Obligatorio 1
Métodos Númericos 2015

Matías Estrada — 3.858.107-2 — estrada.matias@gmail.com

Gonzalo Javiel — 4.666.259-5 — gonzalo.javiel@gmail.com

Andrés Tipoldi — 3.834.437-9 — tipoldi@gmail.com

Raúl Speroni — 4.177.047-8 — raulsperoni@gmail.com

25 de septiembre de 2015

Índice

1. Fundamentos

1.1. Matriz de Google

Se llama Matriz de Google a una matriz estocástica particular utilizada por el algoritmo *PageRank* del buscador de la empresa Google. *PageRank* cuenta la cantidad y calidad de enlaces hacia una página para estimar la importancia de una página web, la hipótesis primordial es que las páginas más relevantes probablemente tengan mayor cantidad de enlaces hacia ellas. La matriz de Google se puede representar como un grafo orientado, donde los nodos son páginas web y las aristas son enlaces entre páginas. [9] El *PageRank* de cada página puede ser calculado iterativamente a partir de la matriz de Google usando por ejemplo el método de las potencias, para que el método converja la matriz debe ser estocástica, irreducible y aperiódica.

Matriz de Adyacencias Sea N la cantidad de páginas, se define A matriz de adyacencias que representa la relación entre enlaces como sigue:

$$A_{i,j} = \begin{cases} 1 & \text{si } j \text{ tiene un enlace hacia } i \\ 0 & \text{en otro caso} \end{cases} \tag{1}$$

Matriz de Markov A partir de A se construye una matriz S que correspondiente a las transiciones en una cadena de Markov [4]. Sea k_j el número de enlaces salientes del nodo i a todos los demás nodos:

$$S_{i,j} = \begin{cases} A_{i,j}/k_j & \text{si } j \text{ tiene un enlace hacia } i \\ 0 & \text{en otro caso} \end{cases} \tag{2}$$

Aquellas columnas j cuyos valores son todos cero representan nodos sin enlaces salientes, dichos vectores son reemplazados por otro cuyos valores sean $\frac{1}{N}$. Por construcción la suma de todos los elementos de cada columna j es la unidad, por lo tanto S está bien definida, pertenece a la clase de Cadenas de Markov y a la clase de operadores de Perron-Frobenius.

Matriz de Google Se puede definir la matriz de Google como sigue:

$$G_{i,j} = \alpha S_{i,j} + (1-\alpha)\frac{1}{N} \tag{3}$$

1.2. Cadenas de Markov

Proceso markoviano Un proceso markoviano es un proceso estocástico en el cual su comportamiento y su evolución futura no depende más que del estado actual del proceso y no de sus estados pasados.

Cadena de Markov Las Cadenas de Markov son procesos markovianos de espacio de estado discreto. Se pueden separar en Cadenas de Markov de tiempo discreto y Cadenas de Markov de tiempo continuo.

Una cadena de Markov puede definirse como una secuencia de variables aleatorias discretas $\{X_n, n \in N\}$ que poseen la siguiente propiedad

$$P(X_{n+1} = e_{n+1}|X_0 = e_0, X_1 = e_1, \cdots, X_n = e_n) = P(X_{n+1} = e_{n+1}|X_n = e_n), \quad \forall n \in N, \forall e_0, e_1, \cdots, e_n, e_{n+1} \in E \tag{4}$$

Esta propiedad se interpreta como la afirmación que la probabilidad (condicional) del proceso alcance el estado futuro x_{n+1} dados los estados pasado1s $x_0, x_1, \cdots x_{n-1}$ y el estado actual x_n es independiente de los estados pasados y depende solamente del estado presente x_n (el pasado no influye en el futuro más que a través del presente).

Cadena de Markov ergódica Una cadena de Markov es ergódica cuando es irreductible y su espacio de estado es un conjunto de estados ergódicos, o sea que todos sus estados son son recurrente positivos y aperiódicos.

Teorema de ergodicidad Toda Cadena de Markov de tiempo continuo homogénea finita e irreductible es ergódica. Si la cadena es infinita, puede no ser ergódica (una condición suficiente para que una cadena irreductible sea ergódica es que los valores v_i estén acotados superiormente).

1.3. Teorema de Perron Frobenius

Dada una matriz cuadrada $A \in M_{n\times n}$ con entradas positivas $a_{i,i} \geqslant 0$ entonces,

a. Existe un valor propio $\lambda > 0$ tal que $Av = \lambda v$, donde el vector propio correspondiente es $v > 0$

b. Ese valor propio es mayor en módulo que todos los valores propios asociados a A .

c. Cualquier otro vector propio positivo de A es múltiplo de v.

Por construcción la matriz de Google es positiva, por lo que según el teorema de Perron Forbenius el valor propio de mayor módulo y el vector propio asociado al mismo existen.

