

DM561 / DM562
Linear Algebra with Applications

Eigenvalues and Page Rank

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Eigenvalue Theory Applications
2. Page Rank Algorithm

Outline

1. Eigenvalue Theory Applications

2. Page Rank Algorithm

Eigenvalues and Eigenvectors

Definition

Let A be a square matrix.

- The number λ is said to be an eigenvalue of A if for some non-zero vector \mathbf{x} ,

$$A\mathbf{x} = \lambda\mathbf{x}$$

- Any non-zero vector \mathbf{x} for which this equation holds is called
eigenvector for eigenvalue λ or
eigenvector of A corresponding to eigenvalue λ

Diagonalization

Recall: Square matrices are **similar** if there is an invertible matrix P such that $P^{-1}AP = M$.

Definition (Diagonalizable matrix)

The matrix A is **diagonalizable** if it is similar to a diagonal matrix; that is, if there is a diagonal matrix D and an invertible matrix P such that $P^{-1}AP = D$

Example

$$A = \begin{bmatrix} 7 & -15 \\ 2 & -4 \end{bmatrix}$$

$$P = \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} \quad P^{-1} = \begin{bmatrix} -1 & 3 \\ 2 & -5 \end{bmatrix}$$

$$P^{-1}AP = D = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

How was such a matrix P found?

When is a matrix diagonalizable?

Summary

- Characteristic polynomial and characteristic equation of a matrix
- eigenvalues, eigenvectors, diagonalization
- finding eigenvalues and eigenvectors
- eigenspace
- diagonalize a diagonalizable matrix
- conditions for diagonalizability
- diagonalization as a change of basis, similarity
- geometric effect of linear transformation via diagonalization

Uses of Diagonalization

- find powers of matrices
- solving systems of simultaneous linear difference equations
- Markov chains
- PageRank algorithm

Powers of Matrices

$$A^n = \underbrace{AAA \cdots A}_{n \text{ times}}$$

If we can write: $P^{-1}AP = D$ then $A = PDP^{-1}$

$$\begin{aligned} A^n &= \underbrace{AAA \cdots A}_{n \text{ times}} \\ &= \underbrace{(PDP^{-1})(PDP^{-1})(PDP^{-1}) \cdots (PDP^{-1})}_{n \text{ times}} \\ &= PD(P^{-1}P)D(P^{-1}P)D(P^{-1}P) \cdots DP^{-1} \\ &= P \underbrace{DDD \cdots D}_{n \text{ times}} P^{-1} \\ &= PD^n P^{-1} \end{aligned}$$

then closed formula to calculate the power of a matrix.

Difference equations

- A **difference equation** is an equation linking terms of a sequence to previous terms, eg:

$$x_{t+1} = 5x_t - 1$$

is a first order difference equation.

- a first order difference equation can be fully determined if we know the first term of the sequence (initial condition)
- a solution is an expression of the terms x_t

$$x_{t+1} = ax_t \implies x_t = a^t x_0$$

System of Difference equations

Suppose the sequences x_t and y_t are related as follows:

$$x_0 = 1, y_0 = 1 \text{ for } t \geq 0$$

$$x_{t+1} = 7x_t - 15y_t$$

$$y_{t+1} = 2x_t - 4y_t$$

Coupled system of difference equations.

Let

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

then $\mathbf{x}_{t+1} = A\mathbf{x}_t$ and $\mathbf{x}_0 = [1, 1]^T$ and

$$A = \begin{bmatrix} 7 & -15 \\ 2 & -4 \end{bmatrix}$$

Then:

$$\mathbf{x}_1 = A\mathbf{x}_0$$

$$\mathbf{x}_2 = A\mathbf{x}_1 = A(A\mathbf{x}_0) = A^2\mathbf{x}_0$$

$$\mathbf{x}_3 = A\mathbf{x}_2 = A(A^2\mathbf{x}_0) = A^3\mathbf{x}_0$$

$$\vdots$$

$$\mathbf{x}_t = A^t\mathbf{x}_0$$

Power sequence generated by A

Markov Chains

- Suppose two supermarkets compete for customers in a region with 20000 shoppers.
- Assume no shopper goes to both supermarkets in a week.
- The table gives the probability that a shopper will change from one to another supermarket:

	From A	From B	From none
To A	0.70	0.15	0.30
To B	0.20	0.80	0.20
To none	0.10	0.05	0.50

(note that probabilities in the columns add up to 1)

- Suppose that at the end of week 0 it is known that 10000 went to A, 8000 to B and 2000 to none.
- Can we predict the number of shoppers at each supermarket in any future week t ? And the long-term distribution?

