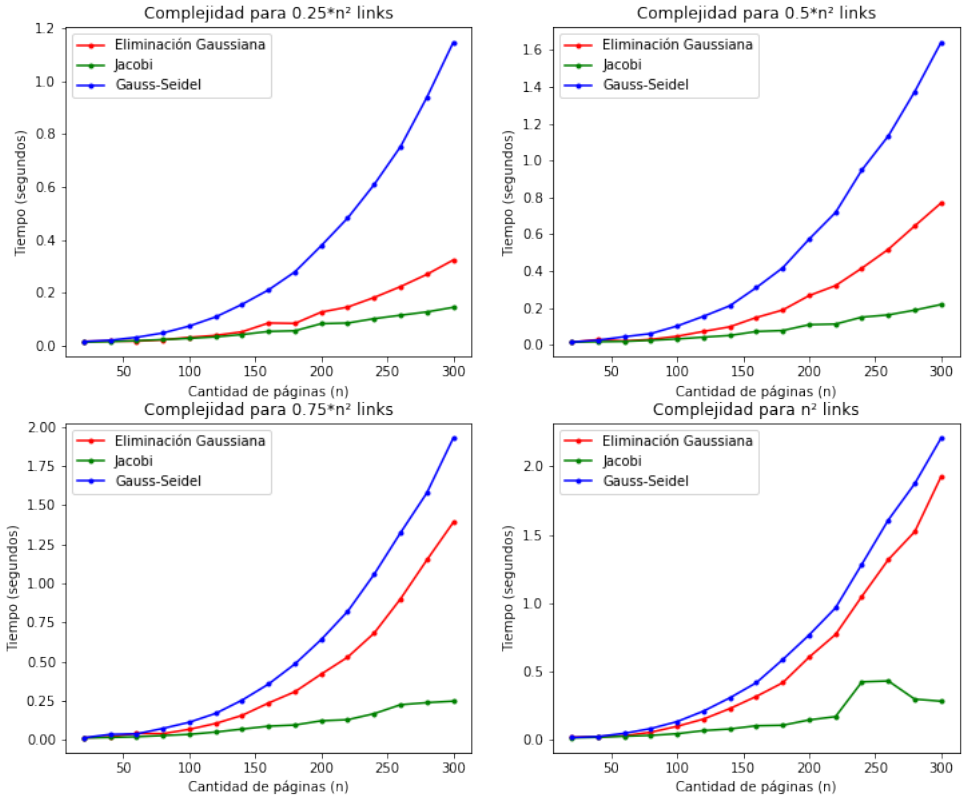

Figura 3: Comparación del tiempo de ejecución de los distintos algoritmos según la cantidad de páginas de la red para diferentes cantidad de links

Como podemos ver en figura 3 contrario a nuestra hipótesis el método de Gauss-Seidel siempre es menos eficiente comprado al resto y el método de Jacobi es el más eficiente. Mientras tanto eliminación gaussiana se comporta de forma más parecida a Jacobi entre más sparce es la matriz mientras que su complejidad aumenta a medida que la matriz tiene más elementos no nulos, hasta asemejarse a la complejidad de Gauss-Seidel.

De esta manera tanto Gauss-Seidel como Jacobi no resultan ser tan dependientes de la densidad de la matriz, la eliminación gaussiana se ve muy afectada cambiando drásticamente su complejidad a medida que aumentamos la cantidad de elementos no nulos en la matriz. Cabe aclarar que la varianza de los datos resultantes de las ejecuciones resulto con una varianza menor a 10^{-4}

3.3. Experimento complejidad instancias 3 Principales

Para estas instancias vamos a graficar los tiempos en función de la cantidad de filas (que es igual a la cantidad de columnas) de la matriz. Nuestra hipótesis es que la estructura de la matriz va a ser muy influyente. Suponemos que la complejidad de los algoritmos en función del tamaño de la matriz, va a ser similar su complejidad en el caso de que la matriz sea sparce con instancias aleatorias. Esto lo suponemos ya que la densidad de la matriz se va a estar

concentrada en solo 3 columnas (por la estructura de la matriz), con lo cual, para el resto de columnas la matriz seguiría siendo sparce.

