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 aproximadamente 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

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, 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.

Los esquemas para ambos métodos 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 la 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 el otro lado, 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 y representación de matrices.* Por último, para poder realizar una comparación entre los *métodos iterativos* y los *métodos exactos*, propondremos una representación de matriz adecuada y un algoritmo eficiente de *eliminación gaussiana* con *sustitución inversa* para la resolución de *PageRank*.

Dadas las cualidades ralas de las matrices asociadas al problema, decidimos trabajar con la siguiente representación: un vector de vectores *sparse* de la librería *Eigen*, que se implementa sobre un esquema CRS —*compressed row storage*— para el indexado a memoria. En el apartado (1.2.4.) explicamos en más detalle esta decisión.

En tanto su implementación, proponemos los siguientes algoritmos para *eliminación gaussiana* y *sustitución inversa*, que aprovechan las herramientas que brinda *Eigen*:

```
1  proc eliminacion_gaussiana(inout A: vector<SparseVector>,
2                             inout b: vector, in ε: Real) {
3
4      n = filas(A)
5      for (i = 0; i < n-1; ++i) {
6
7          mii = A[i].coeff(i)
8          for (j = i+1; j < n; ++j) {
9
10             mij = A[j].coeff(i) / mii
11             if (|mij| < ε) {
12                 continue
13             }
14             A[j] = (A[j] - A[i] * mij).pruned(1, ε)
15             b[j] = b[j] - b[i] * mij
16         }
17     }
18 }
```

Algoritmo 3. Pseudocódigo para la *Eliminación Gaussiana*.

donde *filas*($\mathbf{A}$) refiere a una función que retorna la cantidad de filas en la matriz $\mathbf{A}$, *v.coeff(i)* refiere a un acceso al i-ésimo elemento del vector v y *v.pruned(α, ε)* es una función que reemplaza los elementos explícitos en la memoria del vector v con valor absoluto menor a $\alpha \cdot \varepsilon$ con un cero implícito.

Notamos que la utilización de la función *pruned* permite mantener una matriz esparsa a lo largo del proceso de eliminación a cuestas de una pérdida de precisión en los resultados. Por ello, una buena selección de ε es fundamental para el buen funcionamiento del algoritmo.

```
proc sustitucion_inversa(in A: vector<SparseVector>,
                         in b: vector) {

    x = b   // vector solucion

    for (i = filas(A) - 1; i ≥ 0; --i) {

        parcial = 0
        for (iterador(A[i]); it; ++it) {
            if (pos(it) < i + 1) {
                continue
            }
            parcial += it.coeff() * x[pos(it)]
        }
        x[i] = (x[i] - parcial) / A[i].coeff(i)
    }

    return x
}
```

ALGORITMO 4. Pseudocódigo para *Sustitución Inversa*.

El algoritmo (4.), por su parte, utiliza las funciones *iterador(v)*, que retorna un iterador sobre los elementos no nulos del vector v, y *pos(it)*, que retorna la posición —contando ceros— a la que está apuntando el iterador *it*.

1.2.4. *¿Por qué usamos vector<SparseVector> como representación?* Nuestra primera implementación 'naive' de la *Eliminación Gaussiana* con Eigen fue la siguiente:

```
void eliminacion_gaussiana(SparseMatrix<double> &A,
                           vector<double> &b, double ε) {

    n = A.rows();
    for (i = 0; i < n-1; ++i) {

        mii = A.coeff(i, i);

        for (j = i+1; j < n; ++j) {
            mij = A.coeff(j, i) / mii;
            if (abs(mij) < ε) continue;

            A.row(j) = (A.row(j) - A.row(i) * mij).pruned(1, ε);
            b[j] = b[j] - b[i] * mij;
        }
    }
}
```

ALGORITMO 5. Código C++ de la primera implementación de *Eliminación Gaussiana*.

A pesar de ser un código sencillo, creíamos que al usar las funciones de la librería y la representación *SparseMatrix* podríamos obtener una ventaja en velocidad. Pero la hipótesis fue claramente refutada con los tiempos que obtuvimos: el test *15_segundos*, por su cuenta, demoró tres minutos y trece segundos en ejecutar en una de nuestras máquinas.

