DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 3: Métodos Iterativos

Noviembre 24, 2022 Métodos Numéricos

Grupo 18

Integrante	LU	Correo electrónico
Vekselman, Natán	338/21	natanvek11@gmail.com
Arienti, Federico	316/21	fa.arienti@gmail.com
Manuel Lakowsky	511/21	mlakowsky@gmail.com
Brian Kovo	1218/21	brian.ilank@gmail.com

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

RESUMEN

Los *métodos iterativos* representan una forma alternativa y eficiente para la resolución de sistemas lineales. Dadas las condiciones necesarias, permiten aproximar el resultado de un sistema en tiempo cuadrático, sin amortización. Es decir, logran una complejidad menor a la que se logra con la mayoría de los otros métodos convencionales —como lo son aquellos que requieren un paso previo de factorización—, cuyo costo no amortizado suele ser cúbico.

En este trabajo evaluaremos la eficiencia de dos métodos iterativos: el método de *Jacobi* y el método de *Gauss-Seidel*, como alternativas para la resolución del algoritmo de *PageRank* —desarrollado en el tp1— sobre una serie de casos de test. Para ello, se propondrá una posible implementación en C++ de ambos métodos y se contrastará su eficiencia con una tercera implementación basada en el método de la *eliminación gaussiana* con *sustitución inversa*.

A su vez, extenderemos éste análisis para evaluar el comportamiento de los tres métodos en función de la *densidad* del grafo de entrada, para distintas familias de redes.

Palabras clave: método de Jacobi, Gauss-Seidel, Eliminación Gaussiana, PageRank.

CONTENIDOS

1. Introducción teórica 2
1.1. Métodos iterativos 2
1.2. Implementación 3
1.2.1. Método de Jacobi 3
1.2.2. Método de Gauss-Seidel 4
1.2.3. Eliminación Gaussiana 5
1.2.4. Representación de matrices 5
2. ¿Por qué usamos $vector < SparseVector >$ como representación? 6
3. Evaluación de Convergencia 8
3.1. Casos de test 8
4. Evaluación temporal 13
4.1. Casos de test 13
4.2. En función de la densidad del grafo de entrada 13
5. Conclusiones 14
6. Apéndice 15

1. Introducción teórica

1.1. **Métodos iterativos.** Los *métodos iterativos* son procedimientos que nos permiten resolver algunos sistemas de ecuaciones lineales del tipo $\mathbf{A}x = b$. Contrario a los *métodos exactos* —como la *Eliminación Gaussiana*— que obtienen su resultado en un número finito de pasos, los métodos iterativos generan una sucesión $\{x^{(k)}\}_{k\in\mathbb{N}_0}$ que, de converger, lo hace a la solución del sistema.

Como esquema básico, dado un $x^{(0)}$ inicial, se define de manera genérica una sucesión iterativa $\{x^{(k)}\}_{k\in\mathbb{N}_0}$ de la siguiente manera:

$$x^{(k+1)} = \mathbf{T}x^{(k)} + c \tag{1}$$

donde $\mathbf{T}$ se denomina *matriz de iteración* y c es un vector. En particular, $x^{(k)}$ va a converger a la solución de un sistema particular, para cualquier vector $x^{(0)}$ inicial, si y sólo si el radio espectral de la matriz de iteración $\mathbf{T}$ es menor a 1. Es decir:

$$\rho(\mathbf{T}) = max\{|\lambda| \ : \ \lambda \ autovalor \ de \ \mathbf{T}\} \ < \ 1 \tag{2}$$

En este informe, trabajaremos con los métodos de *Jacobi* y *Gauss-Seidel* para la resolución de sistemas $\mathbf{A}x = b$. Estos descomponen a la matriz $\mathbf{A}$ de la siguiente forma:

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

donde $\mathbf{D}$ es la diagonal de $\mathbf{A}$, $\mathbf{L}$ contiene los elementos negados por debajo de la misma y $\mathbf{U}$ los elementos negados por encima. Luego, los esquemas para ambos métodos iterativos son los siguientes:

Método de Jacobi

$$x^{(k+1)} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})x^{(k)} + \mathbf{D}^{-1}b \tag{4}$$

Método de Gauss-Seidel

$$x^{(k+1)} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}x^{(k)} + (\mathbf{D} - \mathbf{L})^{-1}b \tag{5}$$

Se puede demostrar que, de converger, ambos métodos lo harán a una solución del sistema pedido. Requerimos adicionalmente, para su aplicación, que $\mathbf{A}$ sea una matriz sin elementos nulos en la diagonal. De lo contrario no se podrán calcular las inversas de $\mathbf{D}$ y $\mathbf{D} - \mathbf{L}$.

