Métodos Numéricos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 3

Grupo 14

Integrante	LU	Correo electrónico
Dabbah, Julián	015/09	djulius@gmail.com
Dahlquist, Ariel	383/10	ariel.dahlquist@gmail.com
Garrone, Javier	151/10	javier3653@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

El presente T.P. se preocupa por el algoritmo PageRank para medir la importancia de las páginas web resultantes de una búsqueda, en particular, en el enfoque provisto por el trabajo *Extrapolation methods for accelerating pagerank computations, Kamvar et al.*[1] Implementaremos dos de los algoritmos propuestos en esta publicación, siendo el segundo una optimización del primero, que resuelve el caso más general.

1.1. Algoritmo PageRank

Para ponderar la importancia de una páginas web, este algoritmo propone utilizar como medida la cantidad de enlaces que apuntan a esa página. Además, para agregar significación, cada enlace entrante a la página debe reflejar la importancia de la página desde la cual sale. Formalmente, el rank de la página k: x_k se define como, si $L = \{1 \dots n\}$ es el conjunto de páginas que apuntan a k y llamando n_j a la cantidad de links salientes de la página j:

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j}$$

Traduciendo esto matricialmente, podemos pensar en la matriz A donde $(A)_{i,j} = \frac{1}{n_j}$ si la página j apunta a i, $(A)_{i,j} = 0$ si no. Luego, el vector de ranks $X = (x_k)$ verifica $AX = X$, con lo cual, es un autovector de autovalor 1. Luego, en principio, para obtener los ranks bastaría con encontrar uno de éstos. Sin embargo, el modelo no representa adecuadamente el caso en que una página no apunte hacia otras: en ese caso su rank sería 0 y quedaría excluída. Este fenómeno aparece en la bibliografía como el problema de *dangling nodes*. También podemos justificar el modelo pensando en que la matriz representa una cadena de Markov, donde $(A)_{i,j}$ indica la probabilidad de, estando en la página (estado) j, pasar a la página (estado) i. Luego, podemos agregar las transiciones desde los *dangling nodes* uniformes como $\frac{1}{n}$, donde n es la cantidad total de páginas consideradas (es decir, en esta situación, se abandona la página pasando a cualquier otra con igual probabilidad). Sin embargo, esta solución trae problemas algebraicos, ya que no se puede asegurar que exista un autovector de autovalor 1 ni que el autoespacio asociado tenga dimensión 1. Si llamamos P a la matriz resultante, podemos solucionar el problema considerando una nueva matriz M que se obtiene como $M = cP + (1 - c)E$, para cualquier $0 < c < 1$, donde $E_{i,j} = \frac{1}{n}$. Así, M tiene efectivamente un autovector de autovalor 1 con multiplicidad 1. Explicaciones más detalladas incluyendo demostraciones de algunas de las propiedades enunciadas se pueden encontrar en *Kurt Bryan and Tanya Leise. The linear algebra behind google. SIAM Review, 48(3):569–581, 2006.*

1.2. Búsqueda de Autovalores

Como vimos, para encontrar el rank de las páginas es necesario encontrar un autovector de autovalor 1 de la matriz propuesta. Además, se puede asegurar que este autovalor siempre existe y es el autovalor principal, con lo cual desde el punto de vista numérico, cabe atacar este problema mediante el método de la potencia, es decir, calcular sucesivas iteraciones de $x^{(k+1)} = Ax^{(k)}$, a partir de algún x_0, en principio cualquiera. Sin embargo, dadas las dimensiones en las que se estaría trabajando, no cabe una implementación trivial. Para eso, se proponen las siguientes dos variantes, que simplemente comentamos a continuación. El desarrollo completo se puede encontrar en la publicación citada.

1.2.1. Multiplicación Esparsa

Si bien la matriz original sobre la que trabajaría el algoritmo es típicamente esparsa (ya que la cantidad de enlaces entre dos páginas dadas es significativamente menor que la cantidad de páginas indexadas), las modificaciones para solucionar el problema de los *dangling nodes*, anulan esta propiedad. En esta situación, las sucesivas multiplicaciones de la matriz por el vector, resultarían muy costosas, tanto en tiempo como en espacio. Para esto, los autores proponen realizar esta modificación en los siguientes pasos, que como veremos, resulta equivalente a realizar la multiplicación directamente.

Por un lado, únicamente se almacena la matriz A como la definimos en primer momento, es decir, la que contiene la información sobre la distribución real de las páginas en la web y es altamente esparsa (lo cual permite ahorrar espacio de almacenamiento), y se modifica el vector resultante de realizar el producto. En concreto, se propone:

Algoritmo 1, *Kramver et al.*

$y = cAx$
$\omega = ||x||_1 - ||y||_1$
Devolver $y + \omega[\frac{1}{n}]_{\in \mathbb{R}^n}$

[1] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In Proceedings of the 12th international conference on World Wide Web, WWW '03, pages 261–270, New York, NY, USA, 2003. ACM.

Es decir, se realiza la multiplicación aprovechando la matriz esparsa, y la corrección se agrega al final, en proporción a la diferencia de las normas entre el vector original y el resultado de la transformación. Dado que esta corrección altera a todos los coeficientes de toda la matriz original por igual, puede evitarse a la hora de realizar el primer producto y aplicarse al final, tomando como métrica la relación entre el vector original y el transformado.