Observamos que la asignación de la línea 13 es una operación muy ineficiente. Esto tiene bastante sentido, ya que *SparseMatrix* está implementada usando *CRS* y por ende todos los elementos se encuentran contiguos en un único vector. Insertar y editar un índice en el medio de éste 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 que provee *Eigen*, podríamos acelerar considerablemente el algoritmo. Teniendo esto en mente, probamos la implementación detallada en el apartado anterior.

Esta estrategia resultó considerablemente mejor, ya que —en nuestras máquinas— resuelve correctamente el test *15_segundos* en aproximadamente 2.5 segundos y el test *30_segundos* en 5[1].

[1]Un comentario necesario es que los tiempos mencionados fueron calculados usando $\varepsilon = 10^{-5}$. Esto es relevante ya que modificar esta variable cambia considerablemente la velocidad del algoritmo. Por ejemplo, con $\varepsilon = 10^{-4}$ el algoritmo termina ambos tests en menos de un segundo, pero con $\varepsilon = 10^{-6}$, demora 3 segundos en el de 15 y 8 en el de 30. No profundizaremos en cómo varía la velocidad respecto a ε.

2. Evaluación de Convergencia

2.1. **Casos de test.** Evaluamos[2] el error absoluto en norma L_1 de los resultados de nuestra implementación de *PageRank* sobre el *método de Jacobi* y el *método de Gauss-Seidel* para los casos de test provistos por la cátedra[3], en función de la cantidad de iteraciones realizadas.

Notamos que las matrices asociadas a la resolución de *PageRank* son estocásticas en columna, por lo que ambos métodos siempre convergerán a una solución[4].

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[5] utilizado, 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 únicamente 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 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 este 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 en un sólo caso más iteraciones para alcanzar la misma meta: 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 meseta no ocurre, en particular, para el test *completo*.

Dadas las limitaciones de esta evaluación —por la escasa cantidad de casos de test—, poco se puede decir respecto al comportamiento general de ambos métodos. Como futura hipótesis a investigar, un estudio más detallado podría evaluar la velocidad de convergencia en función de otros parámetros, como el tamaño de la matriz, la densidad o el valor p utilizado.

---

[2]El script asociado se puede encontrar en *./experimentos/convergencia.py*, los archivos resultado en *./experimentos/resultados/convergencia-iterativos*

[3]Los mismos se pueden encontrar en *./catedra/tests-pagerank*

[4]se puede demostrar que los métodos propuestos convergen para matrices estrictamente diagonal dominantes, las matrices estocásticas en columna son una variante de este tipo.

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

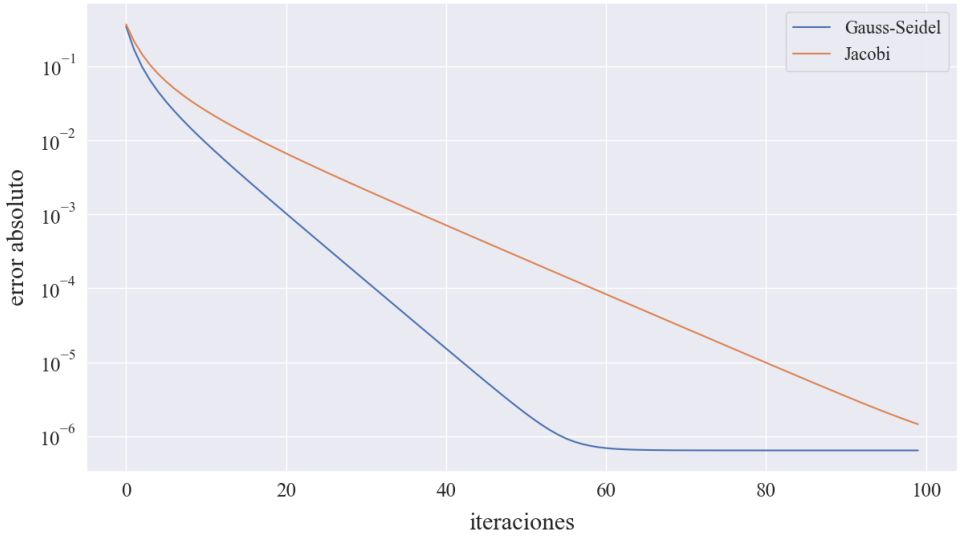

FIGURA 1. Error absoluto L_1 para el grafo del test *15_segundos*, con $n = 2000$ y $p = 0.9$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank* (escala logarítmica).

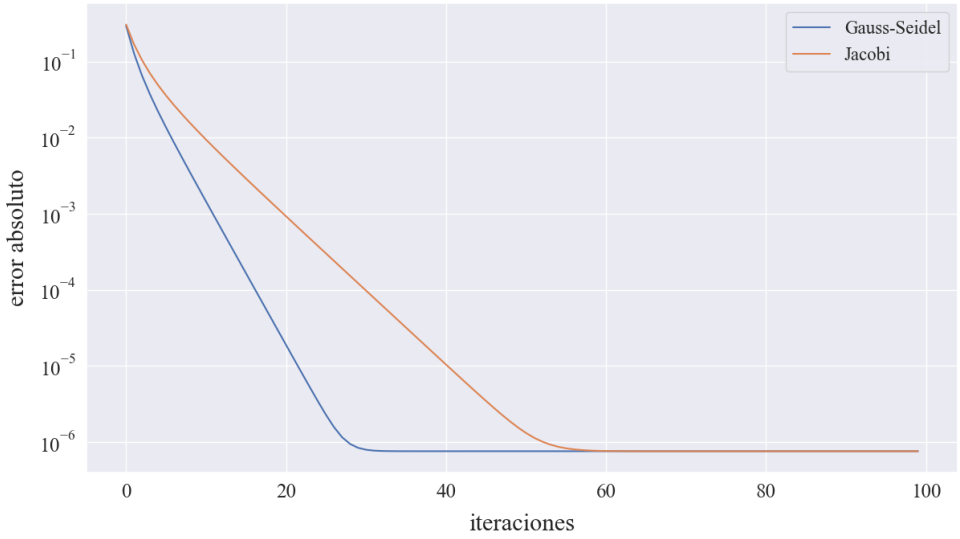

FIGURA 2. Error absoluto L_1 para el grafo del test *30_segundos*, con $n = 3000$ y $p = 0.8$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank* (escala logarítmica).

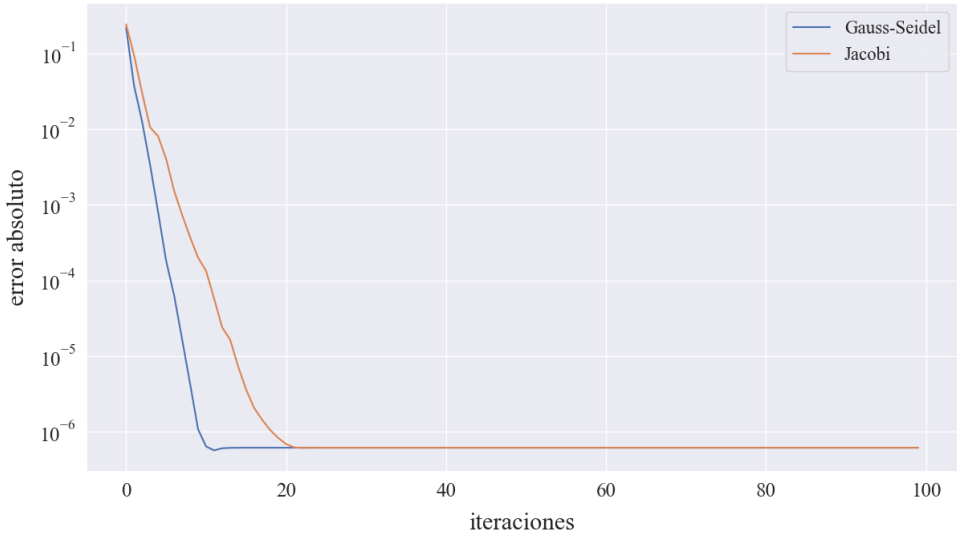

FIGURA 3. Error absoluto L_1 para el grafo del test *desordenado*, con $n = 5$ y $p = 0.76$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank* (escala logarítmica).

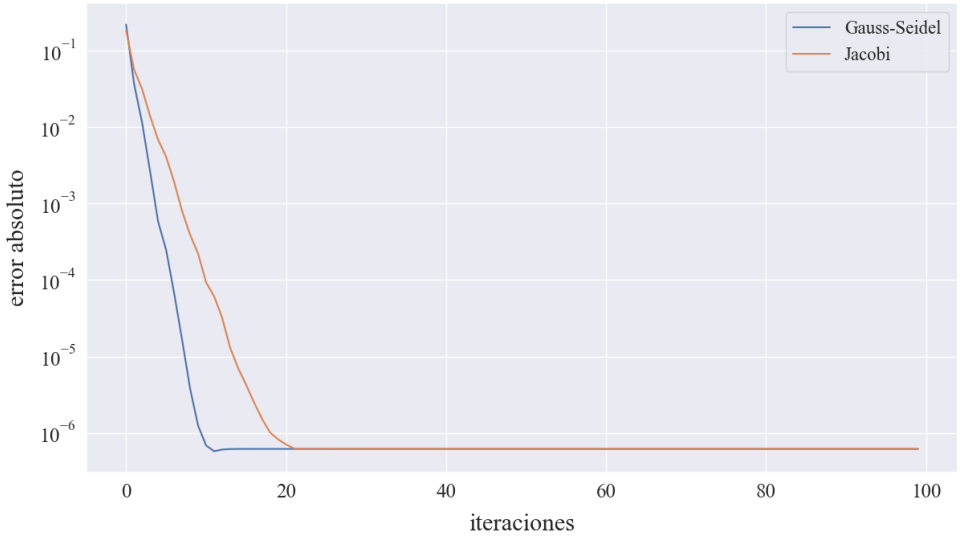

FIGURA 4. Error absoluto L_1 para el grafo del test *aleatorio*, con $n = 5$ y $p = 0.76$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank* (escala logarítmica).

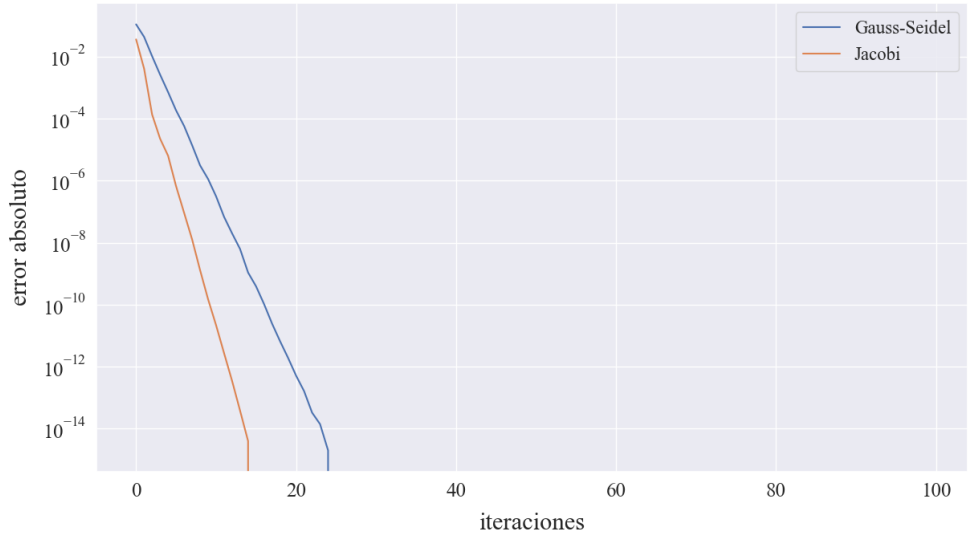

FIGURA 5. Error absoluto L_1 para el grafo del test *completo*, con $n = 5$ y $p = 0.5$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank* (escala logarítmica).

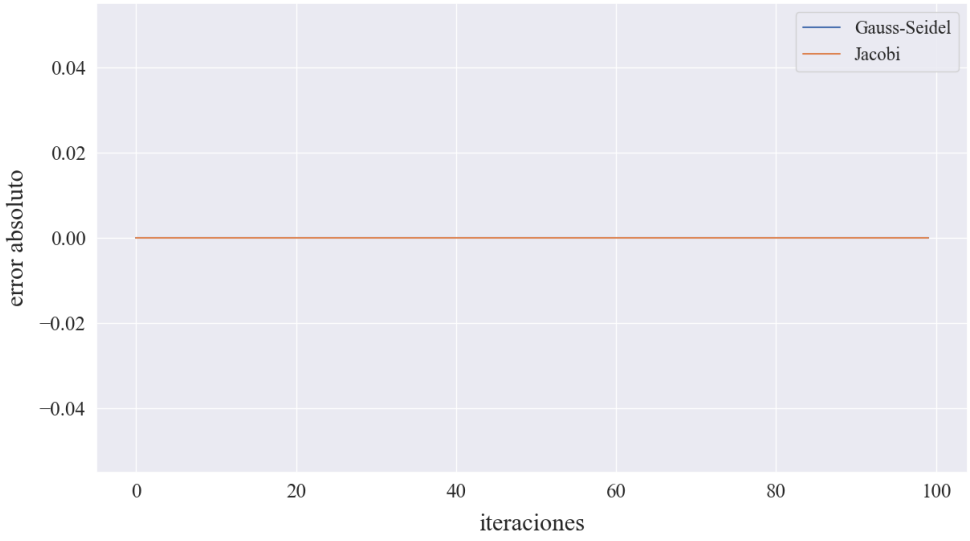

FIGURA 6. Error absoluto L_1 para el grafo del test *sin_links*, con $n = 5$ y $p = 0.64$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank*.

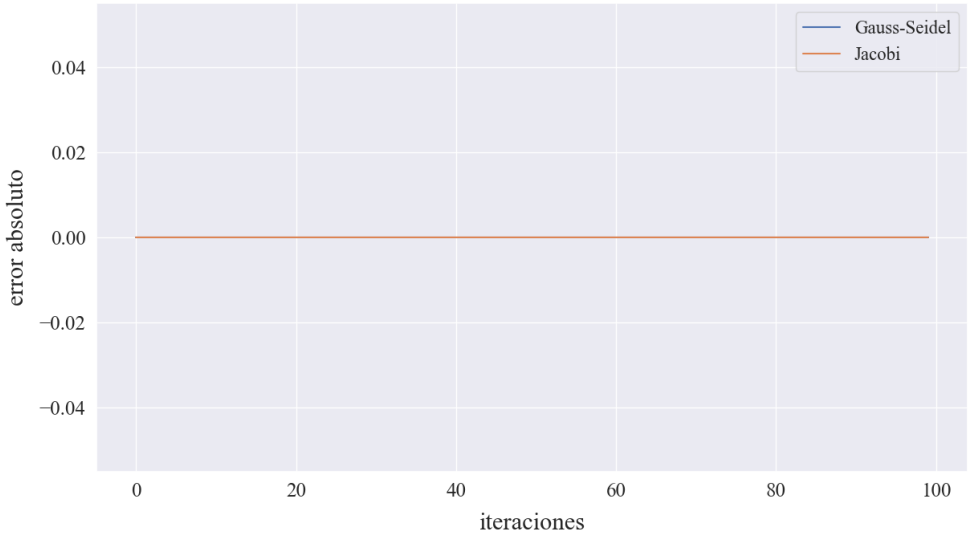

FIGURA 7. Error absoluto L_1 para el grafo del test *trivial*, con $n = 1$ y $p = 0.3$, en función de la cantidad de iteraciones realizadas, para ambas implementaciones de *PageRank*.

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*.

3. Evaluación temporal

3.1. **Casos de test.** Para comenzar el análisis sobre el tiempo de ejecución de cada uno de los métodos, decidimos inicialmente estudiar cómo se comportan frente a los test de la cátedra.

Metodología. Se realizó el cálculo de *PageRank* sobre la implementación que utiliza la *eliminación gaussiana*, con $\varepsilon = 10^{-6}$, y se calculó $||x - pagerank(g, p)||_1$ donde x refiere a la solución verdadera provista por cada uno de los tests. Cabe destacar que este es un proceso determinístico.

A partir de este resultado realizamos una búsqueda de la tolerancia ideal, para el *método de Jacobi* y el *método de Gauss-Seidel*, que nos permitiera calcular *PageRank* con un error similar al obtenido anteriormente. Para esto, se realizó —para ambos métodos— un binary-search donde en cada paso alteramos la tolerancia t y calculamos el resultado de *PageRank*. Editamos los límites de la tolerancia acorde a si el error calculado fue mayor o menor al error que se cometió con la *eliminación gaussiana*. Repetimos el proceso hasta que la diferencia entre ambas soluciones fue menor a $10^{-2} \times \varepsilon$ en norma L_1.

Dados estos parámeteros, se procedió a calcular *PageRank* para cada test, sobre cada implementación y medir el tiempo de ejecución de la etapa de resolución del sistema lineal asociado. Repetimos este paso diez veces para atenuar las fluctuaciones de tiempo de ejecución[6].

Resultados. Procedemos a detallar los resultados en la Figura (9.).

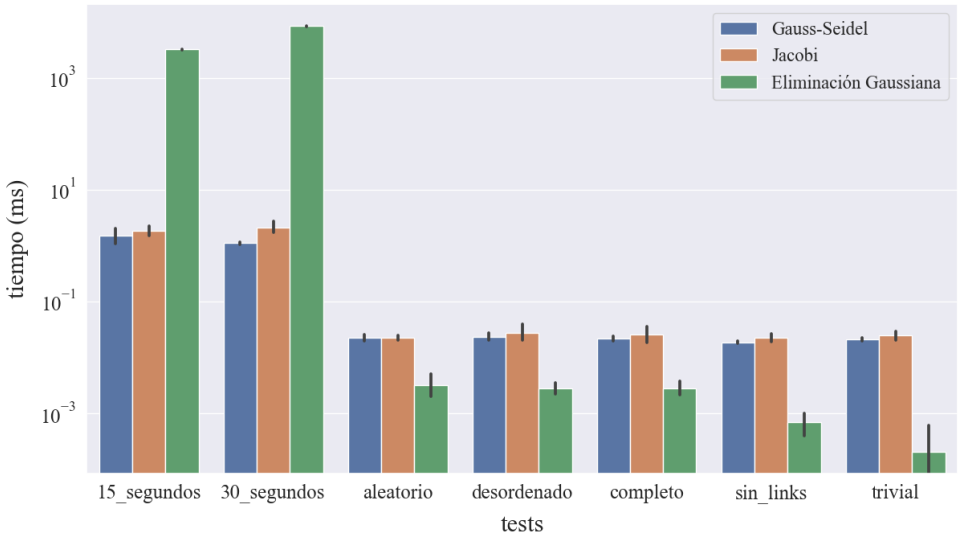

Figura 9. Tiempo de ejecución —en milisegundos— de las distintas implementaciones de *PageRank* para cada uno de los tests de la cátedra (escala logarítmica).

[6]Para evitar disparidades sobre los resultados de este experimento, y el subsiguiente, aclaramos que estos se ejecutaron en una misma computadora.

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. Para el test *30_segundos*, estos métodos tardaron alrededor de dos milisegundos, mientras que la *eliminación gaussiana* tardó mas de ocho segundos. Sin embargo, cuando el tamaño de la matriz es chico, notamos que la *eliminación gaussiana* es más rápida.

Por su parte, los métodos iterativos tienen una velocidad muy similar. El *método de Gauss-Seidel* fue apenas más rápido que el *método de Jacobi* en todos los casos evaluados.

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

4. Conclusiones

5. Apéndice