1.2. **Implementación.**

1.2.1. *Método de Jacobi.* Definamos la siguiente aridad para una implementación posible del Método de Jacobi:

$$jacobi: \ matriz_{n\times n}\ \mathbf{A} \ \times \ vector_n\ \mathrm{b} \ \times \ nat\ \mathrm{q} \ \times \ real\ \mathrm{t} \ \longrightarrow \ vector_n\ \mathrm{x}$$

donde n es un natural, $\mathbf{A}$ es una matriz con elementos distintos a cero en la diagonal, q es un número que indica la cantidad máxima de iteraciones a realizar y $t \geq 0$ representa la tolerancia mínima a partir de la que se considera la convergencia de una solución.

Si la matriz $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$, satisface que $\rho(\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})) < 1$, entonces el método de Jacobi convergerá a una solución del sistema $\mathbf{A}x = b$. Proponemos el siguiente algoritmo:

```
1  proc jacobi(in A: matriz<n, n>, in b: vector, in q: Nat, in t: Real) {
2
3      x := aleatorio(n)   // un vector aleatorio no nulo, ||x||_2 = 1
4
5      i := 0
6      while i < q and ||z||_2 ≥ t {
7          y := x
8          j := 0
9          while j < n {
10             s := 0
11             k := 0
12             while k < n {
13                 if k != j {
14                     s := s + A[j][k] * y[k]
15                 }
16                 k := k + 1
17             }
18             x[j] := (b[j] - s) / A[j][j]
19             j := j + 1
20         }
21         z := x - y
22         i := i + 1
23     }
24
25     return x
26 }
```

Algoritmo 1. Pseudocódigo para el Método de Jacobi.

Notamos que la complejidad del algoritmo es del orden de $\Theta(q * n^2)$ en el peor caso. En consecuencia, se debe precisar con cuidado la cantidad de iteraciones a realizar para que el factor q sea despreciable. Del mismo modo, una selección de t correcta, acorde al uso, puede resultar en mejoras considerables en la complejidad promedio.

1.2.2. *Método de Gauss-Seidel.* De manera similar, definimos la siguiente función que implementa el Método de Gauss-Seidel:

$$gauss\_seidel : \ matriz_{n\times n}\ \mathbf{A} \ \times \ vector_n\ \mathrm{b} \ \times \ nat\ \mathrm{q} \ \times \ real\ \mathrm{t} \ \longrightarrow \ vector_n\ \mathrm{x}$$

donde, nuevamente, n es un natural, $\mathbf{A}$ es una matriz con elementos distintos a cero en la diagonal, q es un número que indica la cantidad máxima de iteraciones a realizar y $t \geq 0$ representa la tolerancia mínima a partir de la que se considera la convergencia de una solución.

Si la matriz $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$, satisface que $\rho((\mathbf{D} - \mathbf{L})^{-1}(\mathbf{U})) < 1$, entonces el método de Gauss-Seidel convergerá a una solución del sistema $\mathbf{A}x = b$. Proponemos el siguiente algoritmo, similar al anterior:

```
1  proc gauss_seidel(in A: matriz<n, n>, in b: vector,
2                    in q: Nat, in t: Real) {
3
4      x := aleatorio(n)   // un vector aleatorio no nulo, ||x||_2 = 1
5
6      i := 0
7      while i < q and ||z||_2 ≥ t {
8          y := x
9          j := 0
10         while j < n {
11             s := 0
12             k := 0
13             while k < n {
14                 if k != j {
15                     s := s + A[j][k] * x[k]
16                 }
17                 k := k + 1
18             }
19             x[j] := (b[j] - s) / A[j][j]
20             j := j + 1
21         }
22         z := x - y
23         i := i + 1
24     }
25
26     return x
27 }
```

Algoritmo 2. Pseudocódigo para el Método de Gauss-Seidel.

Nuevamente, su complejidad temporal es $\Theta(q * n^2)$ en el peor caso.