1.2.2. Extrapolación Cuadrática

Tal como lo explican los autores, la velocidad de convergencia del método de la potencia depende de la relación entre el autovalor principal y el subsiguiente. Luego, se propone agregar una modificación al algoritmo típico del método de la potencia para aprovechar las iteraciones anteriores utilizándolas para estimar una mejor aproximación del autovector buscado. Básicamente, se parte de la suposición de que cada iteración es combinación lineal de tres autovectores asociados a los tres únicos autovalores y se deducen relaciones entre ellos a partir de saber que para cualquier autovalor λ se tiene $\chi_A(\lambda) = 0$ y $\chi_A(A) = 0$ (Teorema de Hamilton-Cayley). Dado que $x^{(k+t)} = A^t x^{(k)}$, $(\chi_A(A)) \cdot x^{(k)} = 0$ relaciona iteraciones sucesivas del método mediante los coeficientes del polinomio y las potencias a las que aparece elevada la matriz. Continuando con la suposición de la existencia de sólo tres autovectores, se pueden deducir coeficientes que anulen la presencia en la combinación lineal de los autovectores no asociados al autovalor principal. Sin embargo, dado que esto no es necesariamente cierto, se utilizan estas relaciones para construir un sistema sobredimensionado para ajustar mediante cuadrados mínimos y apartir de los coeficientes obtenidos, construir el próximo vector para la iteración del método. Debido a que la suposición sobre la cantidad de autovectores no necesariamente es cierta, el vector obtenido no es la solución exacta (como se demuestra, en base a la suposición, en la publicación), pero de todas maneras, representa una buena aproximación, y, sobe todo, reduce efectivamente los coeficientes con los que aparecen en la combinación lineal los autovectores sucesivos, acelerando la convergencia del método.

2. Desarrollo

2.1. Representación de la matriz esparsa

Dado que, como dijimos, la matriz sobre la que trabajaremos es altamente esparsa, utilizamos una estructura de datos que aproveche este aspecto para ahorrar tanto espacio en memoria como tiempo de cómputo. Para esto, representamos la matriz como un arreglo de columnas, cada una de éstas siendo una lista enlazada de pares $< i, a_{i,j} >$, $a_{i,j} \neq 0$. Desde el punto de vista espacial, esto nos permite almacenar únicamente las entradas no nulas de la matriz. Esta representación nos permite realizar varias de las operaciones que se requieren para construir la matriz particular sobre la que trabaja PageRank de manera eficiente al recorrer por columnas, y también nos permite ejecutar eficientemente la operación de multiplicar por un vector a derecha, que es el paso central del método de la potencia que utilizamos para calcular el autovector buscado, de la siguiente manera. Siendo:

$$Av = b \Leftrightarrow b_i = (Av)_i = \sum_{k=1}^{n} a_{i,k} v_k$$

se puede ver que b_i resulta ser la acumulación de la suma de todos los $a_{i,k}v_k$. Dado que al multiplicar un vector por una matriz se utilizan todas las filas una única vez (es decir i adopta todos los valores entre 1 y n), y por ende, todos los elementos de la matriz (una única vez), podemos computar lo anterior como:

$$b = 0;\ \text{Para } a_{i,j}, 1 \leq i \leq n, 1 \leq j \leq n : b_i := b_i + a_{i,j} v_j$$

Dado que la suma es conmutativa, el orden en el que iteremos los $a_{i,j}$ no altera el resultado, con lo cual, podemos iterar "hacia abajo" por las columnas (aprovechando la estructura de lista, donde cada avance tiene costo constante), y esto repetirlo en todas. De esta manera, cubrimos todos los elementos de la matriz una única vez y acumulamos los resultados parciales en la coordenada correspondiente del vector resultado. Además, dado que los elementos nulos de la matriz no están representados en las listas, evitamos computar productos nulos y su acumulación. Luego, el costo de multiplicar esta matriz por un vector cualquiera depende únicamente (y de manera lineal) de la cantidad de elementos no nulos de ésta.

2.2. Método de la Potencia

Como ya dijimos, el método de la potencia es un método iterativo muy simple que converge hacia el autovalor de mayor módulo de la matriz (en caso de que exista uno único). Nuestra implementación es bastante directa, utilizando las estructuras que describimos antes. Consideramos como criterio de parada que la diferencia (en módulo) de la norma 1 de dos iteraciones sucesivas sea menor que $\varepsilon = 10^{-8}$. Consideramos que con este valor es suficiente, ya que utilizaremos el resultado únicamente para ordenar las páginas web entre sí, con lo cual, suponemos que agregar precisión en los decimales menos significativos no debería producir alteración en el orden de las páginas, que estaría dominado por las primeras cifras. Además, para evitar problemas numéricos de representación, normalizamos el vector luego de cada iteración, para controlar el rango de valores de las coordenadas.

2.3. Extrapolación Cuadrática y Cuadrados Mínimos

La implementación del algoritmo de Extrapolación Cuadrática es directa del presentado en la publicación citada. Lo único que debimos agregar fue la manera de resolver el sistema de cuadrados mínimos que genera, utilizando matrices ortogonales. Dado que la aproximación por cuadrados mínimos consiste en encontrar un x tal que minimice $||Ax - b||_2$ y que multiplicar por matrices ortogonales no altera la norma, resulta equivalente buscar un x que minimice $||QAx - Qb||_2$. Dado que la matriz A que construimos tiene dimensión $n \times 2$ (con $n > 2$) y que podemos conseguir Q tal que QA resulte triangular superior, minimizar $||QAx - Qb||_2$ es equivalente a resolver el sistema de 2×2 que resulta de tomar las dos primeras filas de QA y de Qb, lo cual se realiza muy fácilmente. Para conseguir esta tal Q adaptamos el método de Householder para construir la factorización QR de una matriz, de la siguiente manera.

