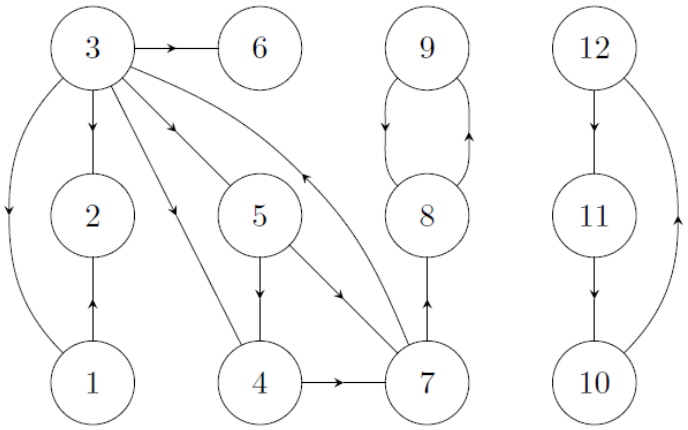
Project 2: Investigating the PageRank Algorithm
Math 458
Kartik Balodi
Github Link to All Code: https://github.com/kartikbalodi/math458_project2

(1) Write a matlab file that implements the page rank algorithm and finds the importance vector associated with a link matrix L that is stochastic and has no cycles. Write your file flexibly, so that it will work whatever size L has.

```
Filename: pageRank.m
% takes in Link Matrix L and returns
% 1) array of ranking by importance of each page in ascending order
% 2) array of corresponding sorted importance for each ranked page
% 3) array of unsorted importance where the index of array corresponds to
%      each website and its respective importance
% 4) iteration in the while loop in calcImportanceVector, to study further
%      the rate of convergence
% uses calcImportanceVector function as helper
function [pageRankAsc,sortedImportanceAsc,importanceVector,iterations] = pageRank(L)
       % obtain importance for each website, total = 1
       [importanceVector,iterations] = calcImportanceVector(L);
       x = importanceVector.';
       % create key-value map of importance as key and website index as value
       % so that the map is sorted by key(importance)in ASCENDING order
       [sortedImportanceAsc,pageRankAsc] = sort(x);
end

% note: L must be a stochastic matrix for this function to work
%      if not, use makeStochastic function first (defined below)
function [x, iterations] = calcImportanceVector(L)
       websiteCount = length(L);
       %initialize x0 by giving equal importance to each webpage
       x0 = ones(websiteCount, 1)/websiteCount;
       x = x0;
       normComparison = norm(L*x-x, 2);
       iterations = 0;
       % repeat steps till norm of Ax=x to 1e-10 closeness based on norm
       while normComparison > 1e-20
       iterations = iterations + 1;
       x = L*x;
       normComparison = norm(L*x-x, 2);
       end
end
```

(2) **Toy Example:** Consider the toy example of a network of 12 websites depicted below.



(a) Write down the link matrix A.

$$A = \begin{bmatrix} 0 & 0 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/5 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(b) Modify the link matrix so it has no columns all of which are 0. This is the matrix B.

$$B = \begin{bmatrix} 0 & 1/12 & 1/5 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1/12 & 1/5 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 1/5 & 0 & 1/2 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 1/5 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 1/5 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 0 & 1 & 1/2 & 1/12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 1/2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1/12 & 0 & 0 & 0 & 1/12 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

This can be verified easily by Matlab.

q2bcdef.m

```
A = [0 0 1/5 0 0 0 0 0 0 0 0 0;
     1 0 1/5 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 1/2 0 0 0 0 0;
     0 0 1/5 0 1/2 0 0 0 0 0 0 0;
     0 0 1/5 0 0 0 0 0 0 0 0 0;
     0 0 1/5 0 0 0 0 0 0 0 0 0;
     0 0 0 1 1/2 0 0 0 0 0 0 0;
     0 0 0 0 0 0 1/2 0 1 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 1;
     0 0 0 0 0 0 0 0 0 1 0 0];
B = fixZeroColumns(A);
disp("B"); disp(B);
```

fixZeroColumns.m

```
% fixes all columns with all-zeros to make each value 1/n for that
function B = fixZeroColumns(L)
    B = L;
    toAdd = 1/length(L); % 1/n
    % find sum of columns, obtain the index of those that are 0
    colSum = sum(L);
    zeroColIndex = find(~colSum);
    zeroColIndex = zeroColIndex.';
    % take the indexes of the 0-columns and add 1/n to them
    for i=1:length(zeroColIndex)
        B(:,zeroColIndex(i)) = toAdd;
    end
end
```

