3. Code

a = 0.85; % alpha

% a = 0.75;

M = [

1/11, (1-a)/11, (1-a)/11, (9\*a+2)/22, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (10\*a+1)/11, (9\*a+2)/22, (8\*a+3)/33, (9\*a+2)/22, (9\*a+2)/22, (9\*a+2)/22, (9\*a+2)/22, (1-a)/11, (1-a)/11;

1/11, (10\*a+1)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (8\*a+3)/33, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (9\*a+2)/22, (9\*a+2)/22, (9\*a+2)/22, (9\*a+2)/22, (10\*a+1)/11, (10\*a+1)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (8\*a+3)/33, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

1/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11, (1-a)/11;

];

x = [1/11; 1/11; 1/11; 1/11; 1/11; 1/11; 1/11; 1/11; 1/11; 1/11; 1/11];

disp(M);

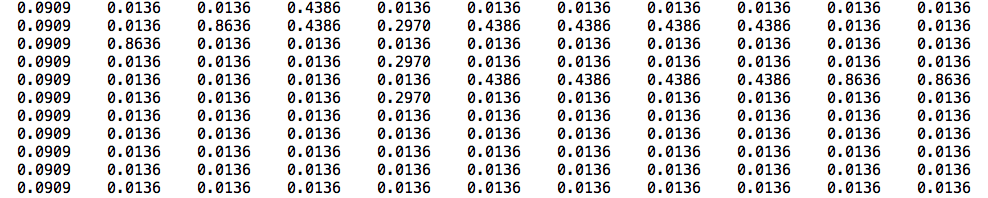
for i = 1:15 % iteration times

x = M\*x;

end

x

3 (a). M when



6 (a).

**ConstructPdpe.m**

function [P, d, p, e] = ConstructPdpe(G)

% Get size of matrix G

[row\_cnt, col\_cnt] = size(G);

% Init. P, d, p, e

P = sparse(row\_cnt, col\_cnt);

d = sparse(row\_cnt, 1);

p = sparse(ones(row\_cnt, 1) / row\_cnt);

e = sparse(ones(row\_cnt, 1));

% Set values

for j = 1:1:col\_cnt

% Get the count of all links from node j

link\_cnt = full(sum(G(:,j)));

if link\_cnt ~= 0

for i = 1:1:row\_cnt

is\_link = full(G(i, j));

if is\_link ~= 0

P(i, j) = 1.0 / link\_cnt;

end

end

else

% node j is a dead end

d(j, 1) = 1;

end

end

**MyPageRank.m**

function [p, iter] = MyPageRank(G, alpha)

tol = 0.0000001;

% Make Sparse Matrix P, d

[P, d, p, e] = ConstructPdpe(G);

% Get size of matrix G

[R, C] = size(G);

% Iterate

p\_last = p;

iter = 0;

while 1

p = alpha\*(P\*p+e\*(transpose(d)\*p)/R)+(1-alpha)\*e/R;

iter = iter + 1;

% Check

p\_diff = abs(p - p\_last);

if max(p\_diff(:,1)) < tol

break;

end

p\_last = p;

end

6 (b).

**q6b.m**

G = sparse(15, 15);

G(1, 5) = 1;

G(2, 1) = 1;

G(2, 3) = 1;

G(3, 2) = 1;

G(3, 4) = 1;

G(4, 8) = 1;

G(5, 2) = 1;

G(5, 9) = 1;

G(6, 3) = 1;

G(6, 9) = 1;

G(7, 2) = 1;

G(7, 12) = 1;

G(8, 3) = 1;

G(8, 12) = 1;

G(9, 1) = 1;

G(9, 13) = 1;

G(10, 5) = 1;

G(10, 6) = 1;

G(10, 7) = 1;

G(10, 9) = 1;

G(10, 14) = 1;

G(11, 6) = 1;

G(11, 7) = 1;

G(11, 8) = 1;

G(11, 12) = 1;

G(11, 14) = 1;

G(12, 4) = 1;

G(12, 15) = 1;

G(13, 10) = 1;

G(13, 14) = 1;

G(14, 13) = 1;

G(14, 15) = 1;

G(15, 11) = 1;

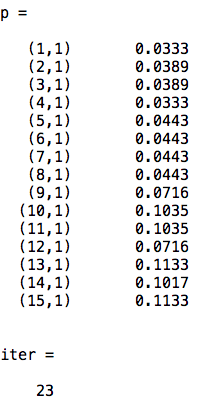
G(15, 14) = 1;

[p, iter] = MyPageRank(G, 0.75);

p

iter

**Result:**



6 (c).

**q6c.m**

load('math\_uwaterloo.mat');

alpha = 0.85;

[p, iter] = MyPageRank(G, alpha);

[y I] = sort(p, 'descend');

for n = 1:min(length(I), 20)

disp([num2str(n) ': ' U{I(n)}]);

end



6 (d).

**q6d.m**

load('math\_uwaterloo.mat');

alpha = [0.15 0.6 0.75 0.95];

for m = 1:length(alpha)

disp(alpha(m));

[p, iter] = MyPageRank(G, alpha(m));

disp(iter);

[y I] = sort(p, 'descend');

for n = 1:min(length(I), 20)

disp([num2str(n) ': ' U{I(n)}]);

end

end

|  |  |
| --- | --- |
| **Alpha ()** | **Iteration times** |
| 0.15 | 6 |
| 0.6 | 21 |
| 0.75 | 37 |
| 0.95 | 201 |

When increases, the number of iterations also increases. Also, the number of iterations increases become faster when increases.

The reason for this relationship is that the convergence of algorithm becomes slower when increases, and slower convergence of algorithm needs more iterations to make the final values of p have the given tolerance.