1.4. Vector propio dominante de la matriz de Google

$$Gv_1 = v_1 \tag{5}$$

Sea v_1 el vector propio asociado al valor propio 1 que verifica la ecuación 5 la i-esima componente de v_1 representa la probabilidad de estar en la página i en un determinado momento n y por lo tanto v_1 representa la distribución de probabilidad de las páginas en el momento n. El vector v_1 se denomina distribución estacionaria. Siendo que nos brinda la probabilidad de encontrarse en cada página v_1 puede usarse para ordenar las páginas de acuerdo a su probabilidad. Este es uno de los componentes fundamentales del motor de búsqueda de Google.

2. Puntuación de Sitios Web

2.1. Método de potencias

Métodos de la potencias Es un método iterativo que calcula sucesivas aproximaciones a los autovectores y autovalores de una matriz. El objetivo del método es, dada una matriz M_{NxN} calcular el valor propio dominante y un vector propio asociado.

Sean los valores propios $|\lambda_1|, ..|\lambda_N|$ que verifican $|\lambda_1| > |\lambda_2| > ... > |\lambda_N|$ y su vectores propios asociados $v_1, v_2, ..., v_n$

Sea $x^{(0)} = \alpha_1 v_1 + \alpha_2 v2 + \alpha_N v_n$ con $\alpha_1 \neq 0$

$$\begin{cases} y^{(j+1)} = Mx^{(j)} \\ c_{(j+1)} = \text{ componente dominante de } y^{(j+1)} \\ x^{(j+1)} = (\frac{1}{c_{(j+1)}})y^{(j+1)} \text{ Normalizado de } y^{(j+1)} \end{cases}$$

Entonces se cumple:

La sucesión de escalares c_j tiende al valor propio dominante λ_1

$$\{c_j\} \to \lambda_1, j = 0 \dots n$$

La sucesión de vectores $\{x^{(j)}\}$ tiende a un vector propio normalizado asociado a λ_1

$$\{x^{(j)}\} \to v_1, j = 0 \dots n$$

2.2. Sistema lineal de ecuaciones

$$Gv = \lambda v \tag{6}$$

$$(G - \lambda)v = 0 \tag{7}$$

Subespacio Krylov Un subespacio de Krylov de orden r generado por una matriz $A \in M_{NxN}$ y un vector v, es el subespacio vectorial generado por $A^k v$ con $k < r$:

$$K_r(A, v) = span(v, Av, \ldots, A^{r-1}v) \tag{8}$$

Matriz de Hassenberg superior Una matriz de Hassenberg superior tiene todos ceros por debajo de la primera subdiagonal.

Método de Arnoldi El método Arnoldi puede ser usado para encontrar todos los valores y vectores propios de la matriz $G \in M_{NxN}$, pertenece a la clase de algoritmos de álgebra lineal que producen un resultado parcial después de un número relativamente bajo de iteraciones. El algoritmo fue creado en principio para tranformar una matriz en forma Hassenberg superior, pero se encontró más adelante que el método podía ser usado para encontrar valores y vectores propios para matrices esparsas de gran tamaño de forma iterativa. Inicialmente el método construye bases del subespacio Krylov, después de construido el subespacio, con m elegido como cantidad de bases, podemos calcular aproximaciones a los valores y vectores propios de la matriz esparsa original G. La matriz de Hassenberg $H \in M_{NxN}$ resultante del método es la clave para calcular esas aproximaciones ya que sus valores propios asociados $\lambda_i^{(m)}$ conocidos como *Valores de Ritz* convergerán hacia los valores propios de la matriz esparsa de gran tamaño A [1].

Los vectores propios de la matriz de Google G pueden calcularse como sigue:

a. Calcular el valor propio de interés de H
b. Obtener el vector propio asociado a ese valor propio.
c. El vector correspondiente de G se obtiene por:

$$v_i^{(m)} = V_m y_i^{(m)} \tag{9}$$

donde $v_i^{(m)}$ es el vector buscado en G, V_m es el vector con bases del subespacio de Krylov y $y_i^{(m)}$ es el vector propio de H asociado al valor propio de interés.

Sistema Alternativo la ecuación 3 determina una matriz G que puede ser expresada como se muestra en la ecuación 10, dónde A es la ***Matriz de Adyacencias*** y D es una matriz diagonal definida por 12 a 13 y con e el vector de todos unos:

$$G = \alpha AD + ez^T \tag{10}$$

$$c_j = \sum_i a_{ij} \tag{11}$$

$$d_{j,j} = \begin{cases} 1/c_j & \text{si } c_j \neq 0 \\ 1/n & \text{en otro caso} \end{cases} \tag{12}$$

$$z_j = \begin{cases} (1-p)/n & \text{si } c_j \neq 0 \\ 1/n & \text{en otro caso} \end{cases} \tag{13}$$

El sistema lineal presentado en 6 puede ser reescrito a partir de la ecuación 10 como se ve en la ecuación 14 según [5]:

$$(I - \alpha AD)v = \beta e \tag{14}$$

Dicho sistema puede ser resuelto con un método iterativo, como por ejemplo el ***Método de Gauss Sidel***.