Formulation as a system of difference equations:

- Let \mathbf{x}_t be the percentage of shoppers going in the two supermarkets or none
- then we have the difference equation:

$$\mathbf{x}_t = A\mathbf{x}_{t-1}$$

$$A = \begin{bmatrix} 0.70 & 0.15 & 0.30 \\ 0.20 & 0.80 & 0.20 \\ 0.10 & 0.05 & 0.50 \end{bmatrix}, \quad \mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

- a **Markov chain** (or **process**) is a closed system of a fixed population distributed into n different states, transitioning between the states during specific time intervals.
- The transition probabilities are known in a **transition matrix** A (coefficients all non-negative + sum of entries in the columns is 1)
- **state vector** \mathbf{x}_t , entries sum to 1.

- A solution is given by (assuming A is diagonalizable):

$$\mathbf{x}_t = A^t \mathbf{x}_0 = (PD^t P^{-1}) \mathbf{x}_0$$

- let $\mathbf{x}_0 = P\mathbf{z}_0$ and $\mathbf{z}_0 = P^{-1}\mathbf{x}_0 = [b_1 \ b_2 \cdots b_n]^T$ be the representation of \mathbf{x}_0 in the basis of eigenvectors, then:

$$\mathbf{x}_t = PD^t P^{-1} \mathbf{x}_0 = b_1 \lambda_1^t \mathbf{v}_1 + b_2 \lambda_2^t \mathbf{v}_2 + \cdots + b_n \lambda_n^t \mathbf{v}_n$$

- Th.: if A is the transition matrix of a regular Markov chain, then $\lambda = 1$ is an eigenvalue of multiplicity 1 and all other eigenvalues satisfy $|\lambda| < 1$

- $\mathbf{x}_t = b_1(1)^t \mathbf{v}_1 + b_2(0.6)^t \mathbf{v}_2 + \cdots + b_n(0.4)^t \mathbf{v}_n$

- $\lim_{t \rightarrow \infty} 1^t = 1$, $\lim_{t \rightarrow \infty} 0.6^t = 0$ hence the long-term distribution is

$$\mathbf{q} = b_1 \mathbf{v}_1 = 0.125 \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.375 \\ 0.500 \\ 0.125 \end{bmatrix}$$

Theory

Definition

A **stochastic process** is any sequence of experiments for which the outcome at any stage depends on chance. A **Markov process** is a stochastic process with the following properties:

1. The set of possible outcomes or states is finite
2. The probability of the next outcome depends only on the previous outcome
3. The probabilities are constant over time:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t \quad \text{A transition matrix}$$

Theory

Definition

If the **distinct** eigenvalues of a matrix A are $\lambda_1, \lambda_2, \dots, \lambda_k$, and if $|\lambda_1|$ is larger than $|\lambda_2|, \dots, |\lambda_k|$, then λ_1 is called a **dominant eigenvalue** of A .

Any eigenvector corresponding to a dominant eigenvalue is called a **dominant eigenvector** of A .

Theorem

If a Markov chain with an $n \times n$ transition matrix A converges to a steady state vector \mathbf{x} , then

- 1. \mathbf{x} is a probability vector*
- 2. $\lambda_1 = 1$ is an eigenvalue of A and \mathbf{x} is an eigenvector belonging to λ_1*

Theory

Theorem (Perron's Theorem)

If A is a positive $n \times n$ matrix, then A has a positive real eigenvalue r with the following properties:

1. r is simple root of the characteristic equation
2. r has a positive eigenvector \mathbf{x}
3. If λ is any other eigenvalue of A , then $|\lambda| < r$.

Special case of a more general theorem due to Frobenius on irreducible nonnegative matrices.

- If A is an $n \times n$ stochastic matrix, then $\lambda_1 = 1$ is an eigenvalue of A and the remaining eigenvalues satisfy $|\lambda_j| \leq 1$ for $j = 2, \dots, n$.
- If A is stochastic and all its entries are positive, it follows from Perron's theorem that $\lambda_1 = 1$ must be a dominant eigenvalue.
- Hence the Markov chain with transition matrix A converges to a steady state vector for any starting state \mathbf{x}_0

Outline

1. Eigenvalue Theory Applications

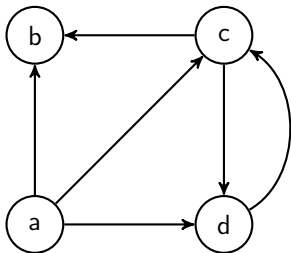
2. Page Rank Algorithm

Page Rank Algorithm

- The PageRank algorithm is one way of ranking the nodes in a graph by importance
- Brin, S.; Page, L. (1998). "The anatomy of a large-scale hypertextual Web search engine". Computer Networks and ISDN Systems. 30: 107–117.
- Currently, PageRank is not the only algorithm used by Google to order search results, but it is the first algorithm that was used by the company, and it is the best-known.