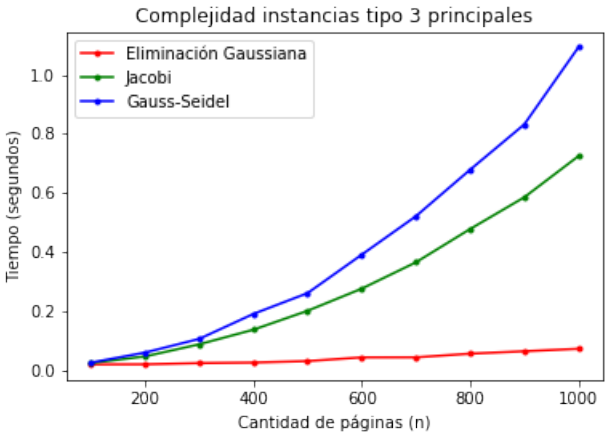

Figura 4: Tiempo de ejecución de cada algoritmo en función de la cantidad de páginas de las instancias del tipo 3 principales

La figura 4 muestra que nuestra hipótesis no fue acertada, si bien la complejidad del método de Gauss-Seidel sigue siendo la peor, con estas instancias, la complejidad de la eliminación gaussiona es notablemente mejor que las de los métodos iterativos. Claramente este tipo de instancias favorece al método de la eliminación gaussiona y es desfavorable para el método de Jacobi.

3.4. Experimento complejidad instancias 3 Principales y un Anillo

Para estas instancias, vamos a graficar los tiempos en función del tamaño del anillo. Nuestra hipótesis es que dada esta estructura de la matriz, como la densidad de la misma continua siendo similar a la del experimento 3 principales, las complejidades serán similares a las del experimento anterior a pesar del anillo.

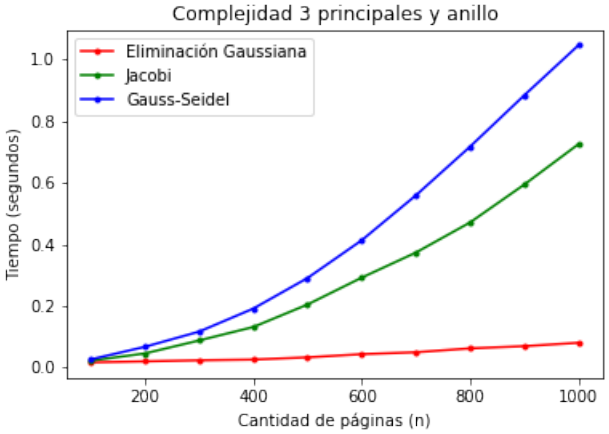

Figura 5: Tiempo de ejecución de cada algoritmo en función de la cantidad de páginas de las instancias del tipo 3 principales y Anillo

La figura 5 muestra que efectivamente las complejidades de los métodos son muy similares a las presentadas en la figura 4.

Esto, sumado a las complejidades obtenidas del experimento instancias Aleatorias nos indica que las complejidades de los mismos, si bien son dependientes de la densidad de las instancias y de la estructura de la matriz, dos estructuras no tan distintas y a su vez con una densidad similar, presentan complejidades similares.

3.5. Experimento complejidad instancias Árbol

Para estas instancias vamos a gráficar los tiempos en función de la cantidad de columnas de la matriz. Nuestra hipótesis es que dada la estructura de las matrices de estas instancias, la complejidad temporal de los métodos iterativos se acercara a $n * k$ y la complejidad del algoritmo de eliminación gussiana sera n^2. Estas estimaciones se basan en que la cantidad de elementos no nulos de la matriz es del orden de n, porque un árbol binario tienen $n-1$ aristas (siendo n la cantidad de nodos).

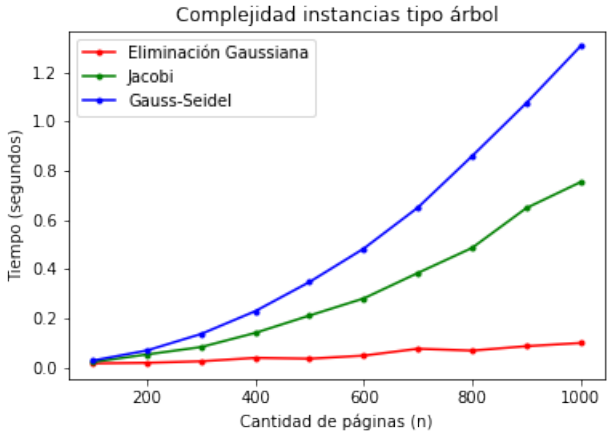

Figura 6: Tiempo que tarda cada algoritmo según la cantidad de páginas de la instancia árbol