Observamos que ambos algoritmos aplican ciertas heurísticas que pueden ayudar a reducir su complejidad temporal. Por un lado, se utiliza un vector inicial aleatorio con norma $L_2 = 1$. Esto nos permite evitar aquellas entradas que causan un comportamiento de peor caso de

manera determinística, a cuestas que una ejecución particular del algoritmo pueda resultar menos eficiente de manera aleatoria. Por otro, se considera la norma L_2 entre dos soluciones consecutivas, como medida de similitud, para definir un quiebre temprano en la iteración externa de los algoritmos en función del parámetro t.

1.2.3. *Eliminación Gaussiana.*

1.2.4. *Representación de matrices.*

2. ¿Por qué usamos $vector < SparseVector >$ como representación?

Nuestra primer implementación "naive" de la eliminación gaussiana usando Eigen fue la siguiente:

```
1  // A de tipo SparseMatrix
2  n = A.rows();
3  for (i = 0; i < n-1; ++i) {
4      mii = A.coeff(i, i);
5      for(j = i+1; j < n; ++j){
6          mij = A.coeff(j, i) / mii;
7          if (abs(mij) < epsilon) continue;
8
9          A.row(j) = (A.row(j) - A.row(i) * mij).pruned(1, epsilon);
10         b[j] = b[j] - b[i] * mij;
11     }
12 }
```

Algoritmo 3. Pseudocódigo de la primer implementación de la Eliminación Gaussiana

A pesar de ser un código sencillo, creíamos que al usar las funciones de Eigen podríamos obtener ventaja en velocidad pero la hipótesis fue claramente refutada por los 3 minutos 13 segundos que demoró en terminar el test de 15 segundos.

Observamos que la asignación de la línea 9 era una operación muy ineficiente. Esto tiene bastante sentido ya que SparseMatrix esta implementada usando CSR y por ende todos los elementos se encuentran contiguos en un único vector, entonces insertar y editar en el medio de este es un proceso costoso.

Concluimos de la observación anterior que en caso de tener un $vector < SparseVector >$, donde podamos hacer reemplazos de una fila por otra considerablemente rápido, además de seguir aprovechando las operaciones optimizadas de Eigen, sería una buena opción. Teniendo esto en mente, probamos la siguiente implementación:

```
1  // A de tipo vector<SparseVector>
2  n = A.size();
3  for (i = 0; i < n-1; ++i) {
4      double mii = A[i].coeff(i);
5      for (j = i+1; j < n; ++j) {
6          double mij = A[j].coeff(i) / mii;
7          if (abs(mij) < epsilon) continue;
8
9          A[j] = (A[j] - A[i] * mij).pruned(1, epsilon);
10         b[j] = b[j] - b[i] * mij;
11     }
12 }
```

Algoritmo 4. Pseudocódigo de la segunda implementación de la Eliminación Gaussiana

Definitivamente esta estrategia es superior ya que resuelve correctamente el test de 15 segundos en aproximadamente 2.5 segundos y el de 30 en 5.

Cabe destacar que para el tp1 hicimos una implementación muy similar, a la presentada, usando $vector < pair >$ como representación pero era bastante mas lenta que la versión final usando Eigen, probablemente se deba al manejo de memoria y la velocidad de los iteradores de la librería mencionada.

Un comentario necesario es que los tiempos mencionados fueron calculados usando $Epsilon = 10^{-5}$. Esto es relevante ya que modificar esta variable cambia considerablemente la velocidad del algoritmo. Por ejemplo con $Epsilon = 10^{-4}$ el algoritmo termina en menos de 1 segundo ambos tests, pero con $Epsilon = 10^{-6}$, demora 4 segundos en el de 15 y 10 en el de 30. No profundizaremos en como varía la velocidad respecto al $Epsilon$.

3. Evaluación de Convergencia

3.1. **Casos de test.** Notamos que las matrices asociadas a la resolución de *PageRank* son estocásticas en columna, por lo que ambos métodos siempre convergerán a una solución[1].

Metodología. Se evaluó el error absoluto $||x - pagerank(g,\ p)||_1$, donde x refiere a la solución verdadera, g refiere al grafo de entrada y p al *valor* p[2], para cada caso de test provisto, en función de la cantidad de iteraciones q a realizar en el rango $[1, 100]$.

Se repitió el experimento para dos implementaciones del algoritmo que difieren unicamente en el método de resolución del sistema lineal asociado a *PageRank*. En la primera se utilizó el *método de Jacobi* y en la segunda el *método de Gauss-Seidel*. Se controló la tolerancia ($t = 0$) para forzar a los algoritmos a iterar de manera exacta.

Resultados. Las figuras (1.) a (7.) muestran los los resultados del error absoluto L_1 para cada caso de test.

Notamos que la implementación de *PageRank* sobre el *método de Gauss-Seidel* tuvo una velocidad de convergencia mayor a la que logró la implementación sobre el *método de Jacobi*.

Además, para éste primer método, bastó $q < 60$ para lograr un error absoluto L_1 menor a 10^{-6} para todos los casos de test. En cambio, el *método de Jacobi* requirió más iteraciones — para alcanzar la misma meta— en un sólo caso: el test *15_segundos*. En particular, para este caso y el test *30_segundos*, notamos que la cantidad de iteraciones requeridas para converger no parece depender de manera estricta del tamaño de la matriz, pero sí parece existir una correlación.

Consideramos, también, que la meseta de error observado por debajo de 10^{-6} debe corresponder a un error de redondeo entre las soluciones esperadas y las computadas durante el experimento. Sin embargo, observamos que esta meceta no ocurre, en particular, en el test *completo*.

[1]Damos una demostración informal de este hecho: se puede demostrar que los métodos propuestos convergen para matrices estrictamente diagonal dominantes. En consecuencia, las matrices de iteración asociadas deben tener radio espectral < 1. Como las matrices estocásticas en columna son la traspuesta de las matrices estrictamente diagonal dominantes y los autovalores de una matriz equivalen a los de su traspuesta, entonces debe ser que las matrices de iteración asociadas a las matrices estocásticas en columna tienen radio espectral < 1. Como además su diagonal es no nula, entonces satisfacen las condiciones de convergencia para los métodos iterativos mencionados.

[2]Para una explicación en más detalle de PageRank, ver el *tp*1.

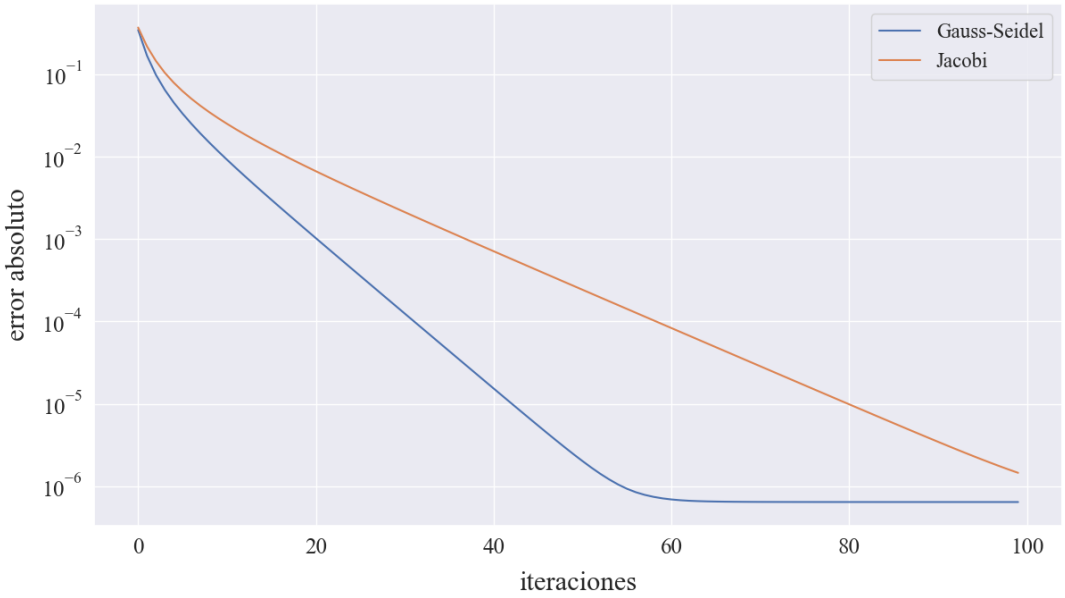

FIGURA 1. Error absoluto L_1 para el grafo del test *15_segundos*, con $n = 2000$ y $p = 0.9$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