The Model

Let's consider a Tiny-Web: **nodes** are pages and **arcs** are hyperlinks.



Adjacency matrix

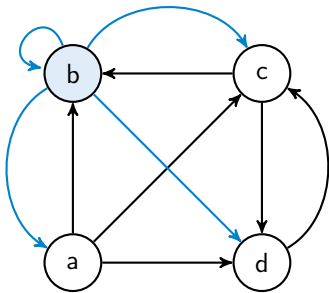
$$A = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

If n users start on random pages in the network and click on a link every 5 minutes, which page in the network will have the most views after an hour?

Which will have the fewest?

The Model

In nodes with no outgoing link (**dangling pages**), the surfer would stand. Unrealistic. \rightsquigarrow modify each sink in the graph by adding edges from the sink to every node in the graph (random jumps).



Adiacency matrix

$$\tilde{A} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

The Model

- Let $x_t(k)$ be the **likelihood** that a particular internet user is surfing webpage k at time t .
- users reaching i at $t+1$ are those that in t were in an adjacent node and chose the link to i
- we assume outgoing links are chosen with equal likelihood
- thus, $x_{t+1}(i)$ can be computed by counting the number of links pointing to page i , weighted by the total number of outgoing links for each node.

Example:

$$x_{t+1}(a) = \frac{1}{3}x_t(b),$$

$$x_{t+1}(b) = \frac{1}{3}x_t(a) + \frac{1}{2}x_t(c).$$

$$x_{t+1}(a) = 0x_t(a) + \frac{1}{3}x_t(b) + 0x_t(c) + 0x_t(d),$$

$$x_{t+1}(b) = \frac{1}{3}x_t(a) + 0x_t(b) + \frac{1}{2}x_t(c) + 0x_t(d).$$

$$x_{t+1}(i) = \sum_{j=1}^n \tilde{A}_{ij} \frac{x_t(j)}{\sum_{k=1}^n \tilde{A}_{kj}}.$$

A More Realistic Model

Let $\epsilon \in [0, 1]$ be the probability that a user follows one of the outgoing links at step t (damping factor) and $1 - \epsilon$ that he jumps at random.

$$x_{t+1}(i) = \underbrace{\epsilon \sum_{j=1}^n \left(\tilde{A}_{ij} \frac{x_t(j)}{\sum_{k=1}^n \tilde{A}_{kj}} \right)}_{\text{User stayed interested and clicked a link on the current page}} + \underbrace{(1 - \epsilon) \sum_{j=1}^n \frac{1}{n} x_t(j)}_{\text{User got bored and chose a random page}}$$

In matrix terms:

$$\mathbf{x}_{t+1} = \epsilon \hat{A} \mathbf{x}_t + (1 - \epsilon) \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{x}_t,$$

where $\mathbf{x}_t = [x_t(1), x_t(2), \dots, x_t(n)]^T$, $\mathbf{1}$ is a vector of n ones, and \hat{A} is the $n \times n$ matrix with entries

$$\hat{A}_{ij} = \frac{\tilde{A}_{ij}}{\sum_{k=1}^n \tilde{A}_{kj}}.$$

For our example:

$$\hat{A} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1/4 & 0 & 0 \\ b & 1/3 & 1/4 & 1/2 & 0 \\ c & 1/3 & 1/4 & 0 & 1 \\ d & 1/3 & 1/4 & 1/2 & 0 \end{array}$$

$$\frac{1}{n}\mathbf{1}\mathbf{1}^T = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1/4 & 1/4 & 1/4 & 1/4 \\ b & 1/4 & 1/4 & 1/4 & 1/4 \\ c & 1/4 & 1/4 & 1/4 & 1/4 \\ d & 1/4 & 1/4 & 1/4 & 1/4 \end{array}$$

$$\mathbf{x}_{t+1} = \left(\epsilon \hat{A} + (1 - \epsilon) \frac{1}{n} \mathbf{1}\mathbf{1}^T \right) \mathbf{x}_t$$

$$\bar{A} = \epsilon \hat{A} + (1 - \epsilon) \frac{1}{n} \mathbf{1}\mathbf{1}^T$$

all terms of \bar{A} are nonnegative and all its columns sum up to 1, ie, \bar{A} is a positive stochastic matrix

$$\mathbf{x}_{t+1} = \bar{A} \mathbf{x}_t$$

is a Markov chain

Computing the Rankