Trabajo Práctico II

Matías Pérez (2/05)
Rodrigo Campos Catelin (561/06)

December 16, 2010

Abstract

El siguiente trabajo analiza las ventajas y desventajas de elegir un método iterativo y un método directo al resolver un sistema de ecuaciones lineales. Se usó como método directo la resolucion mediante factorización LU y como método iterativo el método de Jacobi.

Como problema a resolver se escogió el cálculo del ranking de Page sobre distintos dominios. Dicho problema tiene la característica de que la matriz que representa al sistema de ecuaciones suele ser rala.

Los aspectos tomados en cuenta para el análisis son: el uso de memoria, eficiencia en tiempo, aproximación del resultado y error de cálculo.

En este trabajo se muestra que el consumo de memoria y tiempo de ejecución del método iterativo es ordenes de magnitud menor al del método directo. Por esto, los métodos iterativos hacen posible la resolución de sistemas grandes que con un método directo resultan intratables.

Palabras clave: resolucion de sistemas lineales, factorizacion LU, metodo de jacobis, matriz rala.

Contents

1 Introducción teórica

Sea $A \in \mathbb{R}^{n \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^n$. Consideremos el sistema: $Ax = b$.

1.1 Resolución mediante factorización LU

El método de resolución mediante factorización LU, consta en encontrar dos matrices L y U tales que:

- L sea Triangular Inferior con unos en la diagonal (inversible)
- U sea Triangular Superior
- $A = LU$

De esta forma se reescribe el sistema como: $Ax = LUx = b$, al ser L inversible se procede a resolver el sistema: $Ux = L^{-1}b$. Como U es Triangular Superior la resolución de este sistema es sencilla utilizando la técnica de *back substitution*.

Un método para lograr la factorización LU es el método de Gauss. Dicho método, si existe una factorización LU, la encuentra siempre.

Prop: si la matriz A es estrictamente diagonal dominante, se puede asegurar la existencia de L y U y su inversibilidad.

1.2 Resolución mediante el método de Jacobi

El método de Jacobi pertenece a la familia de métodos iterativos que descompone a A en dos matrices M, K tal que $A = M - K$ con M inversible. Luego:

$$Ax = b$$

$$(M - K)x = b$$

$$Mx - Kx = b$$

$$x = M^{-1}Kx + M^{-1}b$$

Sea $T = M^{-1}K$, $c = M^{-1}b$. Entonces,

$$x = Tx + c$$

En base a esto se plantea el siguiete esquema recursivo:

$$x_{n+1} = Tx_n + c$$

Prop: si la sucesión $\{x_n\}$ converge $\Rightarrow \lim_{n \to \infty} x_n = x$.

Prop: si $\rho(T) < 1$, dicha sucesión converge.

Prop: sea $\|.\|$una norma matricial inducida$\Rightarrow \rho(T) < \|T\|$.

Corolario: sea $\|.\|$una norma matricial inducida, si $\|T\| < 1$, dicha sucesión converge.

En el método de jacobi se considera $M = D$ y$K = L + U$, donde:

- $d_{ij} = \begin{cases} a_{ii} & i = j \\ 0 & i \neq j \end{cases}$
- $l_{ij} = \begin{cases} a_{ij} & i > j \\ 0 & i \leq j \end{cases}$
- $u_{ij} = \begin{cases} a_{ij} & i < j \\ 0 & i \geq j \end{cases}$

2 Desarrollo

2.1 Problema

El ranking de Page calcula, para un determinado conjunto de n páginas web, la probabilidad de que un navegante aleatorio se encuentre en cada una de ellas. Para ello definimos a $W \in \mathbb{R}^{n \times n}$ como la *matriz de conectividad* ignorando "autolinks", $p = 0.85$ la probabilidad de que el navegante siga uno de los links de la página en la que se encuentra y $c_j = \sum_i w_{ij}$ la cantidad de links salientes de la página j.

Por características del dominio al que se aplicará el ranking de Page (conjunto de páginas web), la matriz W resulta ser una matriz rala.

Si definimos $A \in \mathbb{R}^{n \times n}$como:

$$a_{ij} = \begin{cases} \frac{1-p}{n} + \frac{pw_{ij}}{c_j} & c_j \neq 0 \\ \frac{1}{n} & c_j = 0 \end{cases}$$

Entonces, el problema se reduce a resolver el siguiente sistema:

$$Ax = x$$

tal que $x_i \geq 0$ y $\sum_i x_i = 1$

Es importante notar que la matriz A puede reescribirse como:

$$A = pWD + ez^\intercal$$

donde D es una matriz diagonal de la forma:

$$d_{jj} = \begin{cases} \frac{1}{c_j} & c_j \neq 0 \\ 0 & sino \end{cases}$$

e es un vector columna de unos de dimensión n y $z \in \mathbb{R}^n$

$$z_j = \begin{cases} \frac{1-p}{n} & c_j \neq 0 \\ \frac{1}{n} & sino \end{cases}$$

Luego, el sistema:

$$Ax = x$$

$$(pWD + ez^\intercal)x = x$$

$$pWDx + ez^\intercal x = x$$

$$pWDx - x = -ez^\intercal x$$

$$x - pWDx = ez^\intercal x$$

$$(I - pWD)x = ez^\intercal x$$

Se define $\gamma = z^\intercal x \in \mathbb{R}$. De esta manera, el procedimiento sugerido por la cátedra para resolver el sistema es:

1. Suponer $\gamma = 1$
2. Resolver la ecuacion $(I - pWD)x = e\gamma$
3. Normalizar el vector x de manera que $\sum_{i=1}^{n} x_i = 1$

Propiedad: $\forall j \in \mathbb{N}, 1 \leq j \leq n, \sum_{i=1}^{n} (WD)_{ij} = 1$

Demostración: Sea $j \in \mathbb{N}, 1 \leq j \leq n$

$$\sum_{i=1}^{n} (WD)_{ij} = \sum_{i=1}^{n} \frac{w_{ij}}{c_j} = \frac{1}{c_j} \sum_{i=1}^{n} w_{ij} = \frac{1}{\sum_{i=1}^{n} w_{ij}} \sum_{i=1}^{n} w_{ij} = \frac{\sum_{i=1}^{n} w_{ij}}{\sum_{i=1}^{n} w_{ij}} = 1$$

Corolario: $\forall j \in \mathbb{N}, 1 \leq j \leq n \sum_{i=1}^{n} (pWD)_{ij} = 0.85$

Corolario: $\forall j \in \mathbb{N}, 1 \leq j \leq n, \sum_{i=1}^{n} |(pWD)_{ij}| = 0.85$. Pues $(pWD)_{ij} \geq 0$.