Command Window

```
>> q2bcdef
B
         0    0.0833    0.2000         0         0    0.0833         0         0         0         0         0         0
    1.0000    0.0833    0.2000         0         0    0.0833         0         0         0         0         0         0
         0    0.0833         0         0         0    0.0833    0.5000         0         0         0         0         0
         0    0.0833    0.2000         0    0.5000    0.0833         0         0         0         0         0         0
         0    0.0833    0.2000         0         0    0.0833         0         0         0         0         0         0
         0    0.0833    0.2000         0         0    0.0833         0         0         0         0         0         0
         0    0.0833         0    1.0000    0.5000    0.0833         0         0         0         0         0         0
         0    0.0833         0         0         0    0.0833    0.5000         0    1.0000         0         0         0
         0    0.0833         0         0         0    0.0833         0    1.0000         0         0         0         0
         0    0.0833         0         0         0    0.0833         0         0         0         0    1.0000         0
         0    0.0833         0         0         0    0.0833         0         0         0         0         0    1.0000
         0    0.0833         0         0         0    0.0833         0         0         0    1.0000         0         0
```

(c) Modify B so that it has no cyclic paths. This is the matrix

$$C_{u,\alpha} = \alpha B + (1-\alpha)ue^T$$

where $e = [1, 1, \ldots, 1]^T$.

Here we set u = 1/12 * [1, 1, … , 1]^T and alpha = 0.9 (will test with more values of alpha in part d).

q2bcdef.m

```
        0 0 1/5 0 0 0 0 0 0 0 0;
        0 0 0 1 1/2 0 0 0 0 0 0;
        0 0 0 0 0 0 1/2 0 1 0 0;
        0 0 0 0 0 0 0 1 0 0 0;
        0 0 0 0 0 0 0 0 0 0 1;
        0 0 0 0 0 0 0 0 0 1 0];
    B = fixZeroColumns(A);
    %disp("B"); disp(B);
    alpha = 0.9; N = 12;
    u = 1/N*ones(N, 1);
    C = removeCyclicPaths(B, u, alpha);
    disp("C"); disp(C);
```

removeCyclicPaths.m

```
% B = nxn matrix and ensure u is nx1 upon input and sum(u)=
function C = removeCyclicPaths(B, u, alpha)
    e = ones(length(B), 1);
    C = alpha*B + (1-alpha)*u*e.';
end
```

Command Window

```
>> q2bcdef
C
    0.0083    0.0833    0.1883    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.9083    0.0833    0.1883    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.4583    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.1883    0.0083    0.4583    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.1883    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.1883    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.0083    0.9083    0.4583    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.4583    0.0083    0.9083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.0083    0.9083    0.0083    0.0083    0.0083    0.0083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.9083    0.0083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.0083    0.0083    0.9083
    0.0083    0.0833    0.0083    0.0083    0.0083    0.0833    0.0083    0.0083    0.0083    0.9083    0.0083    0.0083
```

(d) Run your page rank m-file on $C_{u,\alpha}$. Explore the page rankings you get for different choices of α and u. Explain. Does the importance vector depend on the vector u and/or α? How and why? Does the rate of convergence depend on u and/or α? How and why?

Intuitively we notice that decreasing alpha reduces the dependency on the original matrix B and increases the dependency on the matrix given by u*eT. Thus, lowering alpha should reduce the importance of cycles in the graph, as it is less likely that they would get stuck within the cycle depending on the distribution of nodes given by the matrix u in u*eT.

We also expect that u plays a part in the results of the importance vector. However, the importance of u only takes precedence when the value of alpha decreases, since only then do we have that u dominates the results of the distribution given by C(u,alpha). Also, choosing dominant values for a certain element of u means the corresponding node will have a lot more weight/attention/probability/proportion directed towards it when superposing u*eT with B, so nodes with higher weights/proportion in u will have much higher importance in the PageRank algorithm.

The rate of convergence should depend on alpha, as a high value of alpha means that if the link matrix contains cycles, they will continually be given higher importance and more cases of cycling would occur. Thus, we should choose a lower alpha to increase effect of u*eT - the superposed link matrix that gives the same superposed values for every webpage - to obtain a lower rate of convergence. Hypothetically, we do not anticipate different values of u to significantly affect the rate of convergence, they would only impact who the values converge to.

To investigate the effect of alpha on rate of convergence and importance,
set u = 1/12 [1 1 … 1]T, and vary alpha = 1.0, 0.9, 0.7, 0.5, 0.3, 0.1, 0.01, 0.001, 0
Full results in appendix

Alpha = 1.0 No convergence, so the program runs infinitely since cycles are present.	Alpha = 0.9 alpha = 9.000000e-01 Time taken: 3.967200e-03 # of iterations(repeats for x=Cx): 283 Websites by importance (most to least): 1. Website: 8 Importance 2.263572e-01 2. Website: 9 Importance 2.159713e-01 3. Website: 12 Importance 1.224980e-01 4. Website: 11 Importance 1.224980e-01 5. Website: 10 Importance 1.224980e-01 6. Website: 7 Importance 4.385164e-02 7. Website: 2 Importance 3.421282e-02 8. Website: 3 Importance 3.198304e-02 9. Website: 4 Importance 2.610979e-02 10. Website: 6 Importance 1.800675e-02 11. Website: 5 Importance 1.800675e-02 12. Website: 1 Importance 1.800675e-02