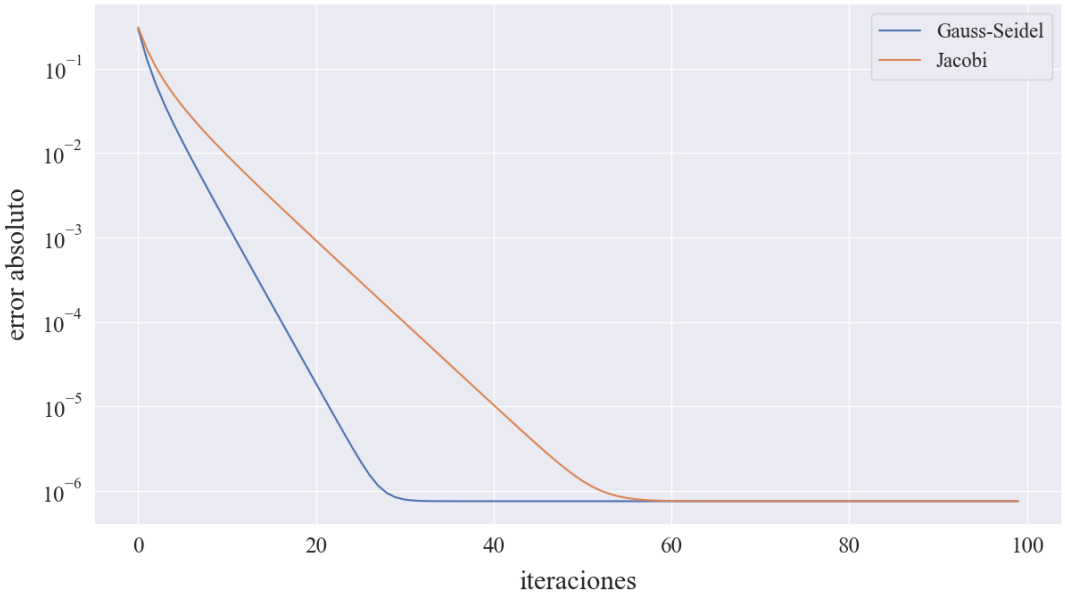

FIGURA 2. Error absoluto L_1 para el grafo del test *30_segundos*, con $n = 3000$ y $p = 0.8$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

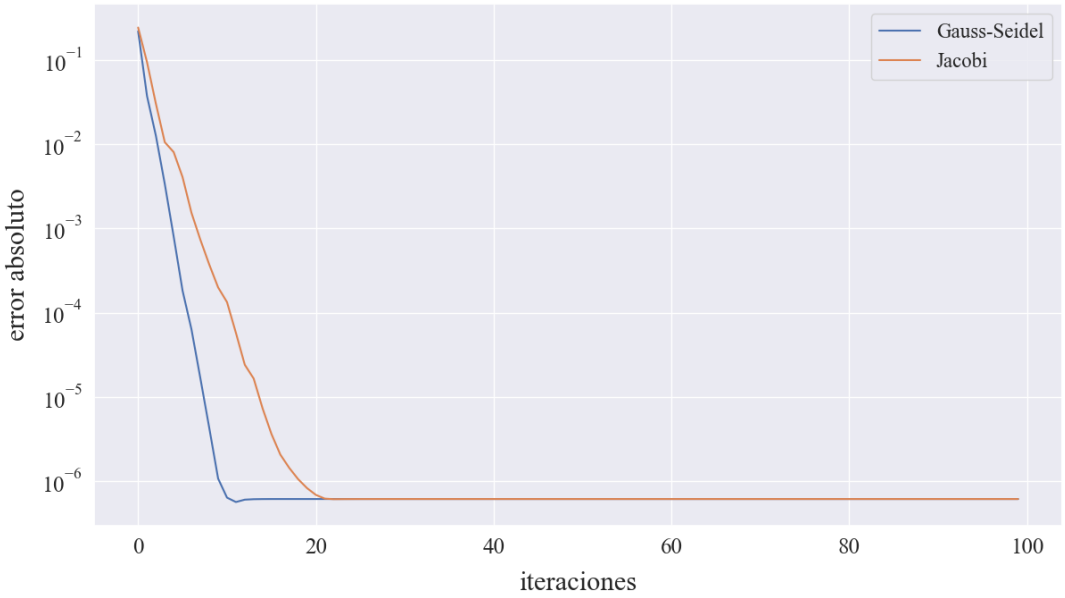

FIGURA 3. Error absoluto L_1 para el grafo del test *aleatorio_desordenado*, con $n = 5$ y $p = 0.76$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

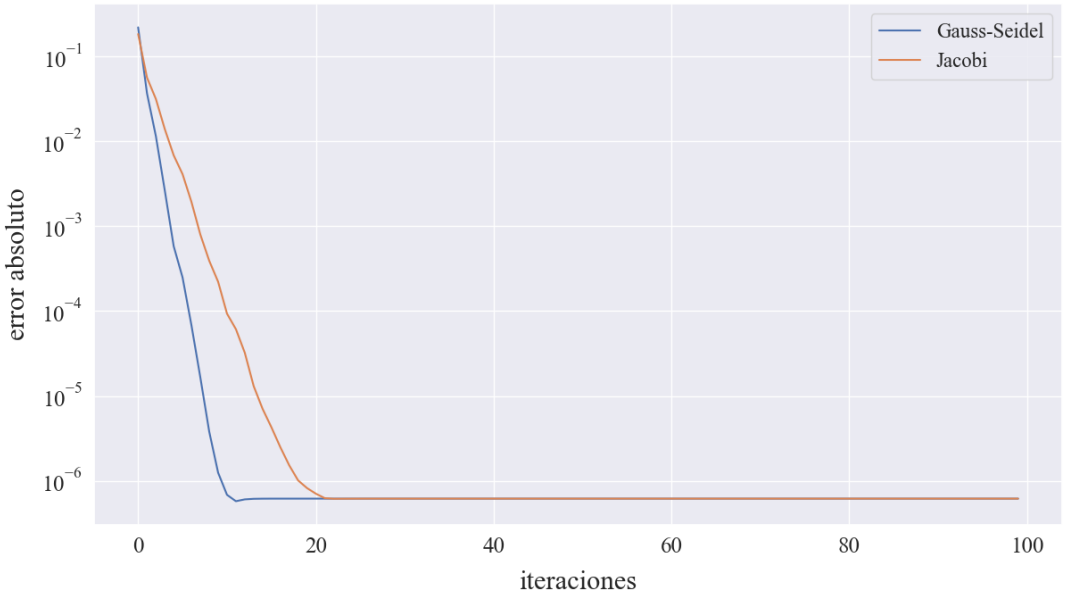

FIGURA 4. Error absoluto L_1 para el grafo del test *aleatorio*, con $n = 5$ y $p = 0.76$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

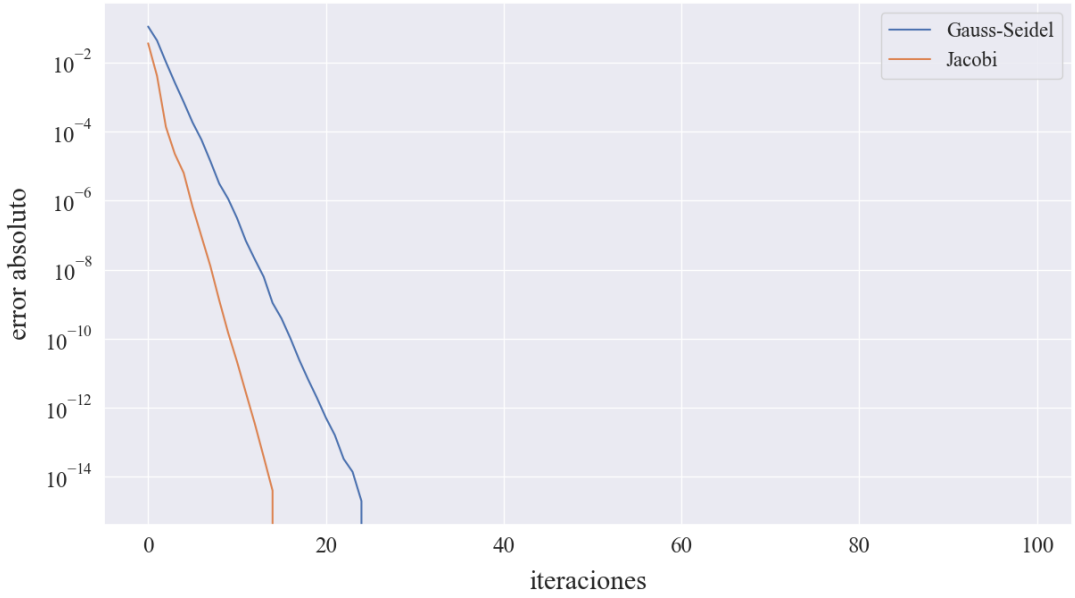

FIGURA 5. Error absoluto L_1 para el grafo del test *completo*, con $n = 5$ y $p = 0.5$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

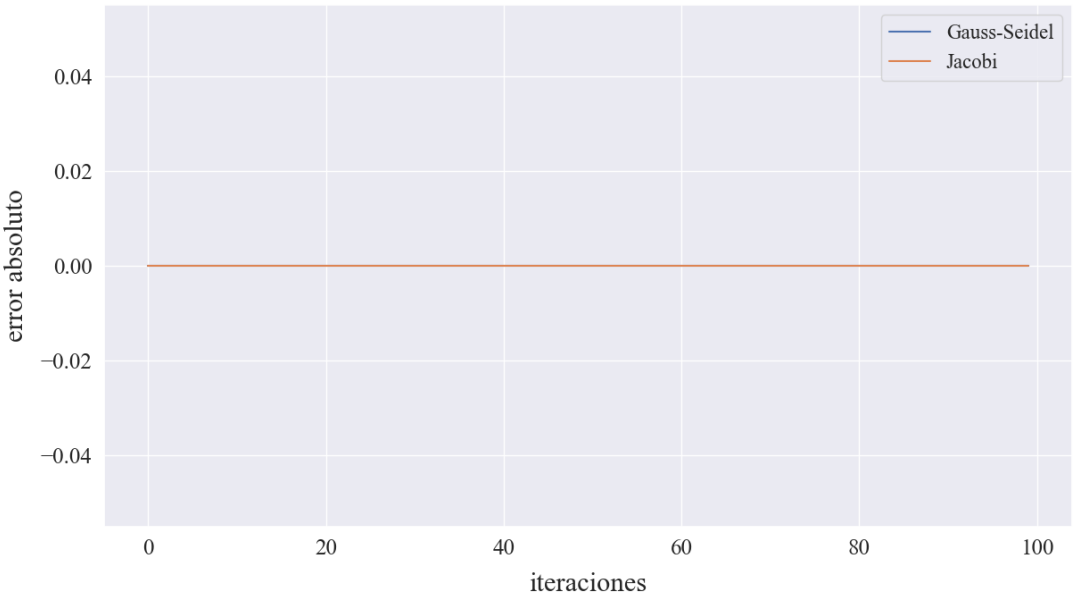

FIGURA 6. Error absoluto L_1 para el grafo del test *sin_links*, con $n = 5$ y $p = 0.64$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

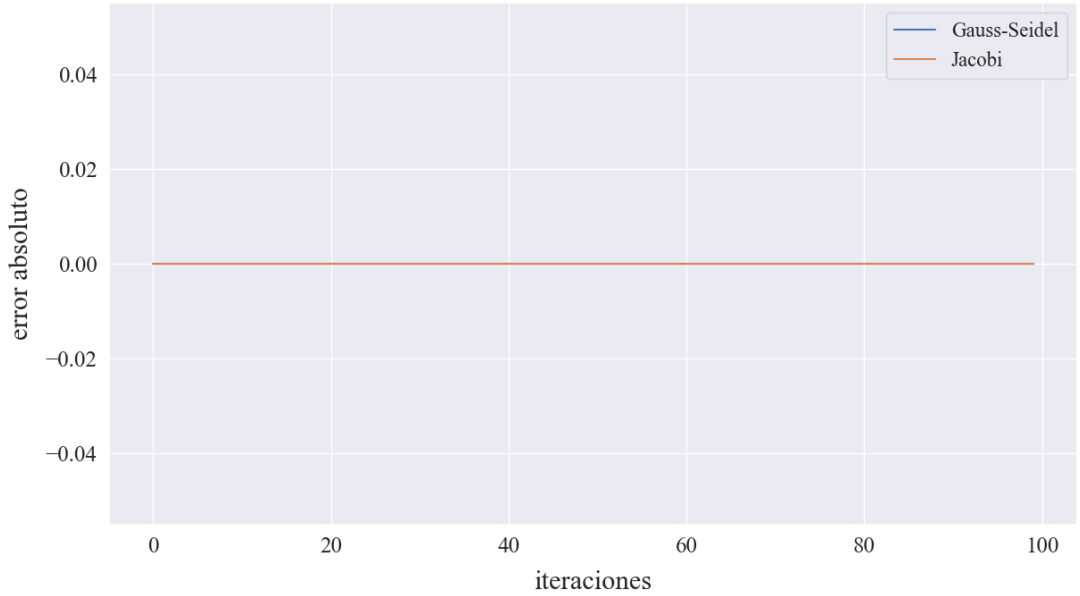

FIGURA 7. Error absoluto L_1 para el grafo del test *trivial*, con $n = 1$ y $p = 0.3$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de pagerank.

Procedemos a mostrar los resultados finales para el error absoluto con norma L_1 y L_∞ —para hacer una comparación por coordenadas—, tras las cien iteraciones:

	Jacobi		Gauss-Seidel	
test	L_1	L_∞	L_1	L_∞
15_segundos	1.45×10^{-6}	9.76×10^{-9}	6.4×10^{-7}	5×10^{-9}
30_segundos	7.54×10^{-7}	5×10^{-10}	7.54×10^{-7}	5×10^{-10}
aleatorio_desordenado	6.18×10^{-6}	2.52×10^{-7}	6.18×10^{-7}	2.52×10^{-7}
aleatorio	6.18×10^{-6}	2.52×10^{-7}	6.18×10^{-7}	2.52×10^{-7}
completo	0	0	0	0
sin_links	0	0	0	0
trivial	0	0	0	0

FIGURA 8. Error absoluto L_1 y L_∞ de los casos de test para las implementaciones de *PageRank* sobre los métodos de *Jacobi* y *Gauss-Seidel*.

4. Evaluación temporal

4.1. **Casos de test.** Para comenzar el análisis sobre el tiempo de ejecución que demora cada uno de los métodos, decidimos inicialmente estudiar como se comportan estos frente a los test de la cátedra. A partir de los resultados logramos obtener mayor claridad sobre como se desenvolvían los métodos en distintos casos. Para luego poder generar con una intuición orientada los tests más profundos.

Metodología. Inicialmente se definio un $epsilon = 10^{-6}$. Para cada uno de los test de la cátedra se realizó el cálculo de pagerank con el epsilon mencioniado para la eliminación gaussiana y se calculó $||x - res||_1$ donde x refiere a la verdadera solución y res al resultado que se obtuvo a partir de pagerank. Cabe destacar que este es un proceso determinístico.

Por otro lado, para el *método de Jacobi* y para el *método de Gauss-Seidel*, realizamos un binary-search donde en cada paso alteramos la tolerancia y calculamos el resultado. Editamos los límites de la tolerancia concorde el error calculado es mayor o menor que el error usando PageRank. Al finalizar este procedimiento obtuvimos tolerancias tales que, al calcular el error de comparar las respuestas que conceden *Jacobi* y *Gauss-Seidel* con la original, sea similar, en orden de magnitud, al error entre la respuesta que retorna PageRank y la original. De este modo podríamos realizar una comparación de tiempo de ejecución "justa", donde todos los métodos obtengan una respuesta con un error muy similar.

Luego para cada test, teniendo en cuenta las consideraciones previamente mencionadas, realizamos el cálculo 1 vez por método. Repetimos este paso 10 veces para atenuar las fluctuaciones de tiempo de ejecución que genera el computador y calculamos el promedio de cada método. Los resultados se pueden ver en la siguiente Figura.

Inserte Gráfico de barras Como se puede apreciar en el gráfico, el *método de Gauss-Seidel* y el *método de Jacobi*, son considerablemente más rápidos cuando el tamaño de la matriz es grande a pesar de que el resultado obtenido es de la misma calidad para los tres métodos en todos los casos de test.

4.2. **En función de la densidad del grafo de entrada.**

5. Conclusiones

6. Apéndice