2.2 Resolución mediante factorización LU

Existencia de factorización LU

Mostraremos que la matriz $(I - pWD)$ es estrictamente diagonal dominante. Es decir que $\forall j \in \mathbb{N}, 1 \leq j \leq n$, vale:

$$|(I - pWD)_{jj}| > \sum_{i=1, i \neq j}^{n} |(I - pWD)_{ij}|$$

Primero veamos que como $p(WD)_{jj} = 0$

$$|(I - pWD)_{jj}| = |I_{jj}| = 1$$

Por otro lado, dado que $i \neq j \Rightarrow I_{ij} = 0$. Entonces:

$$\sum_{i=1, i \neq j}^{n} |(I - pWD)_{ij}| = \sum_{i=1, i \neq j}^{n} |I_{ij} - (pWD)_{ij}| = \sum_{i=1, i \neq j}^{n} |(pWD)_{ij}| = 0.85$$

En consecuencia

$$1 = |(I - pWD)_{jj}| > \sum_{i=1, i \neq j}^{n} |(pWD)_{ij}| = 0.85$$

Luego, la matriz $(I - pWD)$ es estrictamente diagonal dominante. Lo que implica, por la propiedad antes expuesta, que existe una factorización LU.

Implementación

Se utilizó el método de eliminación Gaussiana para hallar U y $L^{-1}b$. Mediante el mismo se logró conseguir un sistema de ecuaciones equivalente cuya matriz asociada es triangular superior. Finalmente, para hallar el vector solución de este sistema equivalente, se utilizó la técnica de backsubstitution.

Para resolver el sistema, se agregó a la matriz el vector "b" como última columna. De esta forma, al triangular la matriz, en cada paso se tiene un sistema equivalente al anterior. Obteniendo al finalizar la matriz U y el vector $L^{-1}b$ buscados. El resto de la implementación no trajo mayores complicaciones.

El método de eliminación Gaussiana tiene la particularidad de que si la matriz es rala no conserva esta propiedad. Por este motivo, se decidió representar la matriz como filas contiguas en memoria, guardando todos sus elementos. Esto resulta en un consumo de memoria $O(n^2)$ para la matriz. Vale recordar que el órden del método es $O(n^3)$.

También se decidió utilizar pivoteo parcial para reducir el error numérico, aunque, como ya se mostró en secciones anteriores, esto no era necesario para encontrar una factorización PLU dado que la matriz tiene factorización LU.

2.3 Resolución mediante el método de Jacobi

Convergencia del método

Para ver la convergencia del método es suficiente ver que $\|T\| < 1$, para $\|.\|$ norma matricial inducida y $T = D^{-1}(L + U)$. Por lo visto en la introducción teórica.

Tomaremos $\|.\|_1$. En resumen, queremos ver que:

$$\|I^{-1}(-pWD)\|_1 < 1$$

$$\|I^{-1}(-pWD)\|_1 = |-1|\|I^{-1}pWD)\|_1 = \|pIWD)\|_1$$

$$\|pWD)\|_1 = \max_j \sum_{i=1}^{n} |(pWD)_{ij}| = 0.85 < 1$$

De esta manera queda asegurada la convergencia del método.

Implementación

En contraposición con el método de eliminación Gaussiana, este método no modifica la matriz. Como consecuencia inmediata, si la matriz es rala, lo será en toda la ejecución del algoritmo. Para aprovechar esto y disminuir el requerimiento de memoria, a la vez que se reducen la cantidad de operaciones, se decidió representar solo los elementos no nulos de la matriz.

Para lograr dicha representación se decidió usar 2 diccionarios anidados. El exterior es un diccionario cuyas claves son las columnas no nulas y sus valores son la representación de la columna correspondiente. Dichas columnas se representan mediante el diccionario interior. Éste tiene como claves las filas no nulas de esta columna y como valor, el elemento correspondiente a esa fila y columna.

El algoritmo, por su naturaleza iterativa, en cada iteración genera una solución que finalmente será la solución del sistema. Cada iteración del algoritmo consta de, a partir de la matriz y la solución anterior, generar la siguiente de manera tal que el elemento i del vector generado es el que resuelve la siguiente ecuación:

$$a_{ii}X_i + \sum_{j=1, j\neq i}^{n} a_{ij}Y_j = b_i$$

siendo X_i la incógnita y Y la solución generada en la iteración anterior.
Quedando finalmente,

$$X_i = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{ij}Y_j}{a_{ii}}$$

Se sabe que b es un vector de unos y $a_{ii} = 1$ para todo i. Entonces queda:

$$X_i = 1 - \sum_{j=1, j\neq i}^{n} a_{ij}Y_j$$

Para que el algoritmo sea eficiente, se recorre el vector Y una sola vez, acumulando en cada paso, en la correspondiente coordenada de X_i, el valor de $-a_{ij}Y_j$, solo para los elementos a_{ij} no nulos.

Criterios de parada

El algorimo de Jacobi es un algoritmo iterativo, como tal se le debe definir un criterio de parada. El criterio para considerar en este caso es si X_i cambia entre dos iteraciones consecutivas. Para esto es necesario determinar si dos vectores son iguales teniendo en cuenta cierta tolerancia, es decir si la distancia entre ambos vectores es menor a cierta cota superior. Para calcular la distancia entre los dos vectores se utilizaron distintas normas de la resta de ambos:

- Norma uno: suma de los módulos de las coordenadas
- Norma infinito: máximo módulo de las coordenadas

- Norma dos: la raiz cuadrada de la suma de los cuadrados de las coordenadas

Dado que $\|.\|_\infty \leq \|.\|_2 \leq \|.\|_1$y se restringe el conjunto con una cota superior resulta ser la norma uno la más restrictiva y la infinito la menos restrictiva entre las tres.

Además se puede especificar opcionalmente una cota superior en la cantidad de iteraciones.

