HW2 - Page Rank

Computational linear algebra for large scale problems

Arash Daneshvar

s314415

Politecnico di Torino

2022-2023

Solutions to Exercises

Exercise 1

Suppose the people who own page 3 in the web of Figure 1 are infuriated by the fact that its importance score, computed using formula (2.1), is lower than the score of page 1. In an attempt to boost page 3's score, they create a page 5 that links to page 3; page 3 also links to page 5. Does this boost page 3's score above that of page 1?

Answer: The real part of the eigenvector for matrix A in Figure 1 is:

$$[0.3871, 0.1290, \mathbf{0.2903}, 0.1935]$$

To determine whether creating a page 5 that links to page 3 (and having page 3 also link to page 5) boosts page 3's score above that of page 1, we need to calculate the updated matrix A.

$$A = \begin{pmatrix} 0 & 0 & 1/2 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 & 1 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 \end{pmatrix}$$

Now I need to find $A\mathbf{x} = \mathbf{x}$. The value for $\mathbf{x}$ is: Real part of the eigenvector:

$$[0.2448, 0.0816, \mathbf{0.3673}, 0.1224, 0.1836]$$

As shown, the score for page 3 has increased, boosting its rank above that of page 1.

At the following, you can see the python code of calculate eigenvector of matrix A.

```
>>> import numpy as np

>>> A = np.array([
    [0  , 0  , 1/2, 1/2, 0],
    [1/3, 0  ,   0,   0, 0],
    [1/3, 1/2,   0, 1/2, 1],
    [1/3, 1/2,   0,   0, 0],
    [0  ,   0, 1/2,   0, 0]])
```

```
>>> eigenvalues, eigenvectors = np.linalg.eig(A)
>>> index = np.where(np.isclose(eigenvalues, 1))[0][0]
>>> eigenvector = eigenvectors[:, index]
>>> eigenvector_normalized = eigenvector / np.sum(eigenvector)
>>> eigenvector_real = np.real(eigenvector_normalized)
>>> print("Real part of the eigenvector:")
>>> print(eigenvector_real)
Real part of the eigenvector:
[0.24489796 0.08163265 0.36734694 0.12244898 0.18367347]
```

Exercise 2

Construct a web consisting of three or more subwebs and verify that dim(V1(A)) equals (or exceeds) the number of the components in the web.

Answer: To clarify I can see the three subweb example:

Given the matrix A representing a web of vectors:

$$A = \begin{pmatrix} 0 & 0 & 1 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To verify that $\dim(V_1(A))$ equals or exceeds the number of components (subwebs) in the web, I proceed as follows:

Calculate the Rank of A:
The rank of a matrix A is the dimension of the vector space spanned by its columns, denoted as $\dim(V_1(A))$. Rank martix A is 4.

Compare the Rank with the Number of Components:
The number of components (subwebs) in the web be $k = 3$. The rank of A, which is 4, is compared with k.

Since $\dim(V_1(A)) = 5$ and $4 \geq 3$, it follows that $\dim(V_1(A))$ equals or exceeds the number of components in the web.

Thus, the condition $\dim(V_1(A)) \geq k$ is satisfied, verifying that the dimension of the vector space spanned by the columns of A meets or surpasses the number of components in the web.

The Python code is as follows:

```
>>> import numpy as np

>>> A = np.array([
    [0  , 0  , 1, 1/2, 0, 0],
    [1/3, 0  , 0,   0, 0, 0],
    [1/3, 1/2, 0, 1/2, 0, 0],
```

```
    [1/3, 1/2, 0,   0, 0, 0],
    [0  , 0  , 0,   0, 0, 0],
    [0  , 0  , 0,   0, 0, 0]])

>>> rank_A = np.linalg.matrix_rank(A)
>>> num_components = 3
>>> print(f"Matrix A:\n{A}")
>>> print(f"Rank of A: {rank_A}")
>>> print(f"Number of components (subwebs): {num_components}")
>>> print(f"dim(V1(A)) {'equals' if rank_A == num_components else
'exceeds' if rank_A > num_components else 'is less than'}
the number of components in the web.")

Matrix A:
[[0.         0.         1.         0.5        0.         0.]
 [0.33333333 0.         0.         0.         0.         0.]
 [0.33333333 0.5        0.         0.5        0.         0.]
 [0.33333333 0.5        0.         0.         0.         0.]
 [0.         0.         0.         0.         0.         0.]
 [0.         0.         0.         0.         0.         0.]]
Rank of A: 4
Number of components (subwebs): 3
dim(V1(A)) exceeds the number of components in the web.
```

Exercise 3

Add a link from page 5 to page 1 in the web of Figure 2. The resulting web, considered as an undirected graph, is connected. What is the dimension of V1(A)?

Answer: After Add a link from page 5 to page 1 in the web of Figure 2. the matrix A would be linke the below.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \frac{1}{3} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{1}{3} \\ 0 & 0 & 1 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

As demonstrated in the Python code below, the dimension of $V_1(A)$, the eigenspace corresponding to the eigenvalue 1, is equal to 2.

The Python code is as follows:

```
>>> import numpy as np

>>> A = np.array([
    [0, 1, 0, 0, 1/3],
    [1, 0, 0, 0, 0],
    [0, 0, 0, 1, 1/3],
    [0, 0, 1, 0, 1/3],
    [0, 0, 0, 0, 0]
])

>>> eigenvalues, eigenvectors = np.linalg.eig(A)


>>> print(f"Eigenvalues of A: {eigenvalues}")
>>> print(f"Eigenvectors of A:\n{eigenvectors}")
>>> eigenvalue_1_index = np.where(np.isclose(eigenvalues, 1))[0]
>>> eigenvectors_1 = eigenvectors[:, eigenvalue_1_index]
>>> print(f"Eigenvectors corresponding to eigenvalue 1:\n
{eigenvectors_1}")
>>> dimension_V1_A = eigenvectors_1.shape[1]
```

```
>>> print(f"Dimension of V1(A): {dimension_V1_A}")

Eigenvalues of A: [ 1. -1.  1. -1.  0.]
Eigenvectors of A:
[[ 0.70710678 -0.70710678  0.          0.          0.        ]
 [ 0.70710678  0.70710678  0.          0.         -0.28867513]
 [ 0.         -0.          0.70710678 -0.70710678 -0.28867513]
 [ 0.         -0.          0.70710678  0.70710678 -0.28867513]
 [ 0.          0.          0.          0.          0.8660254 ]]
Eigenvectors corresponding to eigenvalue 1:
[[0.70710678 0.        ]
 [0.70710678 0.        ]
 [0.         0.70710678]
 [0.         0.70710678]
 [0.         0.        ]]
Dimension of V1(A): 2
```

Exercise 4

In the web of Figure 2.1, remove the link from page 3 to page 1. In the resulting web page 3 is now a dangling node. Set up the corresponding substochastic matrix and find its largest positive (Perron) eigenvalue. Find a non-negative Perron eigenvector for this eigenvalue, and scale the vector so that components sum to one. Does the resulting ranking seem reasonable?

Answer:

$$A\_prime = \begin{pmatrix} 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

The result is as follows:

Perron Eigenvector: $[0.20664504, 0.12270648, 0.43864676, 0.23200172]$

The results are reasonable because page 3, which has no backlink, does not contribute to other pages but receives significant importance from its incoming links. This reflects the fact that despite having no outbound links, page 3 is still valued by the pages linking to it.

The Python code is as follows:

```
>>> import numpy as np

>>> A_prime = np.array([[0, 0, 0, 1/2],
              [1/3, 0, 0, 0],
              [1/3, 1/2, 0, 1/2],
              [1/3, 1/2, 0, 0]])

>>> eigenvalues, eigenvectors = np.linalg.eig(A_prime)
>>> perron_index = np.argmax(eigenvalues.real)
>>> perron_eigenvalue = eigenvalues[perron_index].real
>>> perron_eigenvector = eigenvectors[:, perron_index].real
>>> perron_eigenvector = np.abs(perron_eigenvector)
>>> scaled_perron_eigenvector =
perron_eigenvector / np.sum(perron_eigenvector)
>>> print(f"Perron eigenvalue: {perron_eigenvalue}")
```

```
>>> print(f"Non-negative Perron eigenvector:\n
{perron_eigenvector}")
>>> print(f"Scaled Perron eigenvector:\n
{scaled_perron_eigenvector}")

Perron eigenvalue: 0.5613532393351085
Non-negative Perron eigenvector:
[0.37479335 0.22255348 0.79557628 0.42078293]
Scaled Perron eigenvector:
[0.20664504 0.12270648 0.43864676 0.23200172]
```

Exercise 5

Prove that in any web the importance score of a page with no backlinks is zero.

Answer:
In the formula 2.1 in the paragraph article, $L_k \subset \{1, 2, \ldots, n\}$ denotes the set of pages with a link to page k, that is, L_k is the set of page k's backlinks. Since the summation is over an empty set, it evaluates to zero. Thus, the importance score of a page with no backlinks is indeed zero

Exercise 6

Exercise 7

Prove that if A is an $n \times n$ column-stochastic matrix and $0 \leq m \leq 1$, then $M = (1-m)A + mS$ is also a column-stochastic matrix.

Answer:
To prove that if A is an $n \times n$ column-stochastic matrix and $0 \leq m \leq 1$, then $M = (1-m)A + mS$ is also a column-stochastic matrix, where S is an $n \times n$ matrix with all entries $\frac{1}{n}$, we need to show two things:

1. The columns of M sum up to 1.
2. All elements of M are non-negative.

Columns sum up to 1:

Let's denote the columns of A as $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$, and the columns of S as $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n$. Since A is a column-stochastic matrix, the sum of elements in each column of A is 1. Similarly, since S is a matrix of all $\frac{1}{n}$, the sum of elements in each column of S is $n \cdot \frac{1}{n} = 1$. Now, the j-th column of M is:

$$\mathbf{m}_j = (1-m)\mathbf{a}_j + m\mathbf{s}_j$$

The sum of elements in $\mathbf{m}_j$ is:

$$\sum_{i=1}^{n} m_{ij} = (1-m)\sum_{i=1}^{n} a_{ij} + m\sum_{i=1}^{n} s_{ij} = (1-m)\cdot 1 + m \cdot 1 = 1$$

Therefore, the sum of elements in each column of M is 1, making M column-stochastic.

Non-negativity of elements:

Since A is a column-stochastic matrix, all elements of A are non-negative. Similarly, all elements of S are $\frac{1}{n}$, which is also non-negative. Since m is non-negative, the combination $(1-m)A + mS$ results in non-negative elements. Therefore, all elements of M are non-negative.

Thus, combining both points, we conclude that M is a column-stochastic matrix. This completes the proof.

Exercise 8

Show that the product of two column-stochastic matrices is also column-stochastic.

Answer:
To prove that the product of two column-stochastic matrices is also column-stochastic, let's consider two column-stochastic matrices A and B.

A matrix is column-stochastic if the sum of elements in each column is equal to 1, and all elements are non-negative.

Let $C = AB$ be the product of matrices A and B.

We need to show two things:

1. The columns of C sum up to 1.
2. All elements of C are non-negative.

1) Columns sum up to 1:
Let c_{ij} denote the element at the i-th row and j-th column of C. Since $C = AB$, we have:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

where n is the number of columns in A (equal to the number of rows in B). Now, consider the sum of elements in the j-th column of C:

$$\sum_{i=1}^{n} c_{ij} = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} a_{ik} \cdot b_{kj} \right)$$

Since A is column-stochastic, the sum of elements in each column of A is 1. Therefore, the sum above becomes:

$$\sum_{i=1}^{n} c_{ij} = \sum_{k=1}^{n} b_{kj}$$

Since B is also column-stochastic, the sum of elements in each column of B is 1. Hence, the sum of elements in the j-th column of C is also 1. Therefore, the columns of C sum up to 1.

2) Non-negativity of elements:

Each element c_{ij} of C is a sum of products of non-negative elements from A and B, which are column-stochastic matrices. Therefore, all elements of C are non-negative.
Thus, combining both points, we conclude that C is a column-stochastic matrix. This completes the proof.

The Python code for a sample product of two column-stochastic matrices is also column-stochastic is as follows:

```
>>> import numpy as np

>>> A = np.array([
    [0.5, 0.3, 0.2],
    [0.5, 0.7, 0.8]
])

>>> B = np.array([
    [0.4, 0.6],
    [0.4, 0.3],
    [0.2, 0.1]
])

>>> C = np.dot(A, B)
>>> print("Matrix A:", A)
>>> print("\nMatrix B:", B)
>>> print("\nProduct Matrix C = AB:", C)
>>> non_negative = np.all(C >= 0)
>>> print(f"\nAll entries non-negative: {non_negative}")
>>> column_sums = np.sum(C, axis=0)
>>> print(f"Column sums of C: {column_sums}")
>>> is_column_stochastic = np.allclose(column_sums, 1)
>>> print(f"C is column-stochastic: {is_column_stochastic}")

Matrix A:
[[0.5 0.3 0.2]
 [0.5 0.7 0.8]]

Matrix B:
[[0.4 0.6]
 [0.4 0.3]
```

```
 [0.2 0.1]]

Product Matrix C = AB:
[[0.36 0.41]
 [0.64 0.59]]

All entries non-negative: True
Column sums of C: [1. 1.]
C is column-stochastic: True
```

Exercise 9

Show that a page with no backlinks is given importance score mn by formula (3.2).

Answer:
To prove that a page with no backlinks is given an importance score of $\frac{m}{n}$ by formula (3.2), I will use the framework of the PageRank algorithm.

$$r_k = (1 - m) \sum_{j \in L_k} \frac{r_j}{n_j} + (m)\frac{1}{n}$$

Page with No Backlinks:

- Consider a page k with no backlinks. This implies L_k is empty, meaning no other pages link to page k.
- Therefore, the summation term in formula (3.2) that accounts for backlinks contributions is zero because L_k is empty. There are no pages j in the set L_k, so the sum over an empty set is zero.

Substitute into Formula (3.2):

- Given that L_k is empty, the summation term becomes zero:This simplifies to:

$$r_k = (1 - m) \cdot 0 + (m)\frac{1}{n}$$

$$r_k = (m)\frac{1}{n}$$

Thus, a page with no backlinks is given an importance score of $\frac{m}{n}$ by formula (3.2).

Exercise 10

Exercise 11

Consider again the web in Figure 2.1, with the addition of a page 5 that links to page 3, where page 3 also links to page 5. Calculate the new ranking by finding the eigenvector of M (corresponding to $\lambda = 1$) that has positive components summing to one. Use $m = 0.15$.

Answer:
The real part of the eigenvector for matrix A in Figure 2.1 with the addition of a page 5 that links to page 3, where page 3 also links to page is:

$$[0.23710.09710.34880.13840.1782]$$

The ranking is the same as compute in ecercise 1

At the following, you can see the python code of calculate eigenvector of matrix A.

```
>>> import numpy as np

>>> A = np.array([
    [0  , 0  , 1/2, 1/2, 0],
    [1/3, 0  ,   0,   0, 0],
    [1/3, 1/2,   0, 1/2, 1],
    [1/3, 1/2,   0,   0, 0],
    [0  ,   0, 1/2,   0, 0]])

>>> n = 5
>>> S = np.ones((n, n)) / n
>>> m = 0.15
>>> M = m * S + (1 - m) * A
>>> eigenvalues, eigenvectors = np.linalg.eig(M)
>>> index = np.where(np.isclose(eigenvalues, 1))[0][0]
>>> eigenvector = eigenvectors[:, index]
>>> eigenvector_normalized = eigenvector / np.sum(eigenvector)
>>> eigenvector_real = np.real(eigenvector_normalized)
>>> print("Real part of the eigenvector:")
>>> print(eigenvector_real)

Real part of the eigenvector:
[0.23714058 0.09718983 0.34889409 0.13849551 0.17827999]
```

Exercise 12

Add a sixth page that links to every page of the web in the previous exercise, but to which no other page links. Rank the pages using A, then using M with $m = 0.15$, and compare the results.

Answer:

The following table shows the pages sorted by their importance scores in descending order:

```
{
    'Page 3': 0.34017174
    'Page 1': 0.23121207
    'Page 5': 0.17382299
    'Page 4': 0.13503312
    'Page 2': 0.09476009
    'Page 6': 0.025
}
```

At the following, you can see the python code of calculate eigenvector of matrix A.

```
>>> import numpy as np

>>> A = np.array([
    [0  , 0  , 1/2, 1/2, 0, 1/5],
    [1/3, 0  ,   0,   0, 0, 1/5],
    [1/3, 1/2,   0, 1/2, 1, 1/5],
    [1/3, 1/2,   0,   0, 0, 1/5],
    [0  ,   0, 1/2,   0, 0, 1/5],
    [0  ,   0,   0,   0, 0,   0]])

>>> n = 6
>>> S = np.ones((n, n)) / n
>>> m = 0.15
>>> M = m * S + (1 - m) * A
>>> eigenvalues, eigenvectors = np.linalg.eig(M)
>>> index = np.where(np.isclose(eigenvalues, 1))[0][0]
>>> eigenvector = eigenvectors[:, index]
>>> eigenvector_normalized = eigenvector / np.sum(eigenvector)
```

```
>>> eigenvector_real = np.real(eigenvector_normalized)
>>> print("Real part of the eigenvector:")
>>> print(eigenvector_real)

Real part of the eigenvector:
[0.23121207 0.09476009 0.34017174 0.13503312 0.17382299 0.025]
```

At the following, you can see the python code of calculate ranking.

```
>>> values = [0.23121207, 0.09476009, 0.34017174,
0.13503312, 0.17382299, 0.025]
>>> page_names = ['Page 1', 'Page 2', 'Page 3',
'Page 4', 'Page 5', 'Page 6']
>>> page_ranking = {page_names[i]: values[i]
for i in range(len(values))}
>>> sorted_page_ranking = dict(sorted(page_ranking.items(),
key=lambda item: item[1], reverse=True))
>>> print(sorted_page_ranking)

{'Page 3': 0.34017174,
'Page 1': 0.23121207,
'Page 5': 0.17382299,
'Page 4': 0.13503312,
'Page 2': 0.09476009,
'Page 6': 0.025}
```

Exercise 13

Construct a web consisting of two or more subwebs and determine the ranking given by formula (3.1).

Answer:
Given the matrix A representing a web of vectors:

$$A = \begin{pmatrix} 0 & 0 & 1 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

The following table shows the pages sorted by their importance scores in descending order:

```
{
    'Page 1': 0.24543378
    'Page 3': 0.19197442
    'Page 5': 0.16666667
    'Page 6': 0.16666667
    'Page 4': 0.13471889
    'Page 2': 0.09453957
}
```

At the following, you can see the python code of calculate eigenvector of matrix A.

```
>>> import numpy as np

>>> A = np.array([
    [0, 0, 1, 1/2, 0, 0],
    [1/3, 0, 0, 0, 0, 0],
    [1/3, 1/2, 0, 1/2, 0, 0],
    [1/3, 1/2, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 1, 0]
])

>>> n = 6
```

```
>>> S = np.ones((n, n)) / n
>>> m = 0.15
>>> M = m * S + (1 - m) * A
>>> eigenvalues, eigenvectors = np.linalg.eig(M)
>>> index = np.where(np.isclose(eigenvalues, 1))[0][0]
>>> eigenvector = eigenvectors[:, index]
>>> eigenvector_normalized = eigenvector / np.sum(eigenvector)
>>> eigenvector_real = np.real(eigenvector_normalized)
>>> print("Real part of the eigenvector:")
>>> print(eigenvector_real)

Real part of the eigenvector:
[0.24543378 0.09453957 0.19197442 0.13471889 0.16666667 0.16666667]
```

At the following, you can see the python code of calculate ranking.

```
>>> values = [0.24543378, 0.09453957, 0.19197442,
0.13471889, 0.16666667, 0.16666667]
>>> page_names = ['Page 1', 'Page 2', 'Page 3',
'Page 4', 'Page 5', 'Page 6']
>>> page_ranking = {page_names[i]: values[i]
for i in range(len(values))}
>>> sorted_page_ranking = dict(sorted(page_ranking.items(),
key=lambda item: item[1], reverse=True))
>>> print(sorted_page_ranking)

{'Page 1': 0.24543378,
'Page 3': 0.19197442,
'Page 5': 0.16666667,
'Page 6': 0.16666667,
'Page 4': 0.13471889,
'Page 2': 0.09453957}
```

Exercise 14

For the web in Exercise 11, compute the values of $\|\mathbf{M}^k\mathbf{x}_0-\mathbf{q}\|_1$ and $\frac{\|\mathbf{M}^k\mathbf{x}_0-\mathbf{q}\|_1}{\|\mathbf{M}^{k-1}\mathbf{x}_0-\mathbf{q}\|_1}$ for $k = 1, 5, 10, 50$, using an initial guess $\mathbf{x}_0$ not too close to the actual eigenvector $\mathbf{q}$ (so that you can watch the convergence). Determine $c = \max_{1\le j\le n} |1 - 2\min_{1\le i\le n} M_{ij}|$ and the absolute value of the second largest eigenvalue of $\mathbf{M}$.

Answer:
To analyze the convergence, we need to calculate the 1-norm differences between $M^k x_0$ and q for various values of k and determine how quickly these differences converge.

Matrix M

$$M = \begin{pmatrix} 0.03 & 0.03 & 0.455 & 0.455 & 0.03 \\ 0.31333333 & 0.03 & 0.03 & 0.03 & 0.03 \\ 0.31333333 & 0.455 & 0.03 & 0.455 & 0.88 \\ 0.31333333 & 0.455 & 0.03 & 0.03 & 0.03 \\ 0.03 & 0.03 & 0.455 & 0.03 & 0.03 \end{pmatrix}$$

Initial Guess x_0

$$x_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Placeholder for Actual Eigenvector q

$$q = \begin{pmatrix} 0.20664504 \\ 0.12270648 \\ 0.43864676 \\ 0.23200172 \\ 0.0 \end{pmatrix}$$

Convergence Calculation

- Calculate $\|M^k x_0 - q\|_1$ for $k = 1, 5, 10, 50$
- Calculate $\frac{\|M^k x_0-q\|_1}{\|M^{k-1} x_0-q\|_1}$ for the same values of k
- Determine $c = \max_{1\le j\le n} |1 - 2\min_{1\le i\le n} M_{ij}|$

- Calculate the second largest eigenvalue of M

The results of the calculations are:

$\|M^k x_0 - q\|_1$ **for** $k = 1, 5, 10, 50$

- $k = 1$: 0.6039
- $k = 5$: 0.4229
- $k = 10$: 0.4179
- $k = 50$: 0.4176

$\frac{\|M^k x_0 - q\|_1}{\|M^{k-1} x_0 - q\|_1}$ **for** $k = 1, 5, 10, 50$

- $k = 1$: N/A
- $k = 5$: 1.0604
- $k = 10$: 1.0021
- $k = 50$: 1.0000

Other Calculations

- $c = \max_{1 \leq j \leq n} |1 - 2 \min_{1 \leq i \leq n} M_{ij}| = 0.94$
- The absolute value of the second largest eigenvalue of M: 0.2859

```
>>> import numpy as np
>>> M = np.array([
    [0.03, 0.03, 0.455, 0.455, 0.03],
    [0.31333333, 0.03, 0.03, 0.03, 0.03],
    [0.31333333, 0.455, 0.03, 0.455, 0.88],
    [0.31333333, 0.455, 0.03, 0.03, 0.03],
    [0.03, 0.03, 0.455, 0.03, 0.03]
])


>>> x0 = np.array([1, 0, 0, 0, 0])
>>> q = np.array([0.20664504, 0.12270648, 0.43864676, 0.23200172, 0.0])

>>> def norm_difference(M, x0, q, k):
```

```
        xk = x0
        differences = []
        ratios = []
        for i in range(k):
            xk = M @ xk
            difference = np.linalg.norm(xk - q, 1)
            differences.append(difference)
            if i > 0:
                ratios.append(differences[i] / differences[i-1])
        return differences, ratios

>>> k_values = [1, 5, 10, 50]
differences, ratios = norm_difference(M, x0, q, max(k_values))
>>> results = {k: (differences[k-1], ratios[k-2] if k > 1 else None)
for k in k_values}
>>> c = 1 - 2 * np.min(M)
>>> eigenvalues = np.linalg.eigvals(M)
second_largest_eigenvalue = sorted(eigenvalues, reverse=True)[1]

>>> print({"Norms", "Ratios"}, results)
>>> print("c", c)
>>> print("Second largest eigenvalue", second_largest_eigenvalue)

{'Norms', 'Ratios'}
{1: (0.6039169300000001, None),
5: (0.4229104212903618, 1.0603507186943075),
10: (0.41785543595143537, 1.0021216076525001),
50: (0.41755108035978783, 1.0000000009654022)}
c 0.94
Second largest eigenvalue (0.2858814853553465+0j)
```

Exercise 15

Answer:

Exercise 16

Consider the link matrix

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} \\ 1 & \frac{1}{2} & 0 \end{pmatrix}.$$

Show that $\mathbf{M} = (1-m)\mathbf{A} + m\mathbf{S}$ (all $S_{ij} = \frac{1}{3}$) is not diagonalizable for $0 \leq m < 1$.

Answer:
To analyze whether M is diagonalizable, we check if M has a full set of linearly independent eigenvectors. If it does not, then M is not diagonalizable.

Link matrix A:

$$A = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} \\ 1 & \frac{1}{2} & 0 \end{pmatrix}$$

Stochastic matrix S:

$$S = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

Combined matrix M:

$$M = (1-m)A + mS$$

The following Python code checks if M is diagonalizable for $0 \leq m < 1$.

```
>>> import numpy as np

>>> A = np.array([
    [0, 0.5, 0.5],
    [0, 0, 0.5],
    [1, 0.5, 0]
])

>>> S = np.ones((3, 3)) / 3
```

```
>>> m_values = np.linspace(0, 1, 1000, endpoint=False)
>>> M = (1 - m) * A + m * S

>>> def is_diagonalizable(matrix):
        eigenvalues, eigenvectors = np.linalg.eig(matrix)
        rank = np.linalg.matrix_rank(eigenvectors)
        return rank == matrix.shape[0]

>>> if len(non_diagonalizable_m) > 0:
        print(f"M is not diagonalizable for all values of 0 <= m < 1")
>>> else:
        print("M is diagonalizable for all values of 0 <= m < 1")

M is not diagonalizable for all values of 0 <= m < 1
```

Results:
The matrix M is not diagonalizable for $0 \leq m < 1$.

Exercise 17

How should the value of m be chosen? How does this choice affect the rankings and the computation time?

Answer:
The parameter m is typically chosen based on the balance between following links on the web and randomly jumping to any page. The original value used by Google is $m = 0.15$. This means there is an 85% chance of following a link and a 15% chance of jumping to a random page.

Effect on rankings

- A lower m value (closer to 0) makes the PageRank algorithm more sensitive to the link structure of the web. Pages with many backlinks will have higher importance.
- A higher m value (closer to 1) reduces the impact of the link structure, leading to a more uniform distribution of importance scores.

Effect on Computation Time

- The choice of m affects the convergence speed of the PageRank algorithm.
- A lower m might slow down convergence because it heavily relies on the link structure.
- A higher m tends to speed up convergence but might produce less accurate rankings.

In conclusion, the value of m is typically chosen as a compromise between accurately reflecting the link structure of the web and ensuring that the algorithm converges efficiently. The commonly used value of $m = 0.15$ has been found to provide a good balance in practice.