Como se puede ver en la figura 6 nuestra hipótesis era errónea. Ocurrió lo contrario a lo esperado, la eliminación gaussiana termina teniendo una complejidad similar a la esperada para los métodos iterativos mientras que estos tuvieron un desempeño peor. Pensamos que esto puede partir de que si bien el grafo grafo tiene $n-1$ nodos, su matriz de adyacencia termina siendo una matriz triangular inferior con a lo sumo un elemento fuera de la diagonal para la fila con la que iteramos y dos por columna, por lo tanto en cada iteración esta realizando una menor cantidad de operaciones comparada con los otros algoritmos.

3.6. Experimento Cualitativo

Para la experimentación cualitativa procederemos a calcular el error absoluto del resultado encontrado por los distintos métodos respecto de los resultados brindados por la cátedra con la instancia test_30_segundos. Para luego graficar un histograma en función del error. Elegimos ésta instancia específica ya que es la más extensa de las instancias brindadas Nuestra hipótesis es que el error de los métodos iterativos sera menor al error ocasionado por el método de la

eliminación gaussiana, esto lo suponemos debido a que las cuentas de este método tienden a arrastrar error.

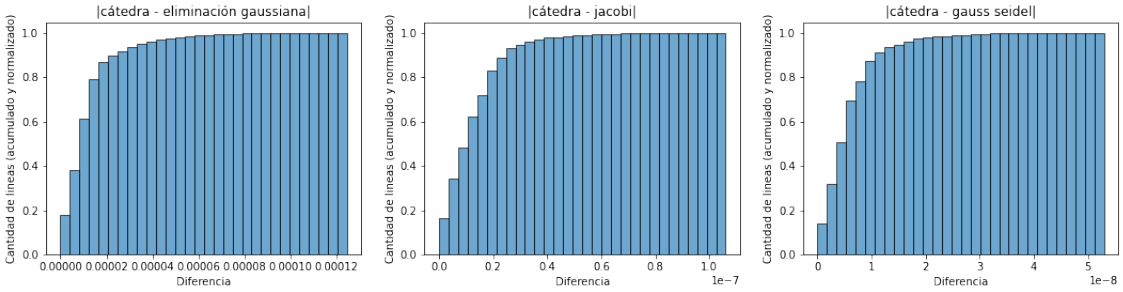

Figura 7: Error absoluto de los distintos métodos respecto de las instancias de la cátedra

Como muestra la figura 7 nuestra hipótesis fue acertada, notar que la escala de los errores de los métodos iterativos esta en $1e^{-7}$ y $1e^{-8}$ respectivamente. La figura 7 nos muestra además que el error generado por los métodos iterativos son parecidos entre si, siendo gauss-seidel es ligeramente mejor.

3.7. Experimento Análisis Convergencia

Para analizar la convergencia de los dos métodos iterativos vamos a calcular la distancia entre el vector provisto por la cátedra y los vectores resultados de cada iteración. Este análisis será realizado con la instancia test_aleatorio. La distancia entre los vectores la calculamos con la norma 2.

Nuestra hipótesis para este análisis es que la convergencia entre los dos métodos va a ser similar.

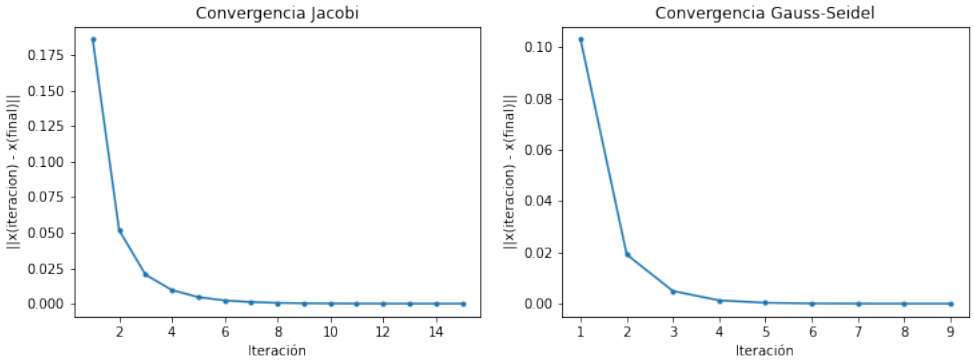

Figura 8: Distancia (norma 2) entre el resultado provisto por la cátedra y el vector resultante de cada iteración

En la figura 8 se puede ver como la convergencia de los vectores es muy veloz, a su vez la cantidad de iteraciones necesarias para el método de Jacobi fueron 14 mientras que con el método de Gauss-Seidel con 9 iteraciones bastó. Cabe aclarar que la teoría no determina una ventaja respecto de la velocidad de convergencia en un método u otro, sino más bien, indica que la cantidad de iteraciones necesarias para cada método depende de la matriz del sistema original.

4. Conclusiones

Los métodos iterativos de Jacobi y Gauss-Seidel resultan aproximar mejor el resultado que la eliminación gaussiana, haciendo que estos sean preferibles cuando se necesita obtener un resultado más acertado, es específico Gauss-Seidel termina obteniendo los resultados con menor error numérico, que además efectivamente convergen al resultado en una cantidad de iteraciones razonable.

Por otro lado en cuanto a la complejidad temporal de los algoritmos Gauss-Seidel fue el algoritmo que tardó más tiempo para todas las instancias, mientras que el método de Jacobi fue siempre el mejor de los métodos iterativos en esta cuestión.

Eliminación gaussiana, en un principio, parecía ser menos eficiente en general comparado con Jacobi Entre menos sparce era la matriz mayor complegidad tenía eliminación gaussiana, mientras que Jacobi parecía mantenerse igual, pero después pudimos probar que si bien en estas instancias el algoritmo no era el más eficiente había algunas con las cual se desempeñaba mucho mejor que el resto en cuestiones de tiempo.

En los casos de tres principales y de la instancia anillo, las matrices de adyacencia de la red tienen la mayor parte de sus elementos concentrados en una parte de esta y el resto siendo muy sparce, dando esto una estructura para la matriz en la que eliminación gaussiana tarda mucho menos que el resto de algoritmos evaluado, además el método de Jacobi empeora mucho su complejidad al tratar con este tipo de matrices haciendo que asemeje su comportamiento a uno más similar al del método de Gauss-Seidel. Entonces estas instancias nos dan la intuición de que si la matriz del sistema $Ax = b$ tiene la mayoría de sus elementos no nulos de un lado eliminación gaussiana tiene una mejor complejidad.

Ahora para las instancias de 3 principales y un Anillo y 3 Principales sucede que no tienen la mayoría únicamente los elementos acumulados en un lado de la matriz si no que estos están bajo la diagonal de esta, al igual que en las instancias Árbol donde eliminación guassiana, Jacobi y Gauss-Seidel se comportaron de una forma similar que en las dos experimentaciones anteriores. Entonces intuimos que estas matrices funcionan de esta forma específicamente porque son triangulares inferiores (en el caso de las instancias de Árbol) o muy similares a una matriz triangular inferior con una matriz muy rala, entonces eliminación gaussiana debe realizar menos operaciones en la resta de filas mientras que los métodos numéricos el coste de multiplicación de su matriz es comparativamente mayor.

Entonces el método de Jacobi parece ser el mejor en casos donde la matriz tiene una distribución de elementos aleatoria tanto en tiempo como en precisión pues la diferencia del error comparada con el error de Gauss-Seidel es muy chica. Por otro lado eliminación gaussiana, si bien su resultado genera un mayor error numérico, es mucho más rápida que el

método de Jacobi para ciertos tipos de matrices y si se sabe que nuestras matrices van a ser como las instancias estudiadas en este trabajo práctico puede resultar conveniente sacrificar la exactitud del algoritmo para ganar en tiempo. Gauss-Seidel resulto ser menos eficiente para las instancias estudiadas pero fue más exacto, si bien el resto de algoritmos termina teniendo un mayor desempeño en cuanto a la complejidad temporal, si necesitamos ser lo más exactos posibles Gauss-Seidel aparenta ser mejor.

5. Referencias

1. Elminacion Gaussiana: https://www.uv.es/~diaz/mn/node29.html

2.