Alpha = 0.7

```
alpha = 7.000000e-01
Time taken: 1.522000e-04
# of iterations(repeats for x=Cx): 84
Websites by importance (most to least):
1.  Website: 8  Importance 1.608629e-01
2.  Website: 9  Importance 1.438523e-01
3.  Website: 12 Importance 1.041609e-01
4.  Website: 11 Importance 1.041609e-01
5.  Website: 10 Importance 1.041609e-01
6.  Website: 7  Importance 8.262293e-02
7.  Website: 2  Importance 6.744164e-02
8.  Website: 3  Importance 6.016629e-02
9.  Website: 4  Importance 5.355659e-02
10. Website: 6  Importance 3.967155e-02
11. Website: 5  Importance 3.967155e-02
12. Website: 1  Importance 3.967155e-02
```

Alpha = 0.5

```
alpha = 5.000000e-01
Time taken: 1.688000e-04
# of iterations(repeats for x=Cx): 43
Websites by importance (most to least):
1.  Website: 8  Importance 1.263360e-01
2.  Website: 9  Importance 1.105062e-01
3.  Website: 7  Importance 9.497883e-02
4.  Website: 12 Importance 9.467635e-02
5.  Website: 11 Importance 9.467635e-02
6.  Website: 10 Importance 9.467635e-02
7.  Website: 2  Importance 8.166969e-02
8.  Website: 3  Importance 7.108288e-02
9.  Website: 4  Importance 6.805808e-02
10. Website: 6  Importance 5.444646e-02
11. Website: 5  Importance 5.444646e-02
12. Website: 1  Importance 5.444646e-02
```

Alpha = 0.3

```
alpha = 3.000000e-01
Time taken: 1.403000e-04
# of iterations(repeats for x=Cx): 24
Websites by importance (most to least):
1.  Website: 8  Importance 1.045124e-01
2.  Website: 7  Importance 9.521783e-02
3.  Website: 9  Importance 9.352570e-02
4.  Website: 12 Importance 8.881713e-02
5.  Website: 11 Importance 8.881713e-02
6.  Website: 10 Importance 8.881713e-02
7.  Website: 2  Importance 8.678705e-02
8.  Website: 4  Importance 7.677316e-02
9.  Website: 3  Importance 7.645467e-02
10. Website: 6  Importance 6.675927e-02
11. Website: 5  Importance 6.675927e-02
12. Website: 1  Importance 6.675927e-02
```

Alpha = 0.1

```
alpha = 1.000000e-01
Time taken: 3.576000e-04
# of iterations(repeats for x=Cx): 13
Websites by importance (most to least):
1.  Website: 8  Importance 8.931687e-02
2.  Website: 7  Importance 8.845162e-02
3.  Website: 2  Importance 8.577844e-02
4.  Website: 9  Importance 8.529634e-02
5.  Website: 12 Importance 8.484962e-02
6.  Website: 11 Importance 8.484962e-02
7.  Website: 10 Importance 8.484962e-02
8.  Website: 4  Importance 8.187942e-02
9.  Website: 3  Importance 8.078724e-02
10. Website: 6  Importance 7.798040e-02
11. Website: 5  Importance 7.798040e-02
12. Website: 1  Importance 7.798040e-02
```

Alpha = 0.01

```
alpha = 1.000000e-02
Time taken: 2.770000e-05
# of iterations(repeats for x=Cx): 6
Websites by importance (most to least):
1.  Website: 8  Importance 8.389290e-02
2.  Website: 7  Importance 8.388491e-02
3.  Website: 2  Importance 8.363286e-02
4.  Website: 9  Importance 8.347763e-02
5.  Website: 12 Importance 8.347343e-02
6.  Website: 11 Importance 8.347343e-02
7.  Website: 10 Importance 8.347343e-02
8.  Website: 4  Importance 8.321884e-02
9.  Website: 3  Importance 8.305812e-02
10. Website: 6  Importance 8.280481e-02
11. Website: 5  Importance 8.280481e-02
12. Website: 1  Importance 8.280481e-02
```

Alpha = 0.001

```
alpha = 1.000000e-03
Time taken: 2.650000e-05
# of iterations(repeats for x=Cx): 4
Websites by importance (most to least):
1.  Website: 8  Importance 8.338893e-02
2.  Website: 7  Importance 8.338885e-02
3.  Website: 2  Importance 8.336383e-02
4.  Website: 9  Importance 8.334728e-02
5.  Website: 12 Importance 8.334723e-02
6.  Website: 11 Importance 8.334723e-02
7.  Website: 10 Importance 8.334723e-02
8.  Website: 4  Importance 8.332219e-02
9.  Website: 3  Importance 8.330558e-02
10. Website: 6  Importance 8.328055e-02
11. Website: 5  Importance 8.328055e-02
12. Website: 1  Importance 8.328055e-02
```

Alpha = 0

```
alpha = 0
Time taken: 4.770000e-05
# of iterations(repeats for x=Cx): 0
Websites by importance (most to least):
1.  Website: 12 Importance 8.333333e-02
2.  Website: 11 Importance 8.333333e-02
3.  Website: 10 Importance 8.333333e-02
4.  Website: 9  Importance 8.333333e-02
5.  Website: 8  Importance 8.333333e-02
6.  Website: 7  Importance 8.333333e-02
7.  Website: 6  Importance 8.333333e-02
8.  Website: 5  Importance 8.333333e-02
9.  Website: 4  Importance 8.333333e-02
10. Website: 3  Importance 8.333333e-02
11. Website: 2  Importance 8.333333e-02
12. Website: 1  Importance 8.333333e-02
```

Clearly, as alpha decreases the time taken, which depends upon the number of iterations of repeating x = Lx till x is approximately equal to Lx, decreases. Also, as alpha decreases, we notice that the importance of cycles decreases as well, since the superposed link matrix u*eT starts to dominate and u*eT distributes the probability of visiting any website equally. Also, we observe that as alpha increases the importance of each website smooths out and becomes closer and closer until the case where alpha = 0, where every website is equally ranked and has the same importance. This is due to the fact that u is a probability matrix that makes every website link to every other website including itself with equal probability. If u = [1 0 0 … 0]T, we would expect that when alpha = 0, website 1 would have full importance while the others have no importance.

For the next part of the testing the focus will be on investigating the impact of different u, and how their influence changes with different alphas. For this, we'll investigate

u = [1 0 0 … 0]T,
u = ½*[1 1 0 0 … 0]T,
u = ⅛*[1 1 1 1 1 1 0 0 0 0 0 0]T, and
u = ⅛*[0 … 0 1 1 1 1 1 1]T (investigate what happens when u favors cycles (website 10,11,12))

with alpha = 0.95

u = [1 0 0 … 0]T

```
alpha = 9.500000e-01
Time taken: 8.231000e-04
# of iterations(repeats for x=Cx): 582
Websites by importance (most to least):
1.  Website: 8  Importance 2.288926e-01
2.  Website: 9  Importance 2.233231e-01
3.  Website: 12 Importance 1.175022e-01
4.  Website: 11 Importance 1.175022e-01
5.  Website: 10 Importance 1.175022e-01
6.  Website: 2  Importance 6.515703e-02
7.  Website: 1  Importance 5.905489e-02
8.  Website: 7  Importance 2.286435e-02
9.  Website: 3  Importance 1.673567e-02
10. Website: 4  Importance 1.335596e-02
11. Website: 6  Importance 9.054889e-03
12. Website: 5  Importance 9.054889e-03
```

u = ½*[1 1 0 0 … 0]T

```
alpha = 9.500000e-01
Time taken: 8.413000e-04
# of iterations(repeats for x=Cx): 583
Websites by importance (most to least):
1.  Website: 8  Importance 2.349161e-01
2.  Website: 9  Importance 2.292000e-01
3.  Website: 12 Importance 1.205944e-01
4.  Website: 11 Importance 1.205944e-01
5.  Website: 10 Importance 1.205944e-01
6.  Website: 2  Importance 6.687169e-02
7.  Website: 1  Importance 3.429318e-02
8.  Website: 7  Importance 2.346604e-02
9.  Website: 3  Importance 1.717609e-02
10. Website: 4  Importance 1.370743e-02
11. Website: 6  Importance 9.293175e-03
12. Website: 5  Importance 9.293175e-03
```

u = ⅙*[1 1 1 1 1 1 0 0 0 0 0 0]T

```
alpha = 9.500000e-01
Time taken: 3.082000e-04
# of iterations(repeats for x=Cx): 591
Websites by importance (most to least):
1.  Website: 8  Importance 2.788297e-01
2.  Website: 9  Importance 2.692442e-01
3.  Website: 12 Importance 8.711891e-02
4.  Website: 11 Importance 8.711891e-02
5.  Website: 10 Importance 8.711891e-02
6.  Website: 7  Importance 3.935117e-02
7.  Website: 2  Importance 3.637079e-02
8.  Website: 3  Importance 3.138108e-02
9.  Website: 4  Importance 2.751124e-02
10. Website: 6  Importance 1.865168e-02
11. Website: 5  Importance 1.865168e-02
12. Website: 1  Importance 1.865168e-02
```

u = ⅙*[0 … 0 1 1 1 1 1 1]T

```
alpha = 9.500000e-01
Time taken: 3.215000e-04
# of iterations(repeats for x=Cx): 580
Websites by importance (most to least):
1.  Website: 8  Importance 2.284115e-01
2.  Website: 9  Importance 2.256546e-01
3.  Website: 11 Importance 1.732738e-01
4.  Website: 12 Importance 1.732738e-01
5.  Website: 10 Importance 1.732738e-01
6.  Website: 7  Importance 1.131774e-02
7.  Website: 3  Importance 5.706283e-03
8.  Website: 2  Importance 2.758373e-03
9.  Website: 4  Importance 2.086461e-03
10. Website: 6  Importance 1.414550e-03
11. Website: 5  Importance 1.414550e-03
12. Website: 1  Importance 1.414550e-03
```

Observations:

- Different u has slightly but negligible impact on the number of iterations run.
- The cycles take top importance for each u when alpha is large. Take the bottom left quadrant, u priorities websites 1-6, but {8,9} and {10,11,12} still take top precedence because alpha is large supersedes the impact of u.
- Time taken varies slightly, probably due to slight variations in Matlab's calculation time for the matrices - the key though is the number of iterations (i.e. setting x=Cx and repeating until Cx and x are close enough

with alpha = 0.5

u = [1 0 0 … 0]T

```
alpha = 5.000000e-01
Time taken: 1.516000e-04
# of iterations(repeats for x=Cx): 43
Websites by importance (most to least):
1.  Website: 1  Importance 5.136116e-01
2.  Website: 2  Importance 2.704174e-01
3.  Website: 8  Importance 3.158399e-02
4.  Website: 9  Importance 2.762654e-02
5.  Website: 7  Importance 2.374471e-02
6.  Website: 12 Importance 2.366909e-02
7.  Website: 11 Importance 2.366909e-02
8.  Website: 10 Importance 2.366909e-02
9.  Website: 3  Importance 1.777072e-02
10. Website: 4  Importance 1.701452e-02
11. Website: 6  Importance 1.361162e-02
12. Website: 5  Importance 1.361162e-02
```

u = ½*[1 1 0 0 … 0]T

```
alpha = 5.000000e-01
Time taken: 4.390000e-05
# of iterations(repeats for x=Cx): 43
Websites by importance (most to least):
1.  Website: 2  Importance 4.056261e-01
2.  Website: 1  Importance 2.704174e-01
3.  Website: 8  Importance 4.737598e-02
4.  Website: 9  Importance 4.143981e-02
5.  Website: 7  Importance 3.561706e-02
6.  Website: 12 Importance 3.550363e-02
7.  Website: 11 Importance 3.550363e-02
8.  Website: 10 Importance 3.550363e-02
9.  Website: 3  Importance 2.665608e-02
10. Website: 4  Importance 2.552178e-02
11. Website: 6  Importance 2.041742e-02
12. Website: 5  Importance 2.041742e-02
```

u = ⅙*[1 1 1 1 1 1 0 0 0 0 0 0]T

```
alpha = 5.000000e-01
Time taken: 4.070000e-05
# of iterations(repeats for x=Cx): 46
Websites by importance (most to least):
1.  Website: 2  Importance 1.597096e-01
2.  Website: 4  Importance 1.330913e-01
3.  Website: 3  Importance 1.204880e-01
4.  Website: 6  Importance 1.064731e-01
5.  Website: 5  Importance 1.064731e-01
6.  Website: 1  Importance 1.064731e-01
7.  Website: 7  Importance 1.042549e-01
8.  Website: 8  Importance 5.693352e-02
9.  Website: 9  Importance 3.955771e-02
10. Website: 12 Importance 2.218189e-02
11. Website: 11 Importance 2.218189e-02
12. Website: 10 Importance 2.218189e-02
```

u = ⅙*[0 … 0 1 1 1 1 1 1]T

```
alpha = 5.000000e-01
Time taken: 4.010000e-05
# of iterations(repeats for x=Cx): 46
Websites by importance (most to least):
1.  Website: 8  Importance 1.957384e-01
2.  Website: 9  Importance 1.814546e-01
3.  Website: 12 Importance 1.671708e-01
4.  Website: 11 Importance 1.671708e-01
5.  Website: 10 Importance 1.671708e-01
6.  Website: 7  Importance 8.570276e-02
7.  Website: 3  Importance 2.167776e-02
8.  Website: 2  Importance 3.629764e-03
9.  Website: 4  Importance 3.024803e-03
10. Website: 6  Importance 2.419843e-03
11. Website: 5  Importance 2.419843e-03
12. Website: 1  Importance 2.419843e-03
```

Observations:

- Different u has slightly but negligible impact on the number of iterations run.
- Alpha is smaller now, and we see that the cycles {8,9} and {10,11,12} no longer dominate. Now we see u is able to influence the outcome of the most important node more. For example, the u in the top left quadrant favors website 1, and that is reflected since website 1 takes the highest importance in the top left quadrant. Similarly, in the bottom left quadrant, since website 1-6 were favored by u, we see them take the top 6 spots in importance. The ordering within the top 6 for 1-6 is then dependent on what matrix C was

with alpha = 0.05

u = [1 0 0 … 0]T

```
alpha = 5.000000e-02
Time taken: 1.772000e-04
# of iterations(repeats for x=Cx): 11
Websites by importance (most to least):
1.  Website: 1  Importance 9.502017e-01
2.  Website: 2  Importance 4.771177e-02
3.  Website: 8  Importance 2.155357e-04
4.  Website: 7  Importance 2.150183e-04
5.  Website: 9  Importance 2.104162e-04
6.  Website: 12 Importance 2.101468e-04
7.  Website: 11 Importance 2.101468e-04
8.  Website: 10 Importance 2.101468e-04
9.  Website: 4  Importance 2.067318e-04
10. Website: 3  Importance 2.050149e-04
11. Website: 6  Importance 2.016896e-04
12. Website: 5  Importance 2.016896e-04
```

u = ½*[1 1 0 0 … 0]T

```
alpha = 5.000000e-02
Time taken: 3.880000e-05
# of iterations(repeats for x=Cx): 11
Websites by importance (most to least):
1.  Website: 2  Importance 5.009736e-01
2.  Website: 1  Importance 4.771177e-01
3.  Website: 8  Importance 2.263125e-03
4.  Website: 7  Importance 2.257692e-03
5.  Website: 9  Importance 2.209370e-03
6.  Website: 12 Importance 2.206541e-03
7.  Website: 11 Importance 2.206541e-03
8.  Website: 10 Importance 2.206541e-03
9.  Website: 4  Importance 2.170684e-03
10. Website: 3  Importance 2.152656e-03
11. Website: 6  Importance 2.117741e-03
12. Website: 5  Importance 2.117741e-03
```

u = ⅙*[1 1 1 1 1 1 0 0 0 0 0 0]T

```
alpha = 5.000000e-02
Time taken: 1.910000e-05
# of iterations(repeats for x=Cx): 11
Websites by importance (most to least):
1.  Website: 2  Importance 1.693773e-01
2.  Website: 4  Importance 1.653445e-01
3.  Website: 6  Importance 1.613117e-01
4.  Website: 5  Importance 1.613117e-01
5.  Website: 1  Importance 1.613117e-01
6.  Website: 3  Importance 1.600532e-01
7.  Website: 7  Importance 1.367789e-02
8.  Website: 8  Importance 1.793195e-03
9.  Website: 9  Importance 1.467531e-03
10. Website: 12 Importance 1.450391e-03
11. Website: 11 Importance 1.450391e-03
12. Website: 10 Importance 1.450391e-03
```

u = ⅙*[0 … 0 1 1 1 1 1 1]T

```
alpha = 5.000000e-02
Time taken: 2.920000e-05
# of iterations(repeats for x=Cx): 11
Websites by importance (most to least):
1.  Website: 8  Importance 1.706354e-01
2.  Website: 9  Importance 1.668654e-01
3.  Website: 12 Importance 1.666670e-01
4.  Website: 10 Importance 1.666670e-01
5.  Website: 11 Importance 1.666670e-01
6.  Website: 7  Importance 1.583367e-01
7.  Website: 3  Importance 3.958759e-03
8.  Website: 2  Importance 4.192508e-05
9.  Website: 4  Importance 4.092686e-05
10. Website: 6  Importance 3.992865e-05
11. Website: 5  Importance 3.992865e-05
12. Website: 1  Importance 3.992865e-05
```

Observations:

- Different u has the same number of iterations run, which means that u truly has negligible impact on the number of iterations and thus on the rate of convergence.
- The magnitude of importance is getting smoothed out as u becomes more distributed. Take the bottom two quadrants and their u's, the websites prioritized by the u's have less variation between their importance as compared to when alpha = 0.95 or 0.5.

with alpha = 0

u = [1 0 0 … 0]T

```
alpha = 0
Time taken: 1.487000e-04
# of iterations(repeats for x=Cx): 1
Websites by importance (most to least):
1.  Website: 1  Importance 1
2.  Website: 12 Importance 0
3.  Website: 11 Importance 0
4.  Website: 10 Importance 0
5.  Website: 9  Importance 0
6.  Website: 8  Importance 0
7.  Website: 7  Importance 0
8.  Website: 6  Importance 0
9.  Website: 5  Importance 0
10. Website: 4  Importance 0
11. Website: 3  Importance 0
12. Website: 2  Importance 0
```

u = ½*[1 1 0 0 … 0]T

```
alpha = 0
Time taken: 2.430000e-05
# of iterations(repeats for x=Cx): 1
Websites by importance (most to least):
1.  Website: 2  Importance 5.000000e-01
2.  Website: 1  Importance 5.000000e-01
3.  Website: 12 Importance 0
4.  Website: 11 Importance 0
5.  Website: 10 Importance 0
6.  Website: 9  Importance 0
7.  Website: 8  Importance 0
8.  Website: 7  Importance 0
9.  Website: 6  Importance 0
10. Website: 5  Importance 0
11. Website: 4  Importance 0
12. Website: 3  Importance 0
```

u = ⅙*[1 1 1 1 1 1 0 0 0 0 0 0]T

```
alpha = 0
Time taken: 1.390000e-05
# of iterations(repeats for x=Cx): 1
Websites by importance (most to least):
1.  Website: 6  Importance 1.666667e-01
2.  Website: 5  Importance 1.666667e-01
3.  Website: 4  Importance 1.666667e-01
4.  Website: 3  Importance 1.666667e-01
5.  Website: 2  Importance 1.666667e-01
6.  Website: 1  Importance 1.666667e-01
7.  Website: 12 Importance 0
8.  Website: 11 Importance 0
9.  Website: 10 Importance 0
10. Website: 9  Importance 0
11. Website: 8  Importance 0
12. Website: 7  Importance 0
```

u = ⅙*[0 … 0 1 1 1 1 1 1]T

```
alpha = 0
Time taken: 3.080000e-05
# of iterations(repeats for x=Cx): 1
Websites by importance (most to least):
1.  Website: 12 Importance 1.666667e-01
2.  Website: 11 Importance 1.666667e-01
3.  Website: 10 Importance 1.666667e-01
4.  Website: 9  Importance 1.666667e-01
5.  Website: 8  Importance 1.666667e-01
6.  Website: 7  Importance 1.666667e-01
7.  Website: 6  Importance 0
8.  Website: 5  Importance 0
9.  Website: 4  Importance 0
10. Website: 3  Importance 0
11. Website: 2  Importance 0
12. Website: 1  Importance 0
```

Observations:

- When alpha = 0, the u*eT is effectively the link matrix, so only the websites that have value are those favored by matrix u. This is reflected in the importance, where only websites with value in u will have non-zero importance, whereas those that have 0 in u will subsequently have 0 importance.

Thus, we notice that u has negligible impact on the rate of convergence, but does play an impact on importance. u only plays a significant role in impacting importance when alpha is low, allowing the u*eT to dominate its influence on the link matrix.

(e) Try running your page rank file with the matrix A. Explain your results. Does it matter what initial distribution you start with?

When running page rank with matrix A, we notice that over many iterations all websites that don't form cycles converge to 0, whereas those with cycles stabilize and rotate the importance value of their cycles indefinitely. If each website/node in the cycle has the same importance the algorithm converges, otherwise it repeats infinitely as shown below. In the layout below, the top number refers to the iteration number and the importance column vector after running pagerank on A is not sorted so the row number corresponds to the website number. If we add up the total importance, the total should sum to less than 1 - an indication that the matrix comprises zero column vectors - thus there is a "loss" of importance whenever the graph traverses to the websites with zero links.

In order to avoid a loss of importance while making the website a "sink" website with no outgoing nodes, one strategy is to have the website have an outgoing node to itself, i.e. self-looping, by having a column of zeros with the diagonal element of the column with respect to the nxn matrix A having value 1 - this means the website has an outgoing link to itself.

429365	429366	429367	429368
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0.1757	0.1681	0.1757	0.1681
0.1681	0.1757	0.1681	0.1757
0.0833	0.0833	0.0833	0.0833
0.0833	0.0833	0.0833	0.0833
0.0833	0.0833	0.0833	0.0833

(f) Try running your page rank file with the matrix B. Explain your results. Does it matter what initial distribution you start with?

When running page rank with matrix B, the program runs without terminating. B is a stochastic matrix with cycle {10,11,12}, so the importance shifts within the cycles as permutations of each other, meaning that the value of x when applying matrix B never converges unless the stabilized importance of each website in the cycle is the same, similar to our observation for part (e), causing the program to run indefinitely. In the layout below, the top number refers to the iteration number and the importance column vector after running pagerank on B is not sorted so the row number corresponds to the website number. Notice here that the sum of the set of websites' importance still adds up to 1, unlike in the case of matrix A.

150932	150933	150934	150935
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0.2859	0.2930	0.2859	0.2930
0.2930	0.2859	0.2930	0.2859
0.1404	0.1404	0.1404	0.1404
0.1404	0.1404	0.1404	0.1404
0.1404	0.1404	0.1404	0.1404

(3) Posted on blackboard under Content is a matlab file that contains the incidence matrix M of a small network of 1000 webpages. In other words $M_{ij} = 1$ if there is a link on website j to website i, otherwise it is 0. Load this matrix into your workspace in Matlab.

(a) Create the link matrix A from M and then modify A to create B and then $C_{u,\alpha}$. Run your Page Rank m-file to rank the pages in order. Use $\alpha = 0.95$ and $u = (1/1000)[1, 1, \ldots, 1]^T$. List the top 10 pages with their corresponding importances in order.

```
Websites by importance (most to least):
1.  Website: 467    Importance: 2.933673e-02
2.  Website: 630    Importance: 2.481727e-02
3.  Website: 344    Importance: 2.443105e-02
4.  Website: 545    Importance: 2.341765e-02
5.  Website: 418    Importance: 2.128381e-02
6.  Website: 683    Importance: 7.832251e-03
7.  Website: 184    Importance: 7.467044e-03
8.  Website: 608    Importance: 7.051288e-03
9.  Website: 500    Importance: 6.523839e-03
10. Website: 914    Importance: 6.473814e-03
```

(b) *Fun part!* Create one or more new websites numbered 1001, 1002, etc to add to the network of websites in (3). Your goal is to get website number 1001 into the top 10% of all websites spending the least amount of money. The creation of any website costs \$1000. In addition, you can pay other websites to link to any one of your webpages. The cost of this depends on the ranking of the website you are paying. If that website has rank i, then the cost is $(1000 - i + 1)^2$ dollars. You can add whatever links you want to your own websites for free. Show the cheapest solution you found to get your website into the top 10%, the cost associated with your method, and explain how you decided to do what you did.

We had a lot of initial approaches to this problem, and wanted to narrow down the possible avenues to explore. The first thing we realized is that adding a link from a website to our website is O(n^2) in cost, and it quickly becomes impractical to add such a link to our site, as it is much too costly. So we first decided to explore how many new web pages we would need that link solely to website 1001 to determine when it would enter the top 10%, and work from there, as these had a constant cost of only 1000$.

We found that this number was 72 new websites, or 72,000 dollars. We eliminated the possibility of having these websites point to anything other than 1001, because this would greatly diminish the probability of quickly reaching 1001 based on how the algorithm works. Now the question was, how would we get our cost to be less than 72,000?

Our next order of business was to try to replace a new website with links to already existing websites, and have these be more cost-effective. Of course, anything earlier than about rank 970 would already be over 1000$, and through some simple checking, we determined the value they added wasn't worth the cost despite being higher ranked than a brand new website. We then started with adding links from the lowest websites.

We also realized that adding links for the lowest cost options to our new websites all pointing to 1001 would also be beneficial in that it raises the probability on a given website of reaching 1001 eventually and more quickly, so we experimented not only with adding links to 1001, but also to our other new websites.

Through a tedious systematic approach of replacing pages with various links, we got down to a cost of \$70,791 which is marginally better than our previous value of \$72,000.

In order to maximize the effect of the webpages, we thought to try to maximize the effect of each additional connection made, be it from an existing or new webpage. Realizing the tediousness of continuing on with guessing and checking, we endeavored to create a greedy algorithm to try and find a better solution for us. In developing a greedy algorithm, we sought to test out multiple methods of optimizing each step cost.

The first method was to add the least expensive (lowest ranked) webpages first, then add new web pages until the desired rank was achieved. This was meant to ensure that the least costly step was chosen at each interval. This method was not optimal, however, and resulted in a total cost of $80,416. This is because previous methods relied on analysis of the type of link and the effect of that link on future links, whereas this method considered only pure cost.

After testing out other metrics to little avail (these methods are all similar to the method described above, with added thresholds or stopping criterion when the price of adding a new connection becomes too high - you can see these methods in q3b_saket and bestattempt70791.mat).

Eventually, the greedy algorithm metric we found to work best was the rank gained to cost ratio of each possible step. To find the rank-gained-to-cost ratio, the cost of adding a connection between each unvisited webpage and 1001 was calculated, as well as the potential rank gain. The same analysis was performed on creating a new webpage. We connected all webpages to 1001 only in order to limit cycles (maintain more real circumstances) and to avoid splitting the probability of staying on a webpage between the many webpages we created, as well as to reduce computational load.

At each step, after the rank-gained-to-cost ratio was calculated, the action with the highest ratio was chosen. This ensured that the money was being spent to maximize the rank, as opposed to other methods that either maximized rank or minimized the number of expensive operations. Since each action was optimized by ‘reward’ for money spent, this algorithm resulted in the best performance at $68,806 for a final rank of 107, still only slightly lower than previous efforts. This method was very computationally heavy, having a runtime of O(alpha*N^3). The increase in computational complexity and the relatively small decrease in cost demonstrates the difficulty in further optimizing this problem.

Initially, we planned on taking a shortest-path approach, but quickly found we lacked the space and time required to implement an algorithm like Dijkstra’s due to the possibility of adding a webpage at each step (combined with the already large size of the matrix). In the future, given more computational power this solution may yield even better results. We had avoided doing an algorithm like this because of the huge computational complexity involved. Further areas of exploration may involve converting this problem into a graph problem to apply different styles of walks and paths in order to find more naturalistic ways of increasing the rank of a webpage.

Greedy Algorithm (best approach discovered):

```
% Greedy Algorithm - runs in $68,806, rank 107
% Ranks each possible move by rank gained over cost
% Chooses most efficient move based on highest rank gained over cost
cost = 0;
while getRank(1001, rnkDes) > 0.1*length(M) %Stopping condition - rank has been met
    greatest_step = -1; %Initializing best rank-to-cost ratio to -1
    greatest_step_index = -1; %Initializing index of ebst cost ratio to -1
    currRank = getRank(1001, rnkDes); %Current rank of 1001
    for i = 1:length(visited) % Calculates the best possible unvisited webpage to add
        a connection to
        currRank = getRank(1001, rnkDes);
        if visited(i) == 0
            A = M;
            A = addConnection(1001, i, A);
            rnkDes = getCMatrix(A);
            theoretical_cost = (1000 - i + 1)^2;
            rank_cost_ratio = (getRank(1001, rnkDes) - currRank)/theoretical_cost;
            if rank_cost_ratio > greatest_step
                greatest_step = rank_cost_ratio;
                greatest_step_index = i;
            end
        end
    end

    % Calculates rank-to-cost ratio of adding a new webpage
    A = M;
    A = addPage(A);
    A = addConnection(1001, length(A), A);
    rnkDes = getCMatrix(A);
    theoretical_cost = 1000;
    rank_cost_ratio = (getRank(1001, rnkDes) - currRank)/theoretical_cost;

    % Chooses the best rank-to-cost ratio step
    if rank_cost_ratio > greatest_step
        M = addPage(M);
        M = addConnection(1001, length(M), M);
        visited(length(M)) = 1;
        cost = cost + 1000;
        rnkDes = getCMatrix(M);
    else
        visited(greatest_step_index) = 1;
        M = addConnection(1001, greatest_step_index, M);
        rnkDes = getCMatrix(M);
        cost = cost + (1000 - greatest_step_index + 1)^2;
    end
end
print(cost)
```