El método de Householder genera matrices ortogonales Q_j que, multiplicadas a la izquierda de A, hacen que el producto sea triangular superior. Dado que hay que construir una por cada columna de A, en nuestro caso construimos un sistema equivalente: $Q_2Q_1Ax = Q_2Q_1b$. Dado que Q_2Q_1 es una matriz ortogonal, el sistema resulta equivalente, tanto en términos de cuadrados mínimos (como dijimos antes), como en términos de la solución lineal ($Ax = b \Leftrightarrow MAx = Mb$ si M es inversible).

Sin embargo, dado que las dimensiones de las matrices con las que debe lidiar el algoritmo de PageRank suelen ser muy grandes, construir estas matrices explícitamente no es una opción. En vez de esto, aplicamos las transformaciones que propone el método directamente sobre las columnas de la matriz y sobre el término independiente (dado que sabemos que son equivalentes a la multiplicación por matrices ortogonales). En particular: sabemos que $Q_1 = Id - 2u_1u_1^t$, donde $u_1 = a_1 - ||a_1||_2 e_1$ normalizado, siendo a_1 la primera columna de A y e_1 el primer vector de la base canónica de $\mathbb{R}^n$, y que el efecto de hacer Q_1A es anular toda la primera columna, excepto su primer elemento, que se convierte en la norma de la columna antes de la modificación, y convertir a_2 en Q_1a_2. Además, dado que la misma transformación la debemos aplicar sobre el término independiente, convertimos b en Q_1b. La operación sobre a_1 no es necesario computarla, simplemente corregimos los valores de la matriz según lo que dijimos antes. Los otros productos sí debemos calcularlos, aunque podemos observar lo siguiente: dado que $Q_1 = Id - 2u_1u_1^t$, al multiplicar a derecha por un vector v obtenemos $Q_1v = v - 2u_1u_1^tv = v - 2(u_1^tv)u_1$, pues $u_1^tv \in \mathbb{R}$. Luego, tanto para el caso de a_2 como para b, computar esto no resulta costoso: basta con calcular el producto interno con u_1 (lo que tiene costo lineal en la dimensión de los vectores) y luego restar del vector original que estamos considerando: $v_i := v_i - 2 < u_1, v > (u_1)_i$. Teniendo esto calculado, procedemos de manera análoga para la segunda columna, recordando que no debemos considerar la primera fila tanto de Q_1A como de Q_1b. Finalmente, tenemos un sistema triangular superior, equivalente al original, que podemos resolver mediante backward-substitution en dos pasos.

2.4. Construcción de la matriz de PageRank

Para construir la matriz sobre la cual trabajaremos, primero leemos una a una las líneas de la entrada, colocando un 1 en la posición indicada por cada una. Luego, dado que la matriz debe ser estocástica por columnas, recorriendo éstas una a una dividimos cada valor por la suma de todos sus elementos (ésta es otra de las ventajas de tener la matriz representada por columnas). Como dijimos en la introducción, no realizamos más modificaciones sobre la matriz; los valores de corrección para considerar *dangling-nodes* no los representamos directamente, sino que son consecuencia del manejo de algebraico particular de los vectores, según lo explica el algoritmo que estamos considerando.

3. Resultados

El primer experimento se realizó utilizando el Berkeley-Stanford web graph . La cantidad de nodos/páginas web es de 685230 y la cantidad de ejes/links es de 7600595. Las pruebas se corrieron en una máquina Intel Core i5-3550 CPU @ 3.30GHz x 4 con 8 GB de ram.

3.1. Evolución del error

El experimento fue realizado variando la constante c, la cual determina la importancia proporcional que se quiere entre la probabilidad de pasar a una página desde un link (c más alto) contra la de escribir la url a mano (c más bajo, comportamiento aleatorio).

Como criterio de parada, comparamos la diferencia en norma uno del autovector que se obtiene en cada iteración contra el de la iteración anterior, la cual debe ser menor que $\varepsilon = 10^{-8}$.

A continuación mostramos, para distintos c, la cantidad de iteraciones que necesita el método de la potencia para alcanzar dicho ε, sin utilizar extrapolación cuadrática (QE) primero, e incluyéndola después:

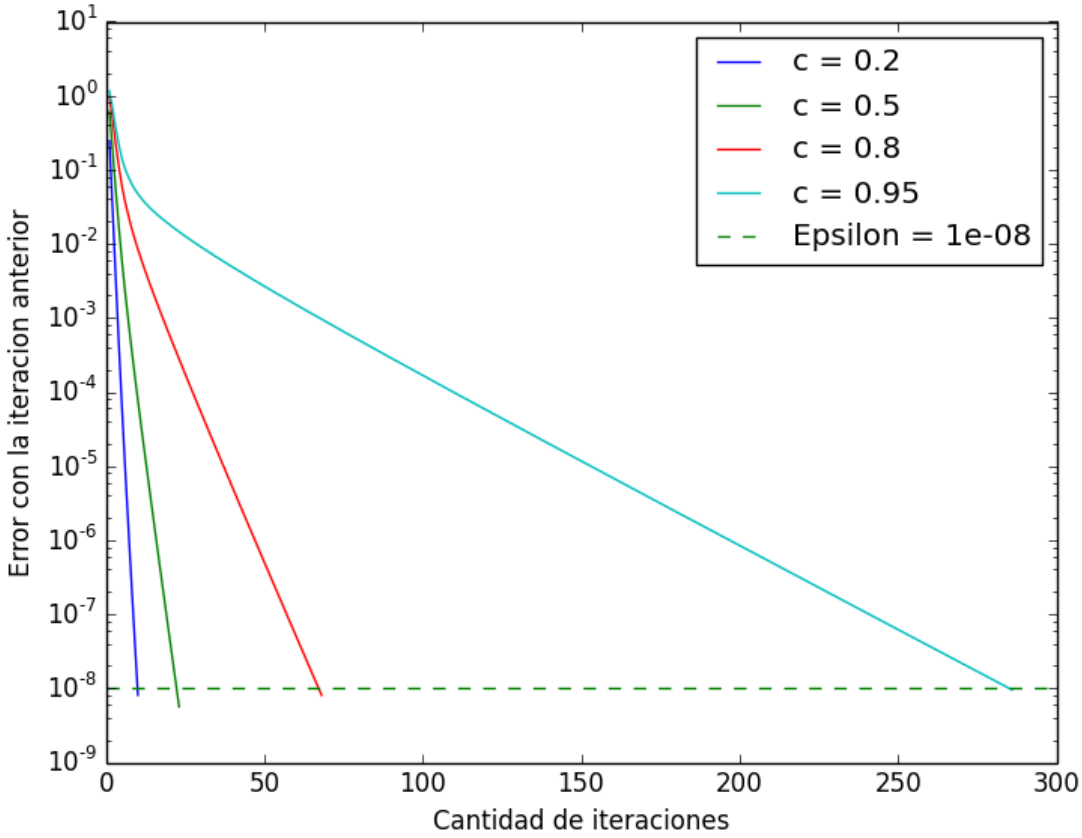

Figura 1: Evolución del error a lo largo de las iteraciones, sin QE

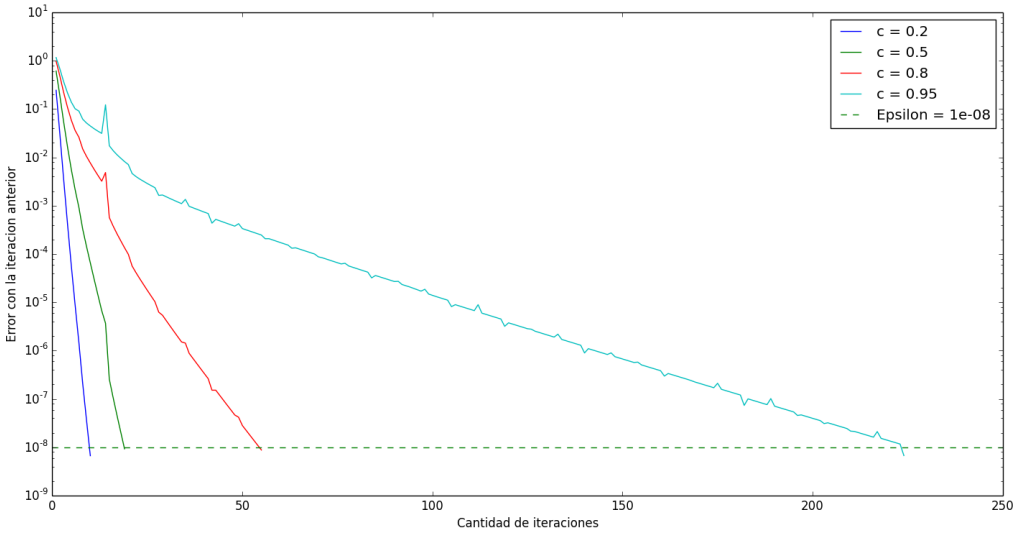

Figura 2: Evolución del error a lo largo de las iteracions, con QE cada 7 iteraciones

El método de extrapolación se aplica cada 7 iteraciones, en el gráfico anterior (Fig. 2) se puede ver que se presentan picos mínimos y máximos, esto se debe a que el método previamente nombrado intenta operar con la combinación lineal de los autovectores que está generando el vector de la iteración en cuestión: minimizar los coeficientes de los autovectores no principales. Esto no necesariamente debería producir una aproximación mejor al autovector buscado, sino que genera condiciones más favorables para la convergencia del método de la potencia.

Se puede obvservar en las figuras que cuanto menor es el valor de c, más rápido converge el método. En estas circunstancias se otorga mayor importancia a la matriz uniforme $\left[\frac{1}{n}\right]$, cuyos autovalores son 1 (dimensión 1) y 0 (dimensión $n-1$). Esta diferencia notable entre los autovalores favorece la convergencia del método de la potencia.

La diferencia que se da al utilizar extrapolación no se ve fácilmente en los gráficos anteriores, por lo que vamos a mostrarlo con mas detalle:

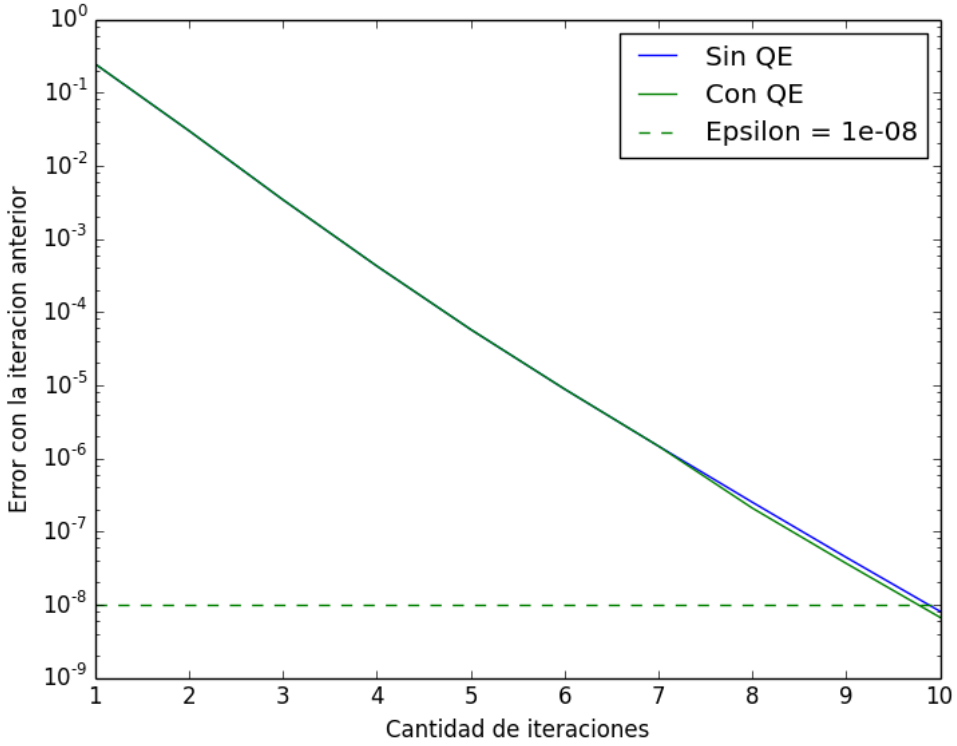

Figura 3: Comparación del uso de QE con $c = 0.2$

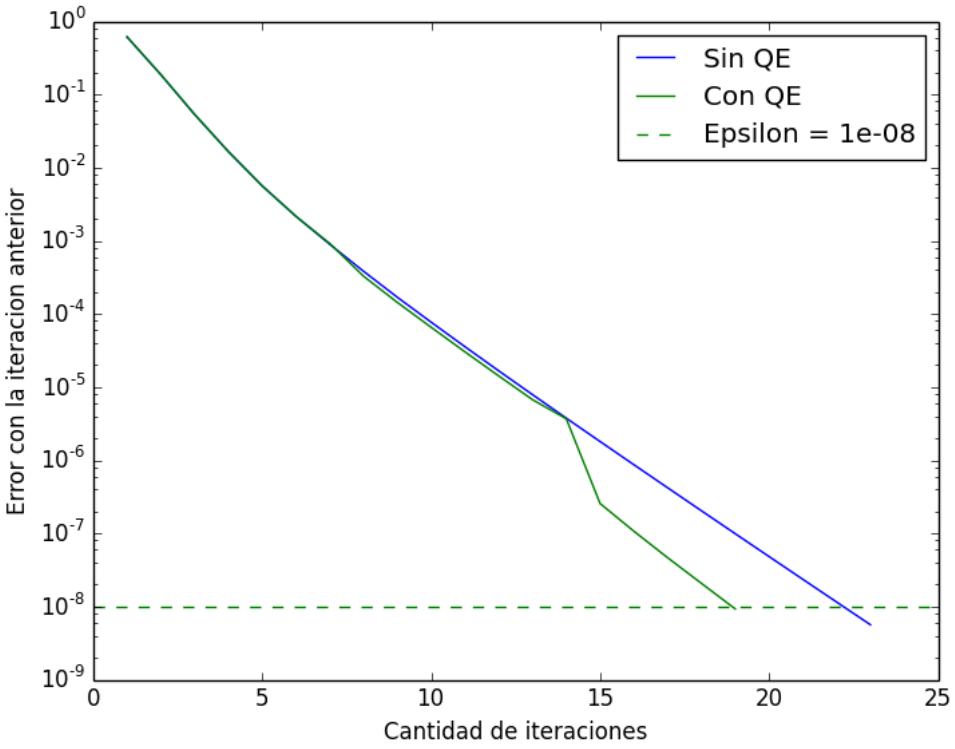

Figura 4: Comparación del uso de QE con $c = 0.5$

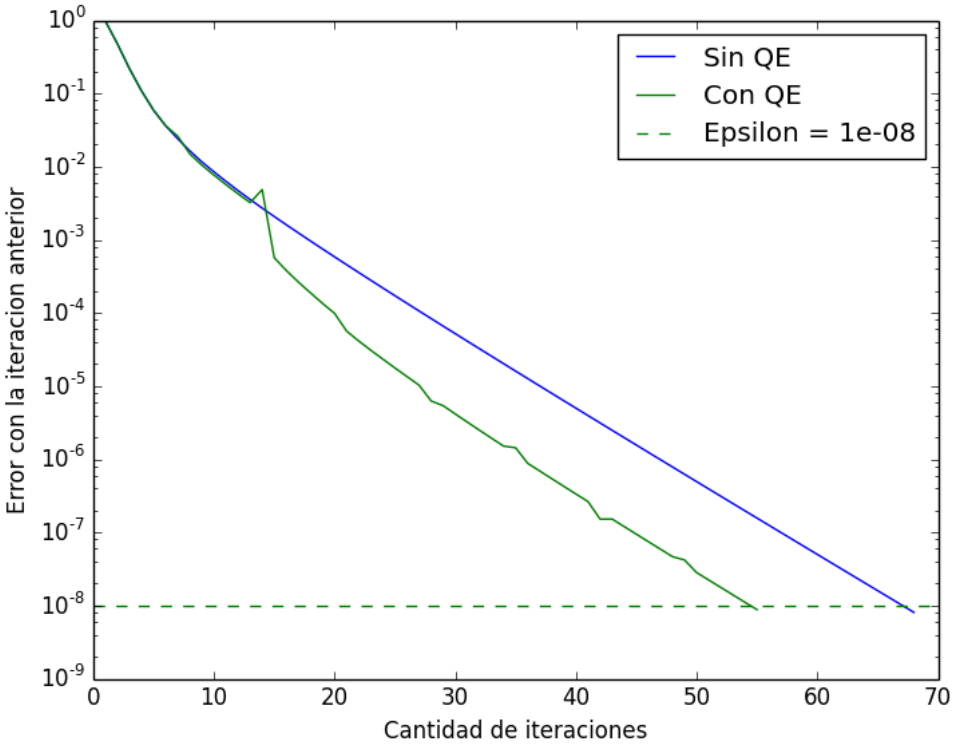

Figura 5: Comparación uso de QE con c = 0.8

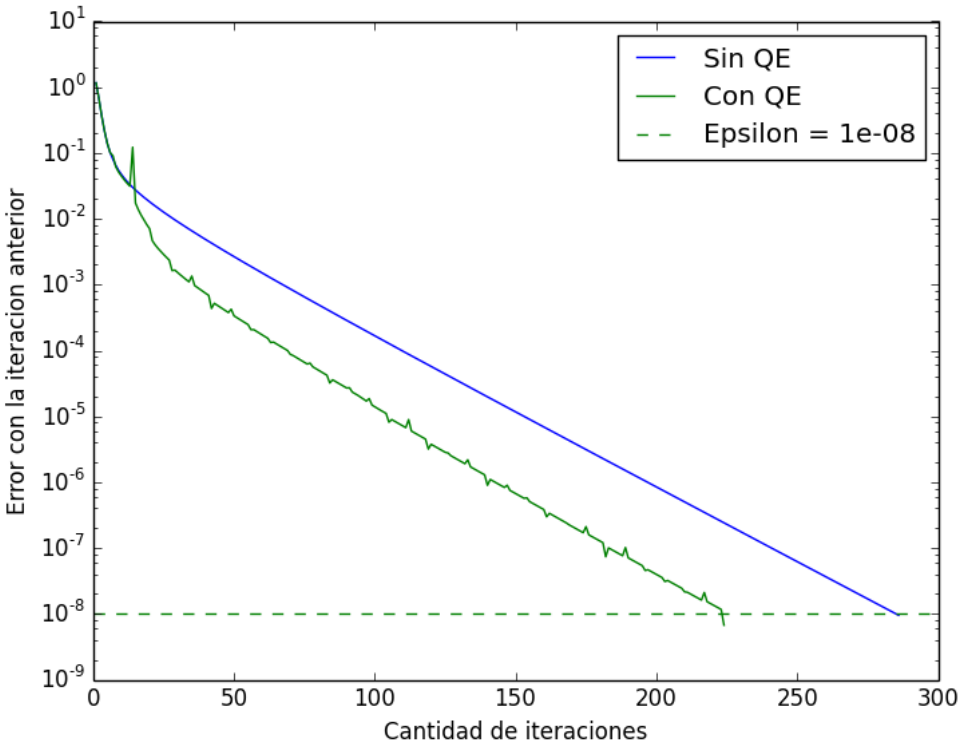

Figura 6: Comparación uso de QE con c = 0.95

En los gráficos anteriores se nota la gran diferencia que hay cuando se aplica el método de extrapolación cuadrática, el cuál parece aumentar el error al principio, pero luego converge más rápidamente. La diferencia tal vez no se aprecia tanto si la convergencia es muy rápida, como en el primer gráfico donde c es igual a 0.2, pero a medida que aumenta la cantidad de iteraciones, más conveniente es utilizarlo. Como dijimos antes, $c = 0.2$ ya provee una situación suficientemente buena para la convergencia del método de la potencia.

3.2. Tiempo de ejecución

Primero medimos tiempos de ejecución para distintos valores de c, aplicando o no QE, cada 7 iteraciones. Aquí notamos, que a pesar de la cantidad de iteraciones utilizando QE es menor, el método de la potencia sin QE converge en menos tiempo de procesamiento (aunque no notablemente). Esto se debe a que realizar la extrapolación implica un tiempo extra de cómputo además de la iteración del método de la potencia, con lo cual, en casos donde éste converge rápidamente, agregar pasos de QE, si bien reduce la cantidad de iteraciones, implica un overhead que no resulta despreciable respecto del tiempo insumido por el algoritmo base. Para ajustar este costo, realizamos nuevas mediciones tomando el caso de mayor cantidad de iteraciones ($c = 0.95$) y variando la periodicidad con la que se aplica el método de extrapolación cuadrática.

A continuación mostramos las tablas de los valores obtenidos para esta experimentación, que puede ser reproducida siguiendo las instrucciones que se encuentran en el README.

Estos son los resultados arrojados por el algoritmo, primero con periodicidad de QE cada 7 iteraciones (Cuadro 1), y luego para el caso particular de $c = 0.95$ con periodicidad variable (Cuadro 2). Como aclaración de la segunda tabla, pQE hace referencia a la periodicidad con la que aplicamos QE en cantidad de iteraciones:

	Sin QE	Con QE
c = 0.2	2 segs.	3 segs.
c = 0.5	6 segs.	5 segs.
c = 0.8	15 segs.	16 segs.
c = 0.95	65 segs.	69 segs.

Cuadro 1: Tiempo de ejecución para distintos valores de c, aplicando o no QE.

Como dijimos, el cuadro 1 muestra que el tiempo de ejecución aplicando QE cada 7 iteraciones del algoritmo es

	Tiempo de ejecución	Cantidad de iteraciones
pQE = 5	78 segs.	229 iters.
pQE = 20	59 segs.	225 iters.
pQE = 35	54 segs.	218 iters.
pQE = 50	52 segs.	213 iters.
pQE = 65	52 segs.	218 iters.
pQE = 70	53 segs.	219 iters.
pQE = 90	52 segs.	222 iters.
pQE = 105	52 segs.	221 iters.
pQE = 120	57 segs.	240 iters.
pQE = 135	57 segs.	241 iters.

Cuadro 2: Tiempo de ejecución y cantidad de iteraciones para la aplicación de QE con distinta periodicidad, $c = 0.95$.

mayor que el que no lo utiliza.

Como podemos observar el cuadro 1, lo que suponíamos es un hecho: variar el momento en que se aplica el método de extrapolación cuadrática así como también la cantidad de veces que se aplica, ayuda a acelerar aún más la velocidad de convergencia del método de la potencia estándar. Es interesante apreciar asimismo, que aún teniendo el mismo tiempo de ejecución para algunos valores de pQE, la cantidad de iteraciones es distinta y esto se debe a que variar el momento en que se aplica QE, y no sólo la cantidad de veces que se aplica es un factor a tener en cuenta. Queda pendiente esta experimentación, la cual es determinar la estrategia más conveniente a la hora de distribuir la aplicación de QE, es decir, ¿conviene aplicar el método cada cierta cantidad de iteraciones fija, de manera uniforme, o dependiendo de otros factores?

Como última observación, el mejor resultado se obtuvo con una periodicidad de 50 iteraciones, donde QE fue aplicado 4 veces logrando cantidad de iteraciones mínima al igual que tiempo de ejecución mínimo, contrastando contra todo los otros resultados obtenidos (recordemos del cuadro 1, que el tiempo de ejecución sin QE para esta situación es de 65 segundos, valor que es mayor que todos los listados en la tabla 2).

3.3. Relevancia de las páginas

A continuación incluimos un ejemplo reducido para observar cómo funciona efectivamente el ranking mediante este algoritmo. A partir del script proporcionado por la cátedra, construimos el grafo de conectividad para algunas páginas con links entre sí (de diferentes diarios y medios) y otras de interés general o de computación que no están apuntadas por éstas. En el cuadro 3 presentamos el ordenamiento obtenido, para los cuatro valores de c con los que trabajamos anteriormente.

Tal como esperábamos ver, las 15 páginas que habíamos notado que no tenían links entrantes, son las últimas en la tabla, independientemente del valor de c y, no casualmente, tienen todas la misma relevancia.

También se observa que a medida que el valor de c aumenta, y por ende, el comportamiento aleatorio tiene menos preponderancia, se agrupan al principio las páginas que efectivamente guardan relación cercana (Olé, Clarín y sus secciones, Ciudad, La Nación, etc), mientras que páginas más generales como YouTube o Taringa pierden importancia. Resaltamos para ejemplificar esto clarin/deportes y YouTube. Pensando en una web exclusivamente compuesta por estas páginas esta situación tiene sentido: un portal de videos es menos relevante que una página de deportes, dado que buena parte de los enlaces existentes en esta red están dedicados a noticias.

www.ole.com.ar	0.0484629
www.clarin.com	0.0468603
www.ciudad.com.ar	0.0430782
www.clasificados.clarin.com	0.0430782
www.youtube.com	0.0420166
www.taringa.net	0.0420166
canchallena.lanacion.com.ar	0.0418644
www.lanacion.com.ar	0.0418644
www.zonaprop.com.ar	0.0405957
www.rollingstone.com.ar	0.0405957
maps.google.com.ar	0.0385152
www.twitter.com	0.0385152
www.clarin.com/deportes	0.0373568
www.9gag.com	0.0350138
www.stackoverflow.com	0.0350138
www.facebook.com	0.0350138
www.hotmail.com	0.0350138
www.gmail.com	0.0350138
www.assembla.com	0.0350138
www.github.com	0.0350138
www.mercadolibre.com.ar	0.0350138
www.yahoo.com	0.0350138
www.pagina12.com.ar	0.0350138
www.mamapuntocero.com.ar	0.0350138
www.infobae.com	0.0350138
www.google.com	0.0350138

(a) Rank obtenido con $c = 0.2$

www.ole.com.ar	0.0708548
www.clarin.com	0.0679806
www.ciudad.com.ar	0.0551093
www.clasificados.clarin.com	0.0551093
canchallena.lanacion.com.ar	0.0476948
www.lanacion.com.ar	0.0476948
www.zonaprop.com.ar	0.0445151
www.rollingstone.com.ar	0.0445151
www.youtube.com	0.0429253
www.taringa.net	0.0429253
www.clarin.com/deportes	0.0371144
maps.google.com.ar	0.0357711
www.twitter.com	0.0357711
www.9gag.com	0.0286169
www.stackoverflow.com	0.0286169
www.facebook.com	0.0286169
www.hotmail.com	0.0286169
www.gmail.com	0.0286169
www.assembla.com	0.0286169
www.github.com	0.0286169
www.mercadolibre.com.ar	0.0286169
www.yahoo.com	0.0286169
www.pagina12.com.ar	0.0286169
www.mamapuntocero.com.ar	0.0286169
www.infobae.com	0.0286169
www.google.com	0.0286169

(b) Rank obtenido con $c = 0.5$

www.ole.com.ar	0.122585
www.clarin.com	0.119996
www.ciudad.com.ar	0.0862634
www.clasificados.clarin.com	0.0862634
canchallena.lanacion.com.ar	0.0492588
www.lanacion.com.ar	0.0492588
www.zonaprop.com.ar	0.0445675
www.rollingstone.com.ar	0.0445675
www.clarin.com/deportes	0.0422953
www.youtube.com	0.032933
www.taringa.net	0.032933
maps.google.com.ar	0.0256146
www.twitter.com	0.0256146
www.9gag.com	0.0182961
www.stackoverflow.com	0.0182961
www.facebook.com	0.0182961
www.hotmail.com	0.0182961
www.gmail.com	0.0182961
www.assembla.com	0.0182961
www.github.com	0.0182961
www.mercadolibre.com.ar	0.0182961
www.yahoo.com	0.0182961
www.pagina12.com.ar	0.0182961
www.mamapuntocero.com.ar	0.0182961
www.infobae.com	0.0182961
www.google.com	0.0182961

(c) Rank obtenido con $c = 0.8$

www.ole.com.ar	0.205635
www.clarin.com	0.204461
www.ciudad.com.ar	0.138847
www.clasificados.clarin.com	0.138847
www.clarin.com/deportes	0.0559973
canchallena.lanacion.com.ar	0.028682
www.lanacion.com.ar	0.028682
www.zonaprop.com.ar	0.0256032
www.rollingstone.com.ar	0.0256032
www.youtube.com	0.0145039
www.taringa.net	0.0145039
maps.google.com.ar	0.0109709
www.twitter.com	0.0109709
www.9gag.com	0.00743788
www.stackoverflow.com	0.00743788
www.facebook.com	0.00743788
www.hotmail.com	0.00743788
www.gmail.com	0.00743788
www.assembla.com	0.00743788
www.github.com	0.00743788
www.mercadolibre.com.ar	0.00743788
www.yahoo.com	0.00743788
www.pagina12.com.ar	0.00743788
www.mamapuntocero.com.ar	0.00743788
www.infobae.com	0.00743788
www.google.com	0.00743788

(d) Rank obtenido con $c = 0.95$

Cuadro 3: Rank obtenido para distintos valores de c

4. Conclusiones

En este T.P. trabajamos primero con un método numérico para hallar un autovector principal y luego le aplicamos una modificación para mejorar su convergencia, la cual debimos estudiar en la práctica para terminar de ajustar su funcionamiento eficiente. Por otro lado, combinamos esto con el algoritmo PageRank que a partir de ciertas formulaciones teóricas propone una manera de determinar la importancia de las páginas de una red, buscando un autovector principal de una matriz en particular. Dado que este algoritmo se utiliza en general con instancias muy grandes, debimos aplicar ciertas optimizaciones para que fuera realizable.

En cuanto a la implementación del método numérico básico, utilizamos el método de la potencia, y seguimos lo propuesto en la publicación que citamos al comienzo, lo que nos permitió optimizar tanto espacio de almacenamiento como tiempo de cómputo, permitiendo resolverlo para matrices de dimensiones muy grandes ($n > 685000$) aunque con muy poca densidad (menos del 0.001 % de elementos no nulos). Esto fue posible gracias a la representación esparsa de la matriz que utilizamos, que pudimos mantener gracias a la modificación que comentamos en la introducción.

En cuanto a la modificación para acelerar su convergencia, implementamos el método de Extrapolación Cuadrática que citamos al principio y en todos los casos, vimos que reducía la cantidad de iteraciones necesarias para converger. Sin embargo, dado que esto implicaba tiempo extra de cómputo, debimos controlar cuándo y cuántas veces ejecutarlo para que resultara efectivamente en una optimización sin agregar overhead excesivo. Para el caso que estudiamos, decidimos que un valor óptimo es aplicarlo cada 50 iteraciones del método estándar, aunque como dijimos ya varias veces, cuantas más iteraciones necesite el método para converger, más provecho se puede sacar de este otro enfoque. Sin embargo, dado que esta cantidad es difícil de predecir en general, consideramos que para disparar la extrapolación es necesario un análisis más fino que obtenga información a partir de otros aspectos, antes que fijar su ejecución a intervalos determinados.

En cuanto al algoritmo PageRank, vimos que darle más importancia al comportamiento aleatorio acelera la convergencia pero degrada el sentido del orden obtenido.