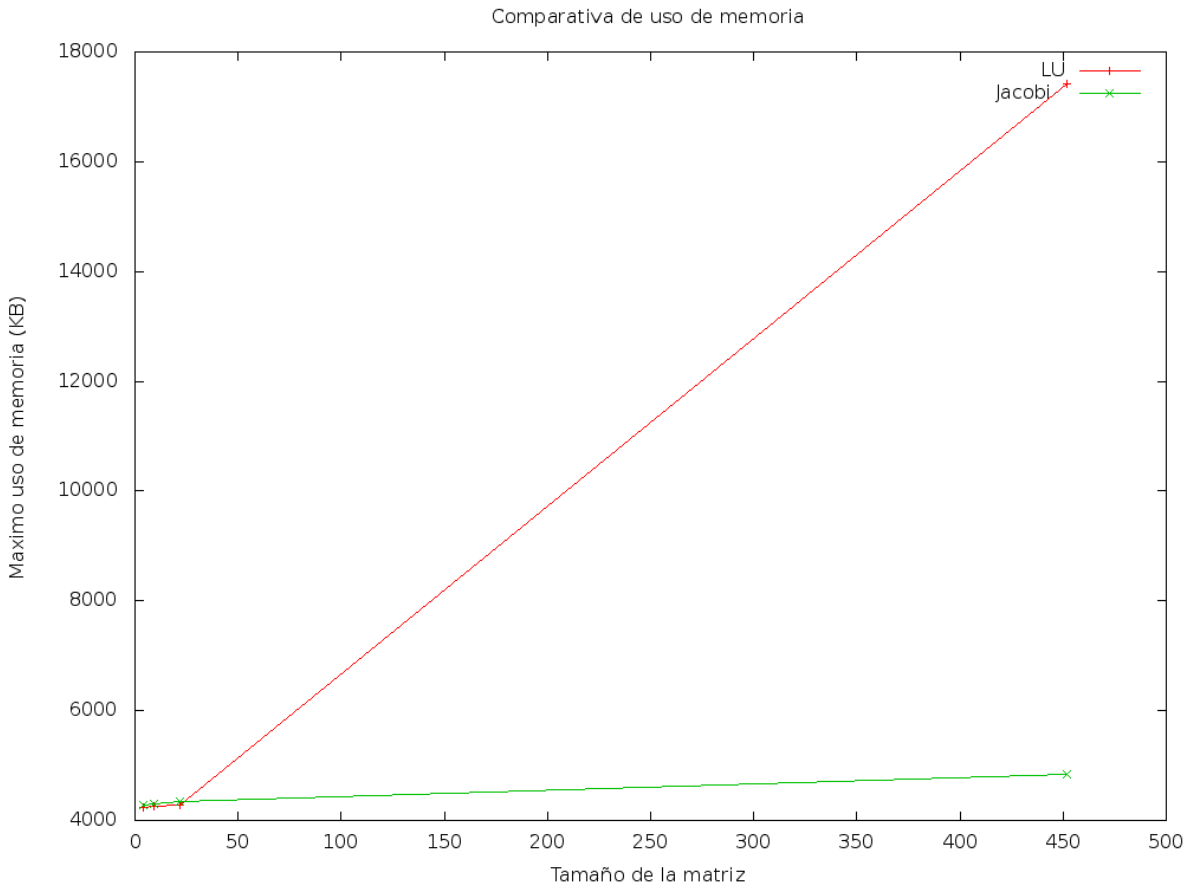
3 Resultados

Para analizar el comportamiento de los algoritmos y los metodos implementados, se realizaron distintos experimentos. A continuacion se incluyen gráficos sobre los comportamientos más relevantes.

Las matrices para los gráficos de memoria y tiempo de ejecución son (en orden ascendente en cuanto al tamaño de la matriz):

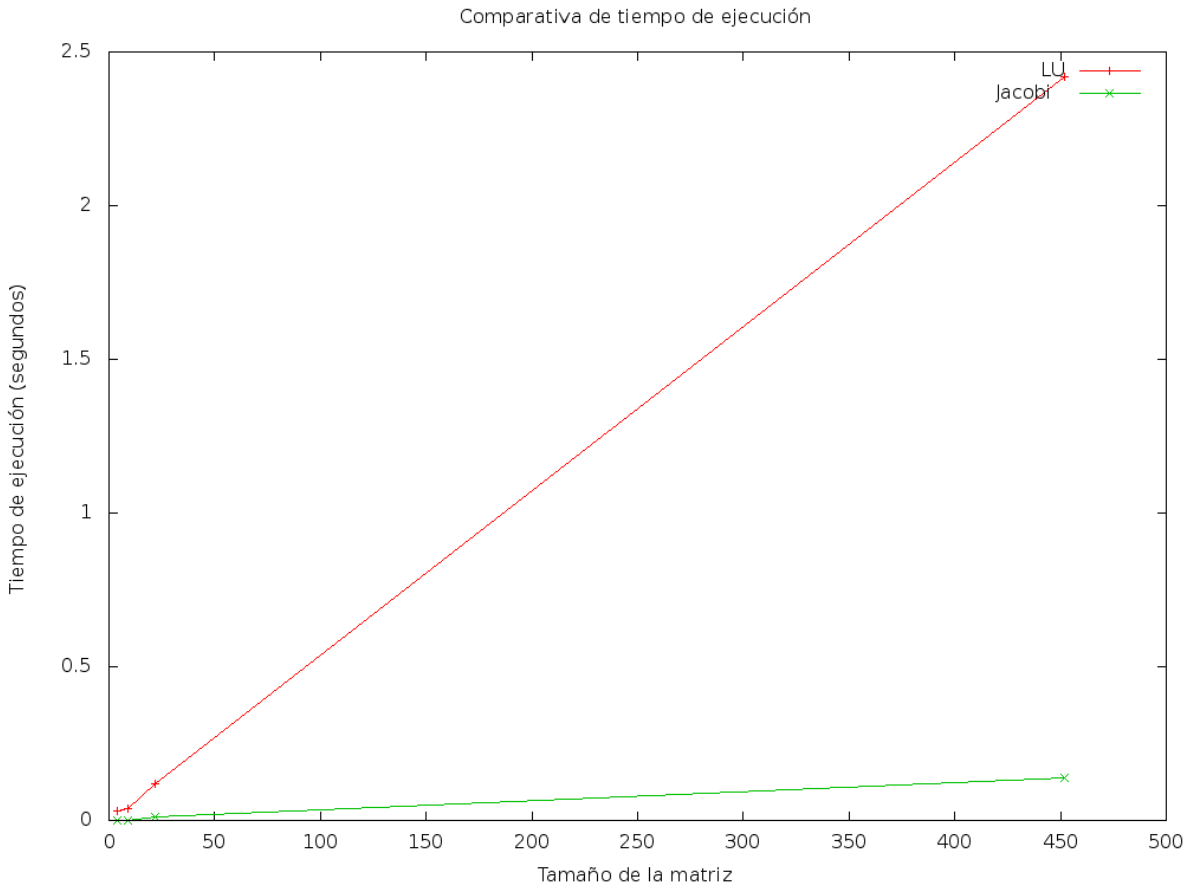
- dc4 (4 páginas)
- ejemplo2 (7 páginas)
- dc9 (9 páginas)
- ejemplo1 (11 páginas)
- dc22 (22 páginas)
- dc452 (452 páginas)
- extwiki-20100920 (2395 páginas)
- iawiki-20100910 (7182 páginas)
- obama.transition.05.11.2009-text (30002 páginas)
- US.CS.2004-11-text (451006 páginas)

Para los gráficos comparativos en performance temporal y espacial de los dos algoritmos sólo se incluyeron las 6 primeras matrices, ya que la memoria requerida por el algoritmo *LU* para las siguientes excede a la disponible en las maquinas de prueba.



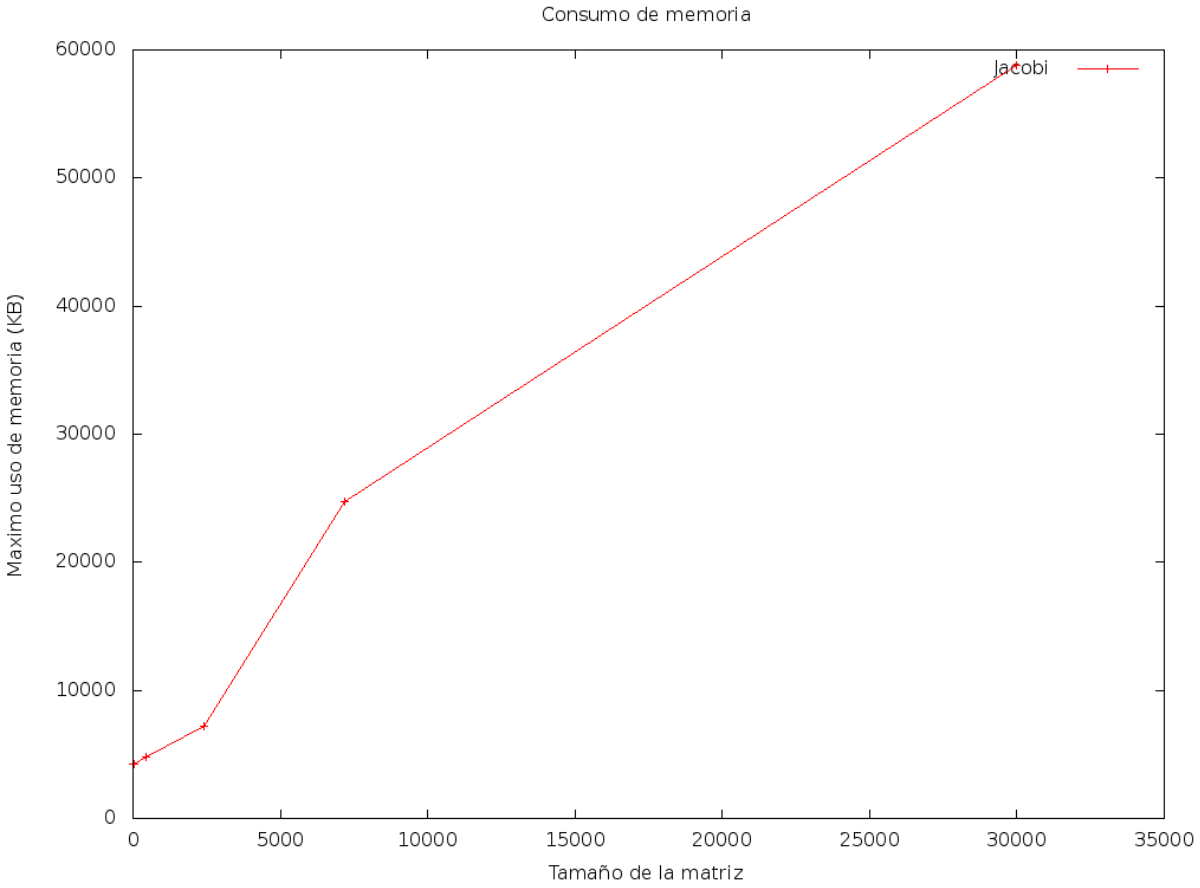
En este gráfico se puede observar el consumo máximo de memoria de ambos algoritmos al variar el tamaño de la matriz de entrada.

Para tomar las mediciones se utilizó el comando *time* de GNU con el parámetro *-f "%M"* que devuelve el consumo máximo de memoria utilizada por el programa en toda su ejecución.



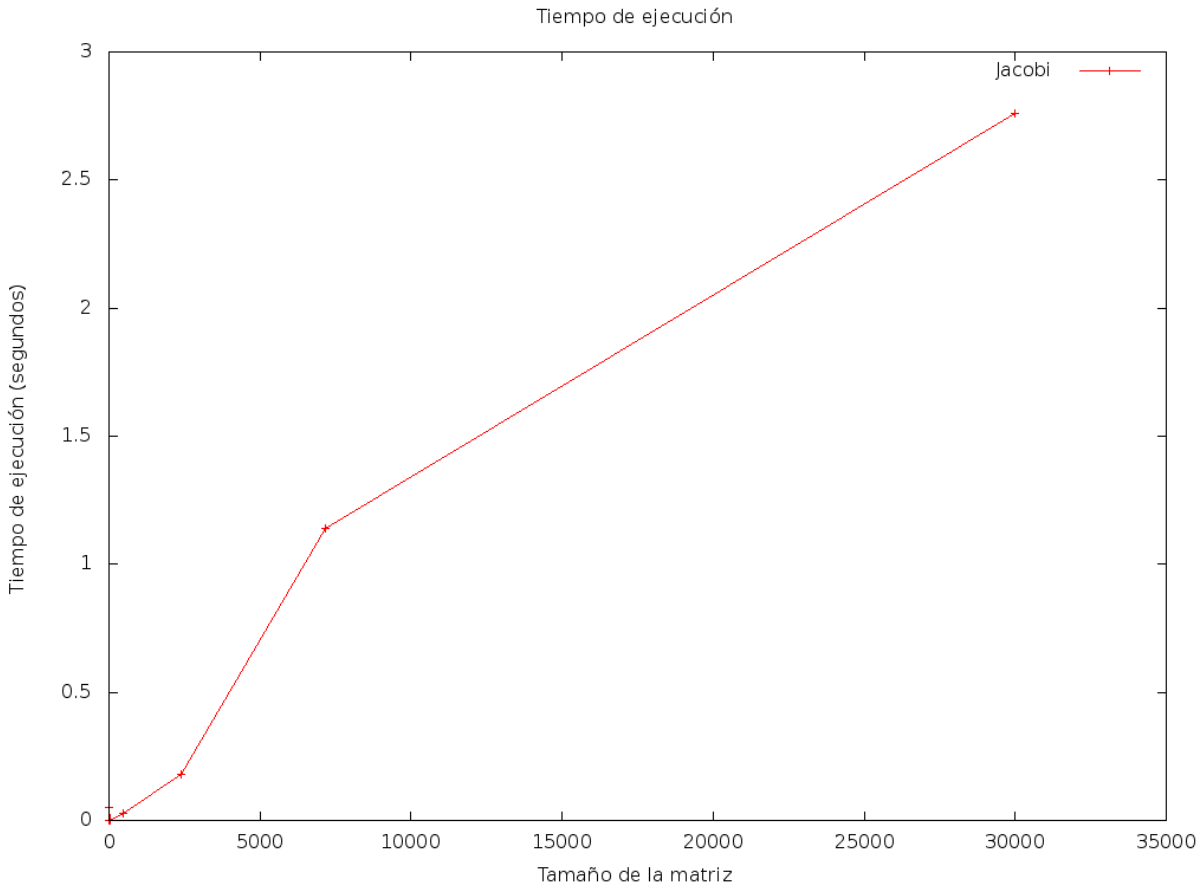
En este gráfico se puede observar cómo varía el tiempo de ejecución al cambiar el tamaño de la matriz de entrada.

Para tomar las mediciones se utilizó el comando *time* de GNU con el parámetro *-f "%e"* que devuelve el tiempo total de ejecución.

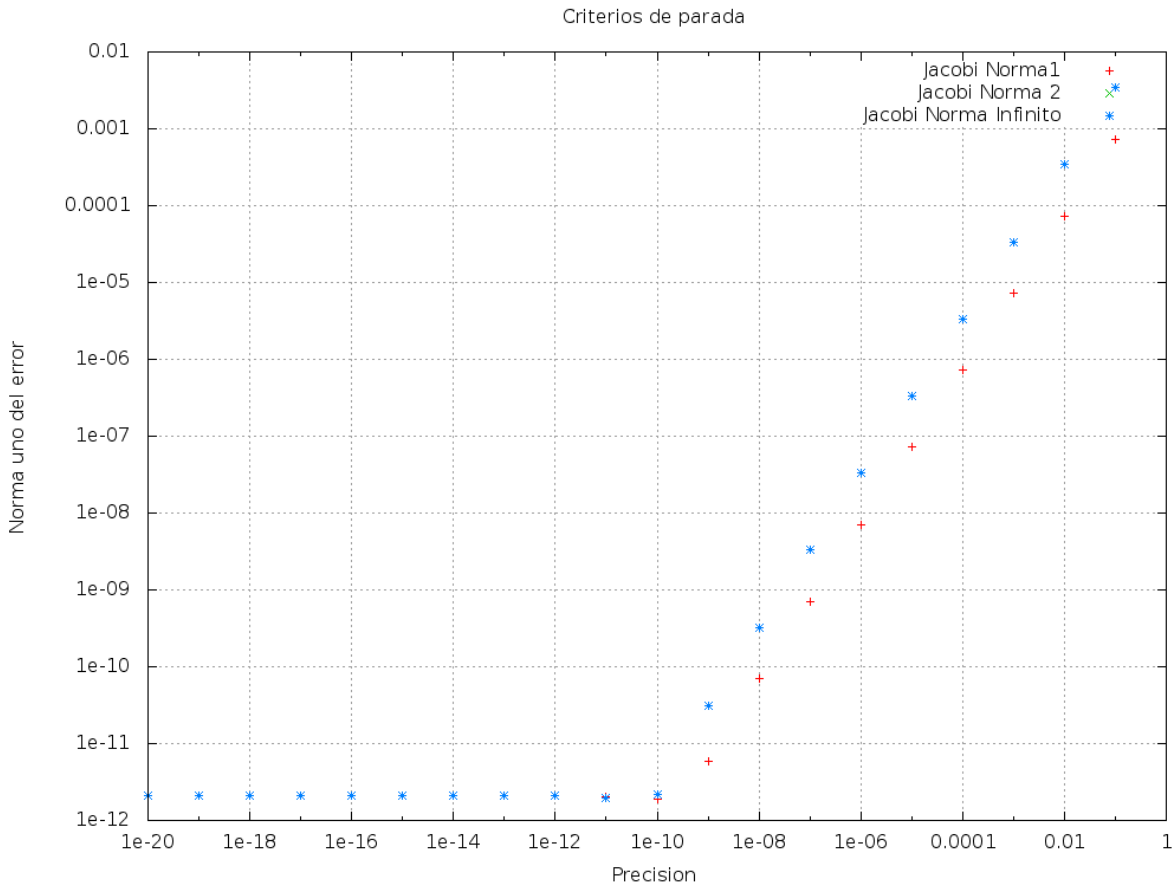


En este gráfico se puede obsevar cómo varía el consumo máximo de memoria del algoritmo de Jacobi en función del tamaño de la matriz para matrices grandes.

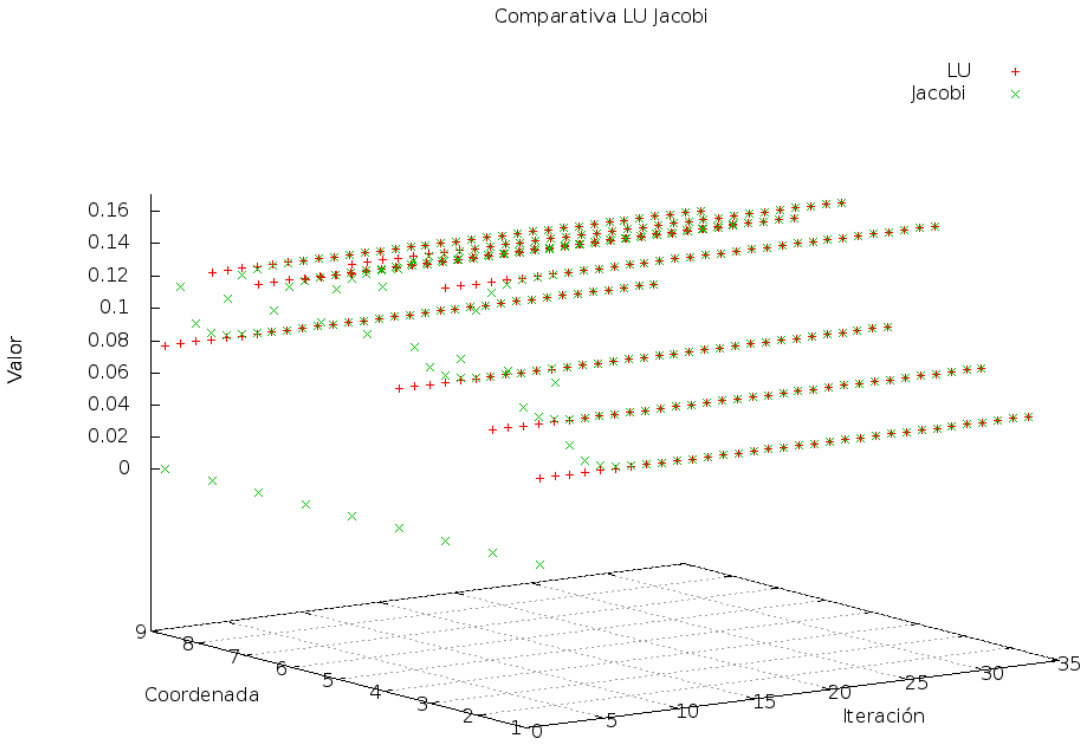
En este gráfico no se muestra el consumo de memoria del algoritmo LU ya que, como se dijo anteriormente, dicho consumo excede la cantidad de memoria disponible en las máquinas de prueba.



En este gráfico se muestra el tiempo de ejecución del algoritmo de Jacobi para matrices grandes. Se excluye el algoritmo de LU por lo mencionado en el caso anterior.



En este gráfico se muestra el error que comete el algoritmo de Jacobi para una misma cota de precisión al considerar los diferentes criterios de parada. Para calcular el error se tomo la norma uno de la diferencia entre el valor alcanzado por el algoritmo de Jacobi y el valor exacto calculado por el algoritmo LU.



En este gráfico se muestra los diferentes resultados obtenidos por el algoritmo de Jacobi al aumentar la cantidad de iteraciones y el valor exacto obtenido mediante el método de LU. Se detalla en el mismo coordenada por coordenada. Para realizar este gráfico se utilizó el set de prueba *dc9*.

A continuación se presentaran gráficos de casos de prueba, en ellos se detallará la distribucion de las páginas y el resultado calculado del ranking de Page utilizando el algoritmo de Jacobi.

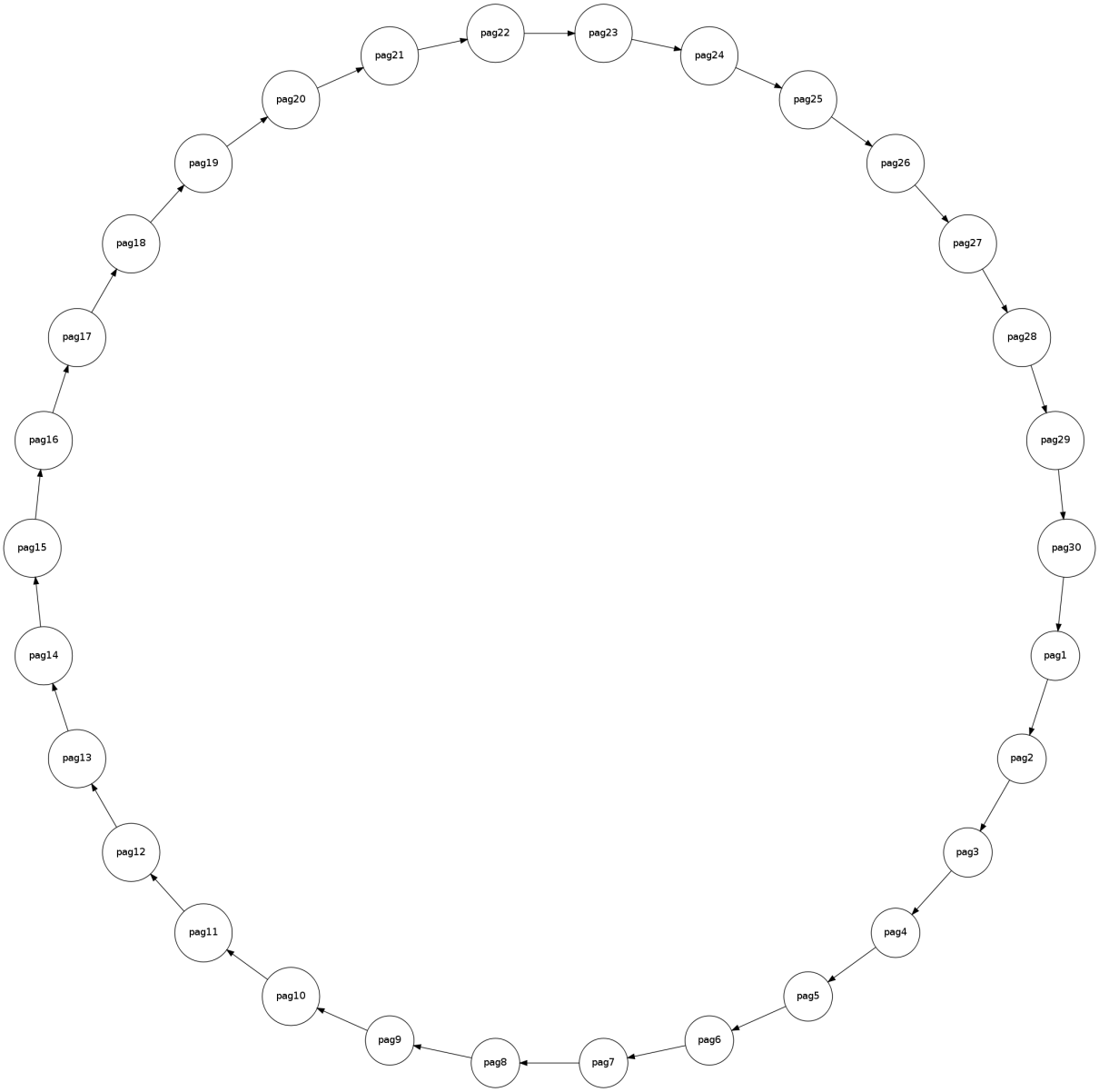


Figura 1: Ejemplo 1

Resultado obtenido: Todas las páginas tiene probabilidad $\frac{1}{30}$.

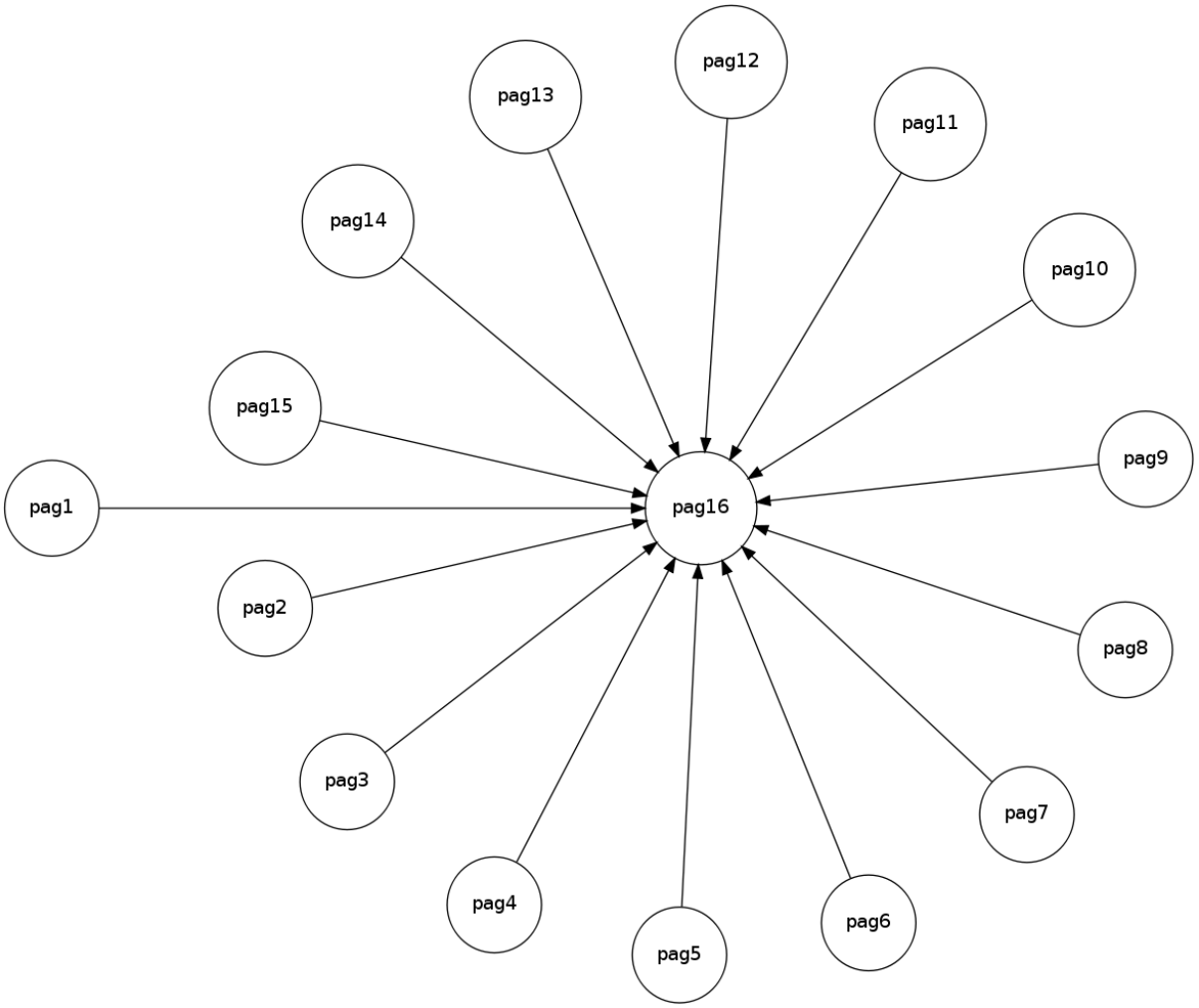


Figura 2: Ejemplo 2

Resultado obtenido: Las páginas 1 a 15 tienen probabilidad 0,03478260 y la página 16 tiene probabilidad 0,47826086.

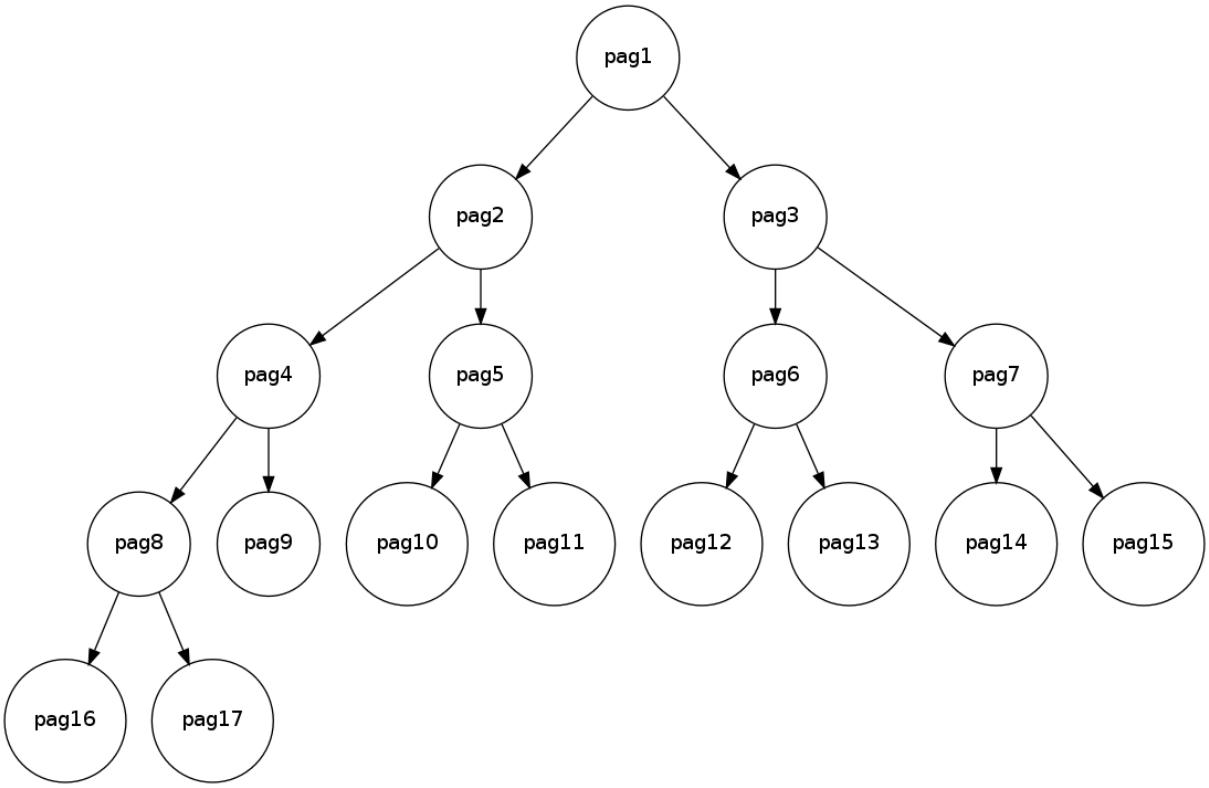


Figura 3: Ejemplo 3

Resultado obtenido:

Páginas	1	2-3	4-7	8-15	16-17
Resultado	0.0368	0.0524	0.0591	0.0619	0.0631

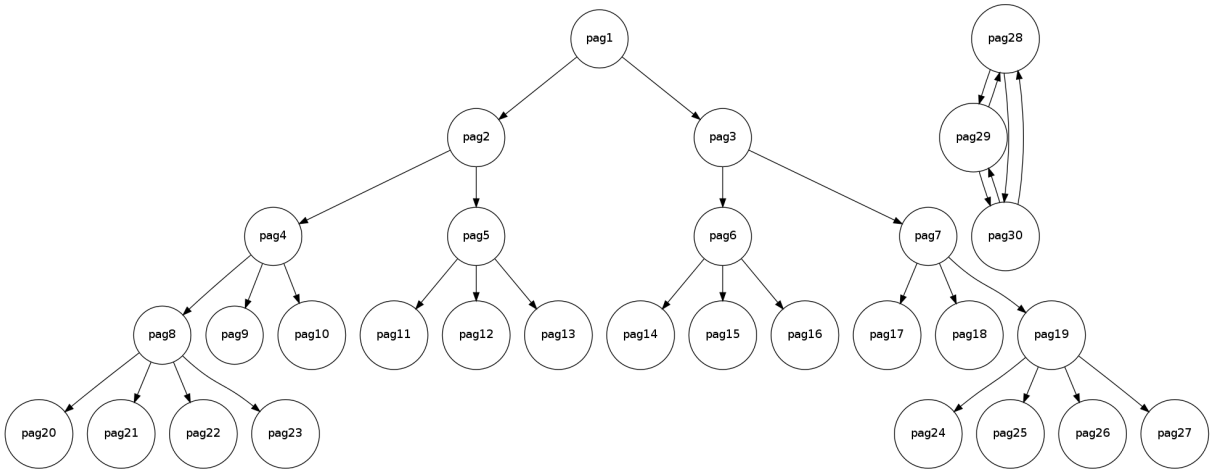


Figura 4: Ejemplo 4

Resultado obtenido:

Páginas	1	2-3	4-7	8-19	20-27	28-30
Resultado	0.0171	0.0244	0.0275	0.0249	0.0224	0.1145

4 Discusión

Luego de obtener los resultados presentados en la sección anterior, se pueden obsevar distintos aspectos tanto de los algoritmos como de los métodos. Los mismos serán discutidos a continuación.

4.1 Consumo de memoria

En el primer gráfico expuesto se puede ver como el consumo de memoria del método *LU* es ordenes de magnitud superior al del método de jacobi. También se puede ver que para matrices pequeñas el consumo resulta similar, pero al aumentar el tamaño de la matriz, la diferencia aumenta en forma notable, debido a que el orden de consumo de memoria del algoritmo *LU* es cuadrático.

El orden del consumo de memoria del algoritmo de jacobi también es cuadrático en peor caso. Pero este caso es muy dificil de encontrar en el dominio del problema ya que las matrices son típicamente ralas. Entonces, el consumo esperado de memoria se acercaría a un consumo de orden lineal en el tamaño de la matriz.

En el tercer gráfico expuesto se puede analizar el comportamiento del algoritmo de Jacobis para matrices más grandes que las soportadas por el algoritmo de *LU*. Se ve como si bien crecen los requerimientos de memoria, este crecimiento es aceptable en función del tamaño de la matriz.

4.2 Tiempo de ejecución

En el segundo gráfico expuesto se puede observar claramente que el tiempo de ejecución del método *LU* es, también, ordenes de magnitud superior al del método de Jacobi. A diferencia del anterior, para matrices pequeñas, ya se nota una diferencia no despreciable en el tiempo de ejecución. Siendo aún mayor para tamaños considerables de matrices.

En el cuarto gráfico se puede ver como el algoritmo de Jacobi tiene una performance superior al la del algoritmo *LU*. Se puede ver que el crecimiento no es excesivo en función del tamaño de la matriz, permitiendo así trabajar con matrices más grandes que *LU*. En particular es interesante notar que el algoritmo de Jacobi tarda poco más de 2.5 segundos para procesar una matriz de $300000x30000$, mientras que el algoritmo *LU* tarda un tiempo similar (apenas menos de 2.5 segundos) para una matriz de $450x450$. Es decir, el algoritmo de Jacobis tarda un tiempo similar al algoritmo de *LU* para procesar una matriz dos ordenes de magnitud mas grande.

4.3 Criterios de parada

Dado que los diferentes criterios de parada no modifcan el comportamiento del algoritmo (ya que sólo determinan cuantas iteraciones se realizan) la elección de uno u otro sólo afectará la presición del resultado final que se logra. En el gráfico de criterios de parada se puede ver cómo usando la *Norma2* y la *NormaInfinito* se logra para una misma cota resultados con menor presición

que usando la *Norma1*, sin embargo se decidió que la *Norma2* es una buena norma para utilizar en el criterio de parada, ya que es una buena forma de calcular distancia entre dos vectores y no resulta una norma tan restrictiva como la *Norma1*. De todas maneras con cualquiera de las tres normas planteadas se vio que se llega a un buen resultado y que el algoritmo devuelve un resultado apropiado para presicion elegida.

4.4 Correctitud

En el gráfico *"Comparativa LU Jacobi"* se compara el resultado alcanzado al utilizar el algoritmo de Jacobi con el obtenido mediante el algoritmo LU. Se puede ver que en las primeras iteraciones el algoritmo de Jacobi se encuentra lejos del resultado calculado por LU, pero al avanzar en la cantidad de iteraciones el primero se acerca al valor del segundo. En conclusión se ve que, dada una cota suficientemente restrictiva, el algoritmo de Jacobi obtendrá el mismo resultado que el algoritmo de LU.

En los ultimo gráficos se presentan casos de prueba más simples y su resultado.

En el *Ejemplo1* se ve una disposición de páginas en una lista circular, como es de esperar al ser indistinguibles los nodos, el ranking de Page para todas resulta ser el mismo.

En el *Ejemplo2* se ve una disposición el la que todas las páginas apuntan a una misma página, en este caso la página con mayor probabilidad resulta ser esta última, ya que desde cualquier página sólo se puede ir a ésta.

En el *Ejemplo3* se presenta una distribución de árbol binario, en este caso se puede observar como a medida que se baja en un nivel del árbol la probabilidad aumenta con respecto al nivel anterior. También se ve que las página en un mismo nivel poseen la misma probabilidad. Esto se corresponde con la noción del ranking de page, ya que a una página dada, se puede llegar por medio de los ancestros de dicha página, por lo que la probabilidad depende directamente de esto.

En el *Ejemplo4* se ve por un lado una distribución de árbol en el que, a medida que se disminuye en el nivel del árbol, la cantidad de hijos aumenta y, por el otro, un grafo K_3. En los resultados obtenidos se ve que en los dos primeros niveles del árbol la probabilidad aumenta, mientras que en los los siguientes niveles (donde las páginas tienen más de dos hijos) la probabilidad disminuye. Esto se debe a que la probabilidad de pasar de un padre a un hijo disminuye a medida que aumenta la cantidad de hijos, por lo que la probabilidad de estar en un nodo con muchos hermanos es menor que la de estar en un nodo con menos hermanos. Por otro lado se puede ver que la mayor probabilidad se logra en la isla del grafo K_3. Ésto es debido a que una vez que se llega a esa isla, (dado que la probabilidad de continuar por los links es mayor a la probabilidad de saltar a una página cualquiera) es probable que se continúe en esa isla.

Es decir, el ranking de Page funcionó como era esperado en estos ejemplos, si bien no fue totalmente trivial pensar cuál era el ranking de Page esperado en algunos casos.

5 Conclusiones

Luego de analizar todos los resultados se llegó a la conclusión de que para matrices suficientemente grandes, los algoritmos exactos como el caso de LU resultan ser prohibitivos, ya sea tanto por el consumo de memoria que el mismo requiere, como por su performance temporal. Es por esto que los métodos iterativos resultan de gran importancia. Cuando se puede asegurar su convergencia, como es el caso de nuestro domino, se puede lograr resolver un sistema de ecuaciones que de otra forma sería imposible de resolver.

También vale decir que aunque se esté en un dominio en el que no sea posible asegurar su convergencia, un método iterativo puede ser una buena opción, ya que un método exacto puede resultar ser inutilizable para contextos con matrices suficientemente grandes, que suelen ser los casos interesantes de resolver. En estos casos un método iterativo puede llegar a ser a lo mejor que se pueda acceder. Y si bien quizá no se puede asegurar la convergencia del método, sí se puede conocer la norma del residuo ($b - Ax$). Entonces pueden ser una excelente alternativa aún cuando no se puede asegurar su convergencia si, para ese caso, alcanza con acotar el residuo.

Los algoritmos iterativos resultan una alternativa performante, tanto en memoria como en tiempo, a los algoritmos exactos tradicionales y hacen posible trabajar con matrices que son intratables con un método directo.

6 Apéndice A

7 Apéndice B

Para compilar todos los métodos se debe ejecutar el comando "make". Se necesita tener el compilador gcc. Este comando generará los siguientes archivos ejecutables:

- jacobis
- lu

Al ejecutarlos[1], los mismos darán un detalle en pantalla sobre los parámetros de entrada, estos son <*pag-file*> <*link-file*> en el caso del algoritmo de LU y <*pag-file*> <*link-file*> *[cota] [iteraciones_max]* en el caso del algoritmo de Jacobi. Los parámetros *cota* y *iteraciones_max* son opcionales, por default la cota es 10^{-8} y la cantidad de iteraciones es n^2. Si se especifica 0 como cantidad máxima de iteraciones, se ignora este parámetro y el criterio de parada sólo tendrá en cuenta la cota.

Los algoritmos imprimen en pantalla el resultado final obtenido para los parámetros dados.

[1] En linux se ejecuta: ./<nombre>