Page Rank

Linear Algebra and its Applications

Henry Kirveslahti

AI511 University of Southern Denmark

Nov 10 2025

Recap of Markov Chains

- A Markov chain is a stochastic process (a sequence of random variables) where the next state only depends on the current state

Recap of Markov Chains

- A Markov chain is a stochastic process (a sequence of random variables) where the next state only depends on the current state
- The important information is encoded in the transition probabilities p_{ij}, which we can represent as a $n \times n$ matrix P (so n states)

Recap of Markov Chains

- A Markov chain is a stochastic process (a sequence of random variables) where the next state only depends on the current state
- The important information is encoded in the transition probabilities p_{ij}, which we can represent as a $n \times n$ matrix P (so n states)
- The chain is ergodic, if after exactly N steps it is possible to go from any state to any other state

Recap of Markov Chains

- A Markov chain is a stochastic process (a sequence of random variables) where the next state only depends on the current state
- The important information is encoded in the transition probabilities p_{ij}, which we can represent as a $n \times n$ matrix P (so n states)
- The chain is ergodic, if after exactly N steps it is possible to go from any state to any other state
- An ergodic Markov chain has a unique stable distribution (also known as the steady-state solution), which we can find from $\lim_{n\to\infty} P^N$ or as the dominating eigenvector (the Perron-Frobenius eigenvector)

Application to web page ranking - the page rank algorithm

- Model the internet surfing as a random person clicking links on a webpage at random
- Where we go next only depends on the current website: what links there are and how many of them

Application to web page ranking - the page rank algorithm

- Model the internet surfing as a random person clicking links on a webpage at random
- Where we go next only depends on the current website: what links there are and how many of them
- This is a Markov process, but is it ergodic?

Application to web page ranking - the page rank algorithm

The following 4 slides contain important notation for the graded assignment, the matrices A, $\tilde{A}$, $\hat{A}$ and $\bar{A}$.
Given n websites and links between them,

- Define an n by n matrix A as follows:

$$A_{ij} = \begin{cases} 1, & \text{if there is a link from site } j \text{ to site } i \\ 0 & \text{otherwise} \end{cases}$$

- In graph theory, this is known as the *adjacency matrix*, because it encodes the directional links from j to i

Application to web page ranking - the page rank algorithm

The following 4 slides contain important notation for the graded assignment, the matrices A, $\tilde{A}$, $\hat{A}$ and $\bar{A}$.
Given n websites and links between them,

- Define an n by n matrix A as follows:

$$A_{ij} = \begin{cases} 1, & \text{if there is a link from site } j \text{ to site } i \\ 0 & \text{otherwise} \end{cases}$$

- In graph theory, this is known as the *adjacency matrix*, because it encodes the directional links from j to i
- This is a matrix of zeros and ones
- Note that this is a directed graph:

Application to web page ranking - the page rank algorithm

- Some of the sites may not contain links to any other site - in the parlance of Markov chains these are *absorbing states*, and we want to get rid of them

Application to web page ranking - the page rank algorithm

- Some of the sites may not contain links to any other site - in the parlance of Markov chains these are *absorbing states*, and we want to get rid of them
- We can do this by creating an edge from such sites to every other site, corresponding to the user changing the address manually by the address bar

Application to web page ranking - the page rank algorithm

- Some of the sites may not contain links to any other site - in the parlance of Markov chains these are *absorbing states*, and we want to get rid of them
- We can do this by creating an edge from such sites to every other site, corresponding to the user changing the address manually by the address bar
- We get the modified adjacency matrix, which we dub $\tilde{A}$, defined as:

$$\tilde{A}_{ij} = \begin{cases} A_{ij}, & \text{if } \sum_i (A_{ij}) > 0 \\ 1 & \text{otherwise} \end{cases}$$

Application to web page ranking - the page rank algorithm

- Some of the sites may not contain links to any other site - in the parlance of Markov chains these are *absorbing states*, and we want to get rid of them
- We can do this by creating an edge from such sites to every other site, corresponding to the user changing the address manually by the address bar
- We get the modified adjacency matrix, which we dub $\tilde{A}$, defined as:

$$\tilde{A}_{ij} = \begin{cases} A_{ij}, & \text{if } \sum_i (A_{ij}) > 0 \\ 1 & \text{otherwise} \end{cases}$$