Algorithm 1 Método Arnoldi

```
1:  procedure ARNOLDI
2:      v_0 = arbitrary nonzero starting vector
3:      v_1 = v_0/||v_0||_2
4:      for j = 1, 2... do
5:          w = Av_j
6:          for j = 1 : j do
7:              h_ij = w * v_i
8:              w = w − h_ij v_i
9:          end for
10:         h_{j+1,j} = ||w||_2
11:         if h_{j+1,j} = 0 then stop
12:         end if
13:         v_{j+1} = w/h_{j+1,j}
14:     end for
15: end procedure
```

2.3. Implementación

Se implementó el ***Método de las Potencias*** partiendo del fundamento matemático presentado en 2.1, fue necesario también implementar una función para transformar una *Matriz de Adyacencia* en una *Matriz de Google* .
Se implementó además el ***Método de Arnoldi*** según se describe en 2.2 siguiendo el algoritmo 1.
Luego de evaluar los resultados se implementó la versión del ***Método de Gauss Sidel*** visto en el curso para resolver el sistema lineal alternativo 14.
Todas las implementaciones se pueden encontrar en el anexo 1.

2.4. Comparación

Se compararon los métodos presentados en la sección 2 entre sí y contra un método de las potencias optimizado encontrado online que utilizamos como *Gold Standard*. Como datos de entrada se utilizaron las *Matrices de Adyacencias* de los dominios de internet de Stanford y Harvard que cumplen con las características realistas en cuanto a su tamaño y dispersión que requiere el problema. Además se utilizó una matriz generada por código de tamaño 50 llamada *Matriz X* y una de tamaño 500 llamada *Matriz Y*. Dichas matrices tienen entre 0 y 15 links en cada fila y es tomada de [1]

Cuadro 1: Datos utilizados

	Tamaño	Fuente
X	50	CreateMatrix(50)
Y	500	CreateMatrix(500)
Harvard	500	[6, harvard500.mat]
Stanford	9.914	[7, wb-cs-stanford.mat]

Los resultados obtenidos se pueden reproducir ejecutando el programa *Comparacion.m* presente en los anexos.

Convergencia Como se puede ver en los cuadros 2 y 3 ambos métodos convergen a la misma solución coincidente con la solución encontrada por el método Gold Standard sobre los datos de las matrices generadas artificialmente, sin embargo como se observa en 4 y 5 el ***Método de Arnoldi*** no converge a la solución correcta mientras que el ***Método de la Potencia*** sí lo hace.

Es notable que el ***Método de la Potencia*** implementado requiere que se convierta la **Matriz de Adyacencias** a una **Matriz de Google**, éste paso consume un tiempo considerablemente mayor que el ***Método de Arnoldi*** que va realizando ésta transformación en cada iteración. Para la matriz de Stanford por ejemplo, la ejecución demora más de 20 minutos dependiendo del hardware.

El ***Método de Gauss-Seidel*** agregado por los malos resultados del ***Método de Arnoldi*** arroja resultados distintos en las matrices generadas por código X e Y pero logra buenos resultados con las matrices de Harvard y Stanford.

Se puede ver en las figuras desde 1 a 6 la velocidad de convergencia de los métodos de Potencias y Arnoldi. Las figuras 2.4 y 2.4 muestran las primeras 50 posiciones del *PageRank* de las matrices de Harvard y Stanford.

Figura 1: Método Potencias, Matriz X

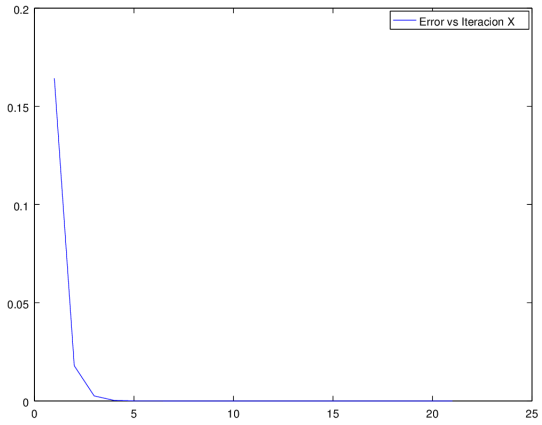


Figura 2: Método Arnoldi, Matriz X

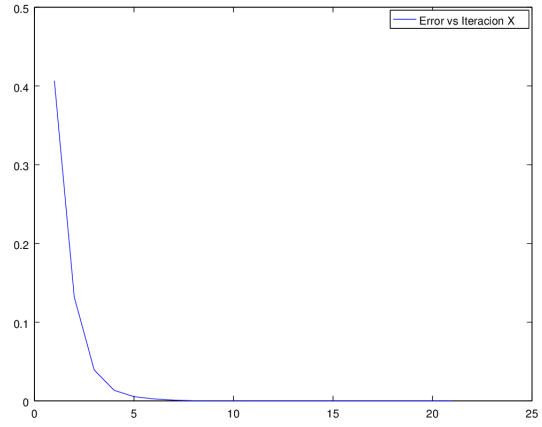


Figura 3: Método Potencias, Matriz Harvard

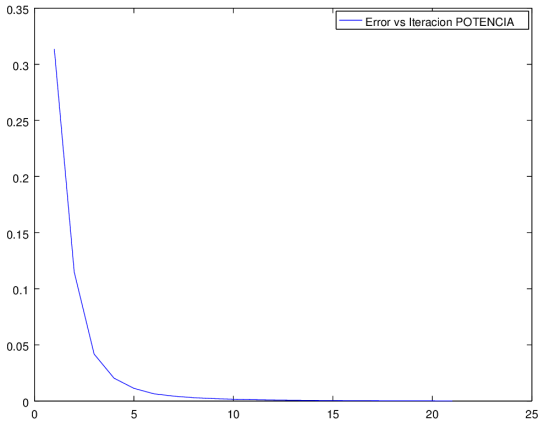


Figura 4: Método Arnoldi, Matriz Harvard

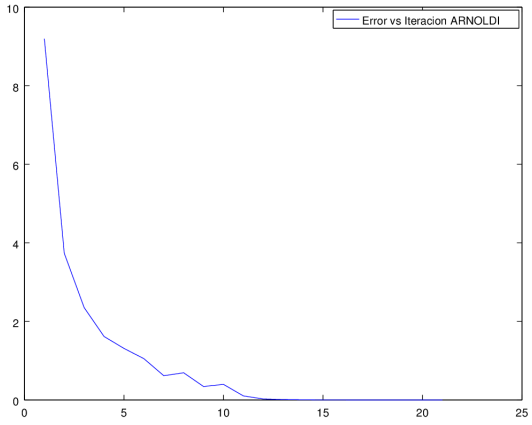


Figura 5: Método Potencias, Matriz Stanford

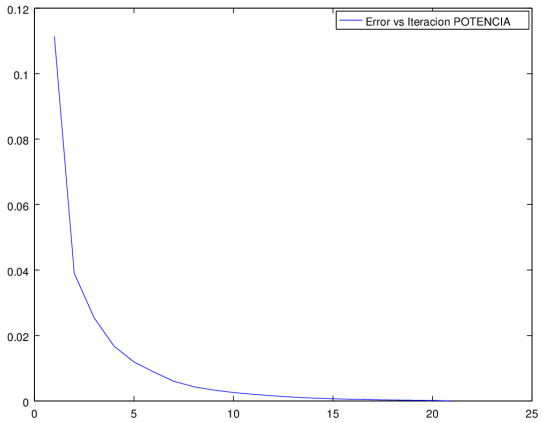


Figura 6: Método Arnoldi, Matriz Stanford

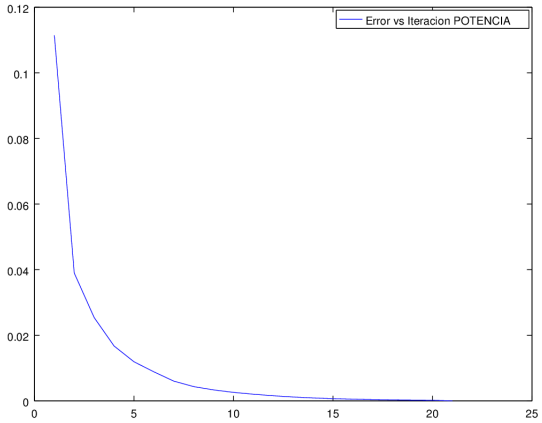


Figura 7: Page Rank, Gold Standard, Matriz Harvard

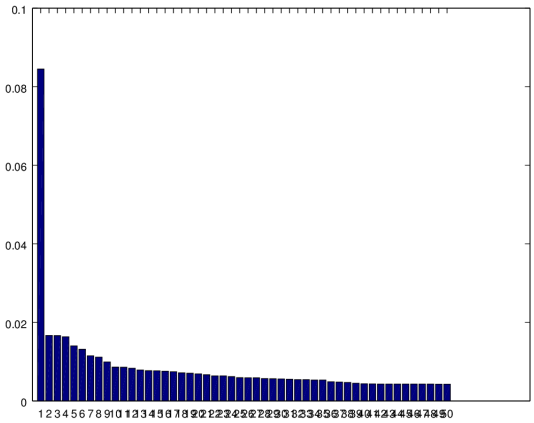
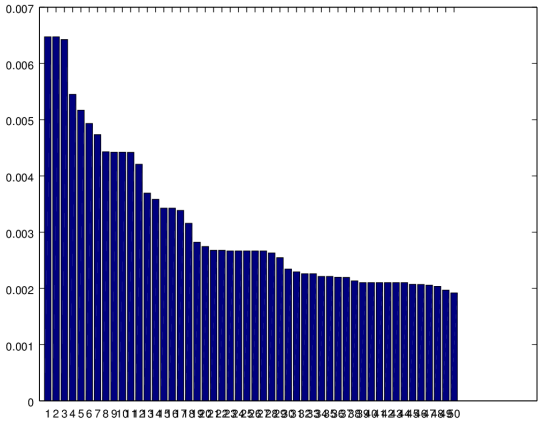