- Expressed in plain language, we obtain $\tilde{A}$ by replacing the zero columns with columns of all ones

Another Matrix notation

- To make this into a stochastic matrix, we want to normalize each column to sum to one. This is encoded in matrix $\hat{A}$, defined as
- $$\hat{A}_{ij} = \frac{\tilde{A}_{ij}}{\sum_j \tilde{A}_{ij}}$$

Another Matrix notation

- To make this into a stochastic matrix, we want to normalize each column to sum to one. This is encoded in matrix $\hat{A}$, defined as
- $$\hat{A}_{ij} = \frac{\tilde{A}_{ij}}{\sum_j \tilde{A}_{ij}}$$
- This matrix describes a random walk following links on webpages, with the user resorting to the address bar if they get stuck
- It need not be ergodic, for example if we had 2 pairs of webpages (so 4 pages), with each pair pointing to each other

One more Matrix

- To allow the users to resort to address bar at any state, we define a new matrix $\bar{A}$, which, for an $\alpha \in [0, 1]$ is defined as
- $$\bar{A} = \alpha \hat{A} + (1 - \alpha)\frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T$$

One more Matrix

- To allow the users to resort to address bar at any state, we define a new matrix $\bar{A}$, which, for an $\alpha \in [0, 1]$ is defined as
- $$\bar{A} = \alpha \hat{A} + (1 - \alpha)\frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T$$
- The first term models the random walk from clicking links on websites

One more Matrix

- To allow the users to resort to address bar at any state, we define a new matrix $\bar{A}$, which, for an $\alpha \in [0, 1]$ is defined as
- $$\bar{A} = \alpha \hat{A} + (1 - \alpha)\frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T$$
- The first term models the random walk from clicking links on websites
- The second term is an $n \times n$ matrix of all entires $1/n$, and it models the part where the user quits clicking and uses the address bar instead

One more Matrix

- To allow the users to resort to address bar at any state, we define a new matrix $\bar{A}$, which, for an $\alpha \in [0, 1]$ is defined as
- $$\bar{A} = \alpha \hat{A} + (1 - \alpha)\frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T$$
- The first term models the random walk from clicking links on websites
- The second term is an $n \times n$ matrix of all entires $1/n$, and it models the part where the user quits clicking and uses the address bar instead
- The constant α is called the *dampening factor*, and it can be interpreted as the probability that the user keeps on clicking links
- The matrix $\bar{A}$ is called the *Google matrix*

Page Rank Vector

- The matrix
$$\bar{A} = \alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$$
is an ergodic matrix, because all its entries are positive numbers

Page Rank Vector

- The matrix
$$\bar{A} = \alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$$
is an ergodic matrix, because all its entries are positive numbers
- Then, by the Perron Frobenius theorem, it has a unique steady-state-solution π, and this vector π is known as the *Page Rank vector*

Page Rank Vector

- The matrix
$$\bar{A} = \alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$$
is an ergodic matrix, because all its entries are positive numbers
- Then, by the Perron Frobenius theorem, it has a unique steady-state-solution π, and this vector π is known as the *Page Rank vector*
- it was used by google to rank popularity of webpages, which they used in their search engine with high success
- Next we investigate how we can find this page rank vector

Finding the Page Rank Vector - Method I

- One way we can find the page rank vector is via algebaric manipulation

Finding the Page Rank Vector - Method I

- One way we can find the page rank vector is via algebaric manipulation
- Explicitly, the page rank vector π satisfies

$$\pi = \bar{A}\pi = \left(\alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\right)\pi$$

$$\pi = \alpha\hat{A}\pi + \frac{1-\alpha}{n}\mathbf{1}_n\underbrace{\mathbf{1}_n^T\pi}_{=1}$$

$$I\pi - \alpha\hat{A}\pi = \frac{1-\alpha}{n}\mathbf{1}_n$$

$$\underbrace{(I-\alpha\hat{A})}_{A}\underbrace{\pi}_{x} = \underbrace{\frac{1-\alpha}{n}\mathbf{1}_n}_{b}.$$