Figura 8: Page Rank, Gold Standard, Matriz Stanford



Cuadro 2: Resultados Matriz **X**

	Potencias	Arnoldi	Gauss-Seidel	Gold Standard
Probabilidad	0.0479616	0.0481269	0.0369617	0.0479427
Indice	21	21	21	21
Iteración	20	20	16	7

Cuadro 3: Resultados Matriz **Y**

	Potencias	Arnoldi	Gauss-Seidel	Gold Standard
Probabilidad	0.00557936	0.0055878	0.0051286	0.0055656
Indice	153	153	153	153
Iteración	20	20	10	6

Cuadro 4: Resultados Matriz **Harvard**

	Potencias	Arnoldi	Gauss-Seidel	Gold Standard
Probabilidad	0.0842866	0.0520312	0.0842759	0.0844947
Indice	1	335	1	1
Iteración	20	20	20	11

Cuadro 5: Resultados Matriz **Stanford**

	Potencias	Arnoldi	Gauss-Seidel	Gold Standard
Probabilidad	0.00648635	0.0264089	0.00648123	0.0064746
Indice	270	6562	270	270
Iteración	20	20	20	21

3. Profundización

3.1. El segundo valor propio y el problema de Spam

Si bien el vector propio dominante de la matriz de Google es fundamental ya que representa la distribución estacionaria de la cadena de Markov y por lo tanto indica la probabilidad para un usuario de encontrarse en cada página, los vectores propios asociados al segundo valor propio tiene también importancia según [5] ya que puede ser usado para detectar spam.

Grafo de la Web Se puede ver la Web como un grafo dirigido donde los nodos son las páginas web y los enlaces son los links de una página a otra. Este grafo no es fuertemente conectado. Se dice que el grafo de la web tiene

forma de moño: hay tres categorías principales para las páginas web, *IN*, OUT, *SCC*. Un usuario puede pasar de una página de *IN* a una de *SCC*, de una de *SCC* puede pasar a una de OUT y puede pasar de una de SCC a otra de SCC, sin embargo no es posible pasar de una de *SCC* a una de *IN*, ni de OUT a *SCC* o a *IN*. Notablemente se ha encontrado que *IN* y OUT son de tamaños similares mientras que *SCC* es mucho más grande. La mayoría de las páginas web pertenecen a éstas categorías. [8] [2]

Se demuestra en [3] que para toda matriz $A = [cP + (1-c)E]^T$, donde P es una matriz estocastica por filas nxn, E es una matriz no negativa nxn de rango 1 estocástica por filas, $0 \leq c \leq 1$, el segundo vector propio de A tiene módulo $|\lambda_2| \leq c$. Más aún si P tiene al menos dos subconjuntos cerrados irreducibles entonces $\lambda_2 = c$.
Los vectores propios correspondientes al segundo valor propio $\lambda_2 = c$ son indicadores de certeza en el grafo de la web. En particular cada par de nodos hoja en el grafo SCC para la cadena de Markov P corresponde a un vector propio de A con valor propio c. Los nodos hoja en SCC son aquellos subgrafos que pueden tener links entrantes pero no tienen links salientes a otra categoria. Los Spammers utilizan habitualmente dichas estructuras para alterar el PageRank. El análisis de éstas estructuras puede permitir construir estrategias para combatir este tipo de spam. [3]

Referencias

[1] Erik Andersson and Per-Anders Ekström. Investigating google's pagerank algorithm. *Report in Scientifc Computing, advanced course-Spring*, 2004.

[2] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer Networks*, 33(1–6):309 – 320, 2000.

[3] Taher Haveliwala and Sepandar Kamvar. The second eigenvalue of the google matrix. Technical Report 2003-20, Stanford InfoLab, 2003.

[4] P. Ravi Kumar, K. L. Alex Goh, and K. S. Ashutosh. Application of markov chain in the pagerank algorithm. *Pertanika Journal of Science & Technology*, 21(2):541 – 553, 2013.