Finding the Page Rank Vector - Method I

- One way we can find the page rank vector is via algebaric manipulation
- Explicitly, the page rank vector π satisfies

$$\pi = \bar{A}\pi = \left(\alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\right)\pi$$

$$\pi = \alpha\hat{A}\pi + \frac{1-\alpha}{n}\mathbf{1}_n\underbrace{\mathbf{1}_n^T\pi}_{=1}$$

$$I\pi - \alpha\hat{A}\pi = \frac{1-\alpha}{n}\mathbf{1}_n$$

$$\underbrace{(I-\alpha\hat{A})}_{A}\underbrace{\pi}_{x} = \underbrace{\frac{1-\alpha}{n}\mathbf{1}_n}_{b}.$$

- This is of the form $Ax = b$, i.e. a system of linear equations, and solvable by matrix algebra (numpy)

Method II

- The second way we can solve this is as the Perron-Frobenius eigenvector:

$$\bar{A}\pi = \pi$$

Method II

- The second way we can solve this is as the Perron-Frobenius eigenvector:

$$\bar{A}\pi = \pi$$

- The vector π is the unique eigenvector corresponding to the eigenvalue 1 (which is the maximal eigenvalue)

Method II

- The second way we can solve this is as the Perron-Frobenius eigenvector:

$$\bar{A}\pi = \pi$$

- The vector π is the unique eigenvector corresponding to the eigenvalue 1 (which is the maximal eigenvalue)
- In numpy:

```
import numpy as np
A=np.array(([0.7,0.4],[0.3,0.6]))
q,v=np.linalg.eig(A)
print(v[:,0]/sum(v[:,0]))
[0.57142857 0.42857143]
# c.f. the example from last week
```

Method III

- Since we have established that the matrix $\bar{A}$ is stochastic and ergodic, we can just start anywhere, and approximate π as $\bar{A}^M x$ for M a large number

Method III

- Since we have established that the matrix $\bar{A}$ is stochastic and ergodic, we can just start anywhere, and approximate π as $\bar{A}^M x$ for M a large number
- Formally, we can defined max iterations N and tolerance ϵ, and the run the iteration with an initial guess $x_0 = (1/n, 1/n, \ldots, 1/n)$ with

$$x_{k+1} = \bar{A} x_k$$

Method III

- Since we have established that the matrix $\bar{A}$ is stochastic and ergodic, we can just start anywhere, and approximate π as $\bar{A}^M x$ for M a large number
- Formally, we can defined max iterations N and tolerance ϵ, and the run the iteration with an initial guess $x_0 = (1/n, 1/n, \ldots, 1/n)$ with

$$x_{k+1} = \bar{A} x_k$$

- Run the above for $k = 0, 1, \ldots, N$, or while $||x_{k+1} - x_k||_2 > \epsilon$, whichever comes first

Computational Considerations

- We need to find the maximal eigenvector of matrix

$$\bar{A} = \left(\alpha \hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T\right)$$

Computational Considerations

- We need to find the maximal eigenvector of matrix

$$\bar{A} = \left(\alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\right)$$

- When we constructed $\hat{A}$, we added links from the sinks to all the other sites. In reality, we should put this outside the matrix to preserve sparsity.
  We could find the optimum by iterating

$$x_{k+1} = \alpha\hat{A}x_k + (1-\alpha)\frac{1}{n}\mathbf{1}_n$$

Computational Considerations

- We need to find the maximal eigenvector of matrix

$$\bar{A} = \left(\alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\right)$$

- When we constructed $\hat{A}$, we added links from the sinks to all the other sites. In reality, we should put this outside the matrix to preserve sparsity.
  We could find the optimum by iterating

$$x_{k+1} = \alpha\hat{A}x_k + (1-\alpha)\frac{1}{n}\mathbf{1}_n$$

- For realistic applications, very important to keep the matrix $\hat{A}$ sparse

Computational Considerations

- We need to find the maximal eigenvector of matrix

$$\bar{A} = \left(\alpha\hat{A} + (1-\alpha)\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\right)$$

- When we constructed $\hat{A}$, we added links from the sinks to all the other sites. In reality, we should put this outside the matrix to preserve sparsity.
  We could find the optimum by iterating

$$x_{k+1} = \alpha\hat{A}x_k + (1-\alpha)\frac{1}{n}\mathbf{1}_n$$

- For realistic applications, very important to keep the matrix $\hat{A}$ sparse
- We did something similar with the cipher cracking last week, where we didn't explicitly construct the matrix between all the pairs of 26! permutations, but instead just neighbors

Optimized Graph Algorithms

- The python package `networkx` implements many optimized graph algorithms that we can use for more heavy computations
- This represents the matrices as dictionaries (so essentially sparse matrices)

Crash course to `networkx`

import networkx as nx

Task	Command	Notes
Initialization	DG = `nx.DiGraph()`	Initialize Directed graph DG
Add edge	`DG.add_edge('a', 'b')`	Add edge from a to b (nodes added automatically)
Add edge with weight	`DG.add_edge('a', 'c', weight=2)`	Default weight is 1
Compute Page Rank Vector	`nx.pagerank(DG, alpha=0.85)`	α is the dampening factor
Add node	`add_node()`	Add node manually
Remove node	`remove_node()`	Also removes the edges
Remove edge	`remove_edge()`	
Check node	`DG.has_node('a')`	True if node a exist
Check edge	`DG.has_edge('b', 'a')`	True if edge from b to a
List nodes	`list(DG.nodes())`	iterator inside list
List edges	`list(DG.edges())`	iterator inside list
Count nodes	`number_of_nodes()`	similarly for edges

Check the documentation for complete list

We have now covered all the content on this course!

- Any slides after this one are just interesting development around Markov theory – you are not expected to master this material
- Markov chains are very useful for both theoretical development and applications: Modeling time series, random walks, sampling, we have only scratched the surface here
- Also: Please remember to fill out the course survey on itslearning

Convergence of Markov Chains

- Recall the weather example from last time:
- $$P = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}$$

  (Note we have flipped this so that P acts on the states by multiplying from the left
- Question: How fast does this chain converge to the stable distribution?

Observation

- If $x = Px$, then necessarily $x = Px = PPx$, and hence $x = P^N x$ for any $N = 1, 2, \ldots$

Observation

- If $x = Px$, then necessarily $x = Px = PPx$, and hence $x = P^N x$ for any $N = 1, 2, \ldots$
- If we think of the sequnce $x, Px, P^2x, P^3x \ldots$, we know that this sequence converges to the stable distribution π

Observation

- If $x = Px$, then necessarily $x = Px = PPx$, and hence $x = P^N x$ for any $N = 1, 2, \ldots$
- If we think of the sequnce $x, Px, P^2x, P^3x \ldots$, we know that this sequence converges to the stable distribution π
- On the other hand, we also have that the sequence $x, P^2x, (P^2)^2x, (P^2)^3x, \ldots$ necessarily converges to the same distribution π, because it is a subsequence of the sequence above

Observation

- If $x = Px$, then necessarily $x = Px = PPx$, and hence $x = P^N x$ for any $N = 1, 2, \ldots$
- If we think of the sequnce $x, Px, P^2x, P^3x \ldots$, we know that this sequence converges to the stable distribution π
- On the other hand, we also have that the sequence $x, P^2x, (P^2)^2x, (P^2)^3x, \ldots$ necessarily converges to the same distribution π, because it is a subsequence of the sequence above
- Similarly, we could look at any k-step sequence $(P^k)^i$, $i = 0, 1, 2, \ldots$, and observe that the higher the k, the faster it converges.

Observation

- If $x = Px$, then necessarily $x = Px = PPx$, and hence $x = P^N x$ for any $N = 1, 2, \ldots$
- If we think of the sequnce $x, Px, P^2x, P^3x \ldots$, we know that this sequence converges to the stable distribution π
- On the other hand, we also have that the sequence $x, P^2x, (P^2)^2x, (P^2)^3x, \ldots$ necessarily converges to the same distribution π, because it is a subsequence of the sequence above
- Similarly, we could look at any k-step sequence $(P^k)^i$, $i = 0, 1, 2, \ldots$, and observe that the higher the k, the faster it converges.
- Let's try to formalize this idea

Eigendecomposition

- The eigenvalues and eigenvectors of the matrix

$$P = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}$$

are given as $(\lambda_1, \lambda_2) = (1, 0.3)$ and $v_1 = (4/7, 3/7)$, $v_2 = (1, -1)$.

Eigendecomposition

- The eigenvalues and eigenvectors of the matrix

$$P = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}$$

are given as $(\lambda_1, \lambda_2) = (1, 0.3)$ and $v_1 = (4/7, 3/7)$, $v_2 = (1, -1)$.

- For any vector $x = [x_1, x_2]$, we have the following:

$$(x_1, x_2) = \underbrace{(x_1 + x_2)}_{\alpha_1}\underbrace{(4/7, 3/7)}_{=v_1} + \underbrace{(3/7x_1 - 4/7x_2)}_{=\alpha_2}\underbrace{(1, -1)}_{=v_2},$$

i.e., the vector (x_1, x_2) is expressed as linear combination of the eigenvectors

Eigendecomposition

- For vector $x = (x_1, x_2)$ we have

$$(x_1, x_2) = \underbrace{(x_1 + x_2)}_{\alpha_1} \underbrace{(4/7, 3/7)}_{=v_1} + \underbrace{(3/7x_1 - 4/7x_2)}_{=\alpha_2} \underbrace{(1, -1)}_{=v_2},$$

Eigendecomposition

- For vector $x = (x_1, x_2)$ we have

$$(x_1, x_2) = \underbrace{(x_1 + x_2)}_{\alpha_1}\underbrace{(4/7, 3/7)}_{=v_1} + \underbrace{(3/7x_1 - 4/7x_2)}_{=\alpha_2}\underbrace{(1, -1)}_{=v_2},$$

- Now for P^N, and $x = (x_1, x_2)$ we have

$$\begin{aligned} P^N x &= P^n(\alpha_1 v_1) + P^n(\alpha_2 v_2) \\ &= \alpha_1 P^n v_1 + \alpha_2 P^n v_2 \\ &= \alpha_1 1^n v_1 + \alpha_2 0.3^n v_2 \\ &= \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1). \end{aligned}$$

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

  We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)
- The convergence rate depends on the lesser eigenvalues, which is dominated by the second largest eigenvalue (here 0.3)

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

  We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)
- The convergence rate depends on the lesser eigenvalues, which is dominated by the second largest eigenvalue (here 0.3)
- The initial values (x_1, x_2) have no effect on the convergence rate, (but they do affect the error magnitude)

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

  We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)
- The convergence rate depends on the lesser eigenvalues, which is dominated by the second largest eigenvalue (here 0.3)
- The initial values (x_1, x_2) have no effect on the convergence rate, (but they do affect the error magnitude)
- (If we started with the stable distribution, then $\alpha_2 = 0$, and there would be no converge)

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

  We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)
- The convergence rate depends on the lesser eigenvalues, which is dominated by the second largest eigenvalue (here 0.3)
- The initial values (x_1, x_2) have no effect on the convergence rate, (but they do affect the error magnitude)
- (If we started with the stable distribution, then $\alpha_2 = 0$, and there would be no converge)
- Going back to our example of sequence $(P^k)^i$, if λ is an eigenvalue of P, then λ^k is an eigenvalue of P^k.

Eigendecomposition

- $$P^N x = \pi + 0.3^n(3/7x_1 - 4/7x_2)(1, -1)$$

  We see that the first eigenvector and eigenvalue gives the convergent part (i.e., the stable distribution)
- The convergence rate depends on the lesser eigenvalues, which is dominated by the second largest eigenvalue (here 0.3)
- The initial values (x_1, x_2) have no effect on the convergence rate, (but they do affect the error magnitude)
- (If we started with the stable distribution, then $\alpha_2 = 0$, and there would be no converge)
- Going back to our example of sequence $(P^k)^i$, if λ is an eigenvalue of P, then λ^k is an eigenvalue of P^k.
- Oftentimes, we usually want to define chains that have fast convergence – a chain that converges to the stable distribution fast is said to be *fast mixing*

Spectral Graph Theory

- Let us study a graph with an ergodic Markov chain (random walk on it) - i.e. it is possible to get from any node to another node in N steps

Spectral Graph Theory

- Let us study a graph with an ergodic Markov chain (random walk on it) - i.e. it is possible to get from any node to another node in N steps
- From the discussion above, we know that that the convergence rate of the chain is given by the second eigenvector of the scaled adjancency matrix

Spectral Graph Theory

- Let us study a graph with an ergodic Markov chain (random walk on it) - i.e. it is possible to get from any node to another node in N steps
- From the discussion above, we know that that the convergence rate of the chain is given by the second eigenvector of the scaled adjancency matrix
- On the other hand, intuition says that the random walk should converge faster the more connected the graph is

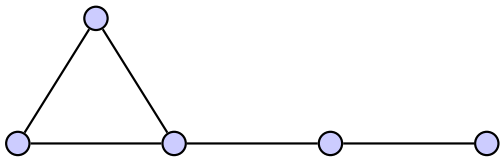
Spectral Graph Theory

- Let us study a graph with an ergodic Markov chain (random walk on it) - i.e. it is possible to get from any node to another node in N steps
- From the discussion above, we know that that the convergence rate of the chain is given by the second eigenvector of the scaled adjancency matrix
- On the other hand, intuition says that the random walk should converge faster the more connected the graph is
- Question: Can we relate the spectral properties to the geometry of the graph?

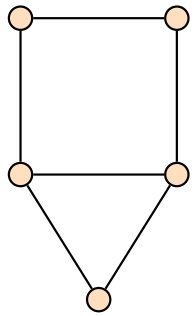
Two Example Graphs

Let's compute the second largest eigenvalue of the transition matrix of the following two graphs (the edges are undirected - we can take it as there being edge both directions)

Graph number 1

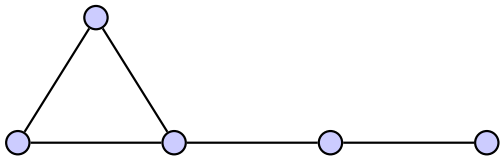


Graph number 2



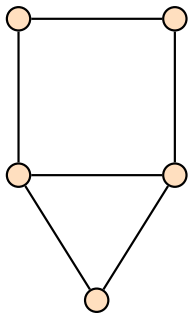
Two Example Graphs

Graph number 1



$$A_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Graph number 2



$$A_2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Two Example Graphs

$$\hat{A}_1 = \begin{pmatrix} 0 & 1/2 & 1/3 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 \\ 1/2 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 \end{pmatrix} \quad \hat{A}_2 = \begin{pmatrix} 0 & 1/2 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 1/3 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 & 0 \end{pmatrix}$$

$$\lambda_2(\hat{A}_1) \approx -0.86$$

$$\lambda_2(\hat{A}_2) \approx -0.83$$

In python

```
import numpy as np
a=0
A=np.array(([[0,1,1,0,a,],[1,0,1,0,0],[1,1,0,1,0],
[0,0,1,0,1],[a,0,0,1,0]]))
col_sums = A.sum(axis=0)
A = A / col_sums
v,t=np.linalg.eig(A)
np.round(v,2)
>>array([ 1.  ,  0.65, -0.86, -0.3 , -0.5 ])
```

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph
- To make this more formal, one usually studies what is called *Laplacian matrix* instead of the transition matrix

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph
- To make this more formal, one usually studies what is called *Laplacian matrix* instead of the transition matrix
- The upshot of this is that the we create an inner product structure on the network, a matrix that satisfies $L = BB^T = D - A$, where A is as above and D is a diagonal matrix

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph
- To make this more formal, one usually studies what is called *Laplacian matrix* instead of the transition matrix
- The upshot of this is that the we create an inner product structure on the network, a matrix that satisfies $L = BB^T = D - A$, where A is as above and D is a diagonal matrix
- Alternatively, we can represent it as $L = BB^T$, where B maps an edge (a, b) to $b - a$ – this defines an inner product on the vertices

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph
- To make this more formal, one usually studies what is called *Laplacian matrix* instead of the transition matrix
- The upshot of this is that the we create an inner product structure on the network, a matrix that satisfies $L = BB^T = D - A$, where A is as above and D is a diagonal matrix
- Alternatively, we can represent it as $L = BB^T$, where B maps an edge (a, b) to $b - a$ – this defines an inner product on the vertices
- This is necessarily a diagonizable matrix, and the second smallest eigenvector approximates the sparsest way to cut the matrix (known as *Fiedler vector*)

More generally

- The eigenvalues of the transition matrix capture something about the connectivity properties of the graph
- To make this more formal, one usually studies what is called *Laplacian matrix* instead of the transition matrix
- The upshot of this is that the we create an inner product structure on the network, a matrix that satisfies $L = BB^T = D - A$, where A is as above and D is a diagonal matrix
- Alternatively, we can represent it as $L = BB^T$, where B maps an edge (a, b) to $b - a$ – this defines an inner product on the vertices
- This is necessarily a diagonizable matrix, and the second smallest eigenvector approximates the sparsest way to cut the matrix (known as *Fiedler vector*)
- This type of thinking is also behind the UMAP algorithm for visualizing high dimensional data