[5] Alex Sangers and Martin B. van Gijzen. The eigenvectors corresponding to the second eigenvalue of the google matrix and their relation to link spamming. *Journal of Computational and Applied Mathematics*, 277:192 – 201, 2015.

[6] Harvard University. Matriz. `http://www.cise.ufl.edu/research/sparse/matrices/MathWorks/Harvard500.html`.

[7] Stanford University. Matriz. `http://www.cise.ufl.edu/research/sparse/matrices/Gleich/wb-cs-stanford.html`.

[8] Stanford University. The web graph. `http://nlp.stanford.edu/IR-book/html/htmledition/the-web-graph-1.html`.

[9] Rebecca S. Wills. Google's page rank: The math behind the search engine. *Mathematical Intelligencer*, 28(4):6 – 11, 2006.

Anexos

Listing 1: Potencia.m

```
function [autoVector,autoValor, errores, iter]=Potencia(X,maxIteraciones,toleran
    %param A matriz
    %param maxIteraciones
    %param tolerancia
        A = ObtenerMatrizDeTransicion(X);

    errores = zeros(1,maxIteraciones);

        %x0 = linspace(1,1,columns(A));
        x0 = zeros(1,columns(A)); % Inicializo x^{x0} con 0's
        x0(1) = 1; % le asigno un 1 a la primer componente

        %maxIteraciones = 1000; %Maxima cantidad de iteraciones
        iter = 0; %Numero de iteracion
        %tolerancia = 10^-10; % Tolerancia
        err = 1 ; % inicializo para que entre al while

        c = 0; % AutoValor
        autoValorAnterior = -1;

        x = x0(:); % Paso el vector x^{x0} como columna

        % Si la cantidad de iteraciones no supero el maximo y la diferencia
        % de dos valores consecutivos es mayor a la tolerancia
        while (iter < maxIteraciones && err > tolerancia)
                autoValorAnterior = c; % lo guardo para la proxima iteracion
                y = A*x; % y^{(j+1)} = Ax^{(j)}
                c = abs(max(y));   % Componente dominante de y^{(j+1)}
                x = (1 / c) * y; % Normalizado de y^{(j+1)}
        err = abs(c-autoValorAnterior);
                iter++;
        errores(iter) = err;
        end
        autoVector = x/sum(x);
        autoValor = sum(autoVector);
end

function [T] = obtenerMatrizGoogle(G)
        [n,n] = size(G);
        d = zeros(1, n);
        p = zeros(1, n);
```

```
        for j = 1:n
        L{j} = find(G(:,j));
        c(j) = length(L{j});
                if (c(j)== 0)
                        d(j)= 1;
                else
                        P (:, j) = G (:, j) /c(j);
                end
                p(j)= 1/n;
                end
        D = p'*d;
        T = P + D;
endfunction

function A = ObtenerMatrizDeTransicion(G)

        vectorSumaColumnas = sum(G);
        [numeroColumnas,numeroFilas] = size(G) ;


        for j = 1:numeroColumnas
                %j
                if (vectorSumaColumnas(j) == 0)
                        A(:,j) = (linspace(1/numeroColumnas,1/numeroColumnas,nun
                else
                        A(:,j) = G(:,j) / vectorSumaColumnas(j); %no tiene 0 po
                end
        end

        v = (1/numeroColumnas);
        e = ones(1,numeroColumnas);
        A = (0.85) * A + (0.15) * v * e'*e;

endfunction
```

Listing 2: Sistema.m

```
function [autoVector,autoValor,errores,iter] = Sistema(A,max_bases,alpha,tol)
        #chequear columnas vacias
        d = columnasVacias(A);

        #crear bases
        [V,H,errores,I] = BaseArnoldi(A,max_bases,alpha,tol,d);

        #busco vector y valor propio de H.
        #uso eig porque H es chica, aun cuando A es grande.
        [EVEC,EVAL] = eig(full(H));
        #Saco dimensiones de H para mostrar lo anterior.
        ch = columns(full(H));
        rh = rows(full(H));
        #busco el mayor valor propio y su indice
        [eigval ind] = max(diag(EVAL));

        #busco el vector propio primario de H
        firstvec = EVEC(:,ind);

        #traer vector propio primario
        eigvec = V*firstvec;

        #normalizar
        eigvec = eigvec./norm(eigvec,1);

        #si negativo
        if (eigvec(1)<0) eigvec = abs(eigvec); end

        autoVector = eigvec;
        autoValor = sum(autoVector);
        iter = I;
endfunction

%%%%%%%%%%%%%%%%%%
%% Base Arnoldi %%
%%%%%%%%%%%%%%%%%%

function [V,h,res,I] = BaseArnoldi(A,max_bases,alpha,tol,d)
        #vector residuos
        res = zeros(1,max_bases+1);
        #tamano de primer columna de A
        n = size(A,1);
        #matriz sparsa de max_bases+1 * max_bases+1
        h = sparse(zeros(max_bases+1,max_bases+1));
```

```
        #vector base inicial de n unos/n
        V = ones(n,1)/n;
        #vector base inicial normalizado
        V = V/norm(V,2);
        #iteracion
        for(j=1:max_bases)
                #Calculo proximo vector base
                #V(:,j) es columna j de V.
                #w es el vector a ortogonalizar
                w = MxV(A,V(:,j),alpha,d);
                #los w son ortogonales y generan el subespacio Kn
                #ortogonalizo con gran-shmidt
                for(i=1:j)
                        h(i,j)=w'*V(:,i);
                        w = w-h(i,j)*V(:,i);
                end
                #voy llenando la matriz H
                h(j+1,j)=norm(w,2);

                if h(j+1,j) < 1e-12
                        break
                end
                #para la proxima iteracion
                vtemp = w/h(j+1,j);
                V = [V vtemp];

                #calculo el residuo de forma eficiente
                [EVEC,EVAL] = eig(full(h(1:j,1:j)));
                [tmp ind] = max(diag(EVAL));
                cheapres = h(j+1,j)*abs(EVEC(j,ind));

                %fprintf(1,'iter:%d res=%g \t(tol=%g)\n',j,cheapres,tol)
                #condicion de parada
                if cheapres < tol;
                        break
                end
                res(j) = cheapres;
        end

        I = j;
        if j == max_bases
                h = h(1:j+1,1:j+1);
        else
                h = h(1:j,1:j);
        end
endfunction
```

```
%%%%%%%%%%%%%%%%%%%%
%% columnasVacias  %%
%%%%%%%%%%%%%%%%%%%%

function [d] = columnasVacias(A)
#devuelve vector marcando columnas vacias
        #d matriz esparsa de #numerodefilasdeA x 1
        d = sparse(size(A,1),1);
        #de 1 a numero de columnas
        for i=1:size(A,2)
                #si la suma de toda la columnas es cero
                if sum(A(:,i))==0
                        #marco su indice en el vector a devolver.
                        d(i)=1;
                end
        end
endfunction

%%%%%%%%
%%MxV  %%
%%%%%%%%

function [y] = MxV(A,v,alpha,d)
        #Tamano de A
        n = size(v,1);
        #Vector de todos 1
        e = ones(n,1);
        #Vector resultado
        y = alpha*A*v;
        # ?? esto viene de check empty columns.
        beta = alpha*d'*v + (1-alpha)*e'*v;
        y = y+beta*e/n;
endfunction
```

Listing 3: PageRankMN.m

```
function [autoVector, iter] = PageRankMN(G, maxIteraciones,p,tolerancia)

        [n,n] = size(G);

        I = eye(n,n); % creo la matriz identidad

        % Quiero obtener la matriz D
        D = zeros(n,n);
        for j = 1:n
        L{j} = find(G(:,j));
        c(j) = length(L{j}); % aca obtengo cuantos unos hay en la columna
                if (c(j) > 0)
                        D(j,j) = 1/c(j);
                end
        end

        A = (I - p*(G*D));
        b = ones(n,1);

        [x,iteracionGS] = GaussSeidel(A,b,tolerancia,maxIteraciones);

        autoVector = x/sum(x); % Normalizado
        iter = iteracionGS; % Cuantas iteraciones llevo encontrar

endfunction

%%%%

function [x,iter] = GaussSeidel(A,b,umbral, iterMax)

%Inicializacion de las variables
n = length(A);
LUD = zeros (n);
x = zeros (n, 1);
error = ones (1, n);

%Genero una matriz con los valores de la Matriz A pero sin su diagonal.
LUD = A - diag(diag(A));
iter = 0;

%iterMax=70; % control de iteraciones
%umbral= 10^-10;  % umbral de parada

while (iter < iterMax && max(error) >= umbral)
```

```
    xAnterior = x;
    for i = 1:n
      x(i) = (b(i)-LUD(i,:)*xAnterior) / A(i,i);
          error(i) = abs(xAnterior(i) - x(i)); % calculo el error por item del vec
    end
    iter ++;

end
endfunction
```

Listing 4: Comparacion.m

```
function Comparacion()

    addpath("matrices");
    load("matrices/harvard500.mat");
    HARV = G - diag(diag(G));
    load("matrices/wb-cs-stanford.mat");
    STANFORD = Problem.A - diag(diag(Problem.A));
    X = createMatrix(50);
    X = X - diag(diag(X));
    Y = createMatrix(500);
    Y = Y - diag(diag(Y));

    disp("POTENCIA vs ARNOLDI vs GAUSS-SEIDEL\n");
    #set(0, 'defaultfigurevisible', 'off');

    maxIteraciones = 20;
    tolerancia = 10^-20;
    alpha = 0.85;

    disp("###################### X ############################\n\n");
    disp("POTENCIA X: ");
    [autoVector,autoValor,errores,iter] = Potencia(X, maxIteraciones,toleran
    [probabilidad indice] = max(autoVector);
    fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
    #plot(errores,";Error vs Iteracion X;");
    #print('ErrorVsIteracionPotencia-X','-dpng');

    disp("ARNOLDI X: ");
    [autoVector,autoValor,errores,iter] = Sistema(X, maxIteraciones,alpha,to
    [probabilidad indice] = max(autoVector);
    fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
    #plot(errores,";Error vs Iteracion X;");
    #print('ErrorVsIteracionArnoldi-X','-dpng');

    disp("PAGERANKPOW X: ");
    [autoVector,iter] = pagerankpow(X);
    [probabilidad indice] = max(autoVector);
    fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n',probabilidad,in

    disp("GAUSS-SEIDEL");
    [autoVector, iter] = PageRankMN(X, maxIteraciones,alpha,tolerancia);
    [probabilidad indice] = max(autoVector);
    fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n\n\n',probabilidad
```

```
disp("##################### Y ############################\n\n");
disp("POTENCIA Y: ");
[autoVector,autoValor,errores,iter] = Potencia(Y, maxIteraciones,toleran
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
#plot(errores,";Error vs Iteracion Y;");
#print('ErrorVsIteracionPotencia-Y','-dpng');

disp("ARNOLDI Y: ");
[autoVector,autoValor,errores,iter] = Sistema(Y, maxIteraciones,alpha,to
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
#plot(errores,";Error vs Iteracion Y;");
#print('ErrorVsIteracionArnoldi-Y','-dpng');

disp("PAGERANKPOW Y: ");
[autoVector,iter] = pagerankpow(Y);
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n',probabilidad,in

disp("GAUSS-SEIDEL");
[autoVector, iter] = PageRankMN(Y, maxIteraciones,alpha,tolerancia);
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n\n\n',probabilidad


disp("##################### HARVARD ############################\n\n");
disp("POTENCIA HARVARD: ");
[autoVector,autoValor,errores,iter] = Potencia(HARV, maxIteraciones,tole
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
#plot(errores,";Error vs Iteracion POTENCIA;");
#print('ErrorVsIteracionPotencia-Harvard','-dpng');

disp("ARNOLDI HARVARD: ");
[autoVector,autoValor,errores,iter] = Sistema(HARV, maxIteraciones,alpha
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
#plot(errores,";Error vs Iteracion ARNOLDI;");
#print('ErrorVsIteracionArnoldi-Harvard','-dpng');

disp("PAGERANKPOW HARVARD: ");
[autoVector,iter] = pagerankpow(HARV);
[probabilidad indice] = max(autoVector);
fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n',probabilidad,in
```

```
        disp("GAUSS-SEIDEL");
        [autoVector, iter] = PageRankMN(HARV, maxIteraciones,alpha,tolerancia);
        [probabilidad indice] = max(autoVector);
        fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n\n\n',probabilidad

        disp("##################### STANFORD ########################\n\n");
        disp("POTENCIA STANFORD: ");
        [autoVector,autoValor,errores,iter] = Potencia(STANFORD, maxIteraciones,
        [probabilidad indice] = max(autoVector);
        fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
        #plot(errores,";Error vs Iteracion POTENCIA;");
        #print('ErrorVsIteracionPotencia-Stanford','-dpng');

        disp("ARNOLDI STANFORD: ");
        [autoVector,autoValor,errores,iter] = Sistema(STANFORD, maxIteraciones,a
        [probabilidad indice] = max(autoVector);
        fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d AutoValor= %g \n',p
        #plot(errores,";Error vs Iteracion ARNOLDI;");
        #print('ErrorVsIteracionArnoldi-Stanford','-dpng');

        disp("PAGERANKPOW STANFORD: ");
        [autoVector,iter] = pagerankpow(STANFORD);
        [probabilidad indice] = max(autoVector);
        fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n',probabilidad,in

        disp("GAUSS-SEIDEL");
        [autoVector, iter] = PageRankMN(STANFORD, maxIteraciones,alpha,toleranci
        [probabilidad indice] = max(autoVector);
        fprintf(1,'Probabilidad: %g Indice= %d Iteracion= %d \n\n\n',probabilidad


endfunction

#crear matriz esparsa de dim*dim
function [A] = createMatrix(dim)
        A = sparse(dim,dim);
        maxnel = min(16,dim);
        for i = 1:dim
                nel = floor(rand(1)*maxnel);
                if(nel == 0)
                        val = 0;
                else
                        val = 1/nel;
                end
                for j = 1:nel
```

```
                col_ind = ceil(rand(1)*dim);
                while(A(col_ind,i) ~= 0)
                        col_ind = ceil(rand(1)*dim);
                end
                A(col_ind,i) = val;
            end
        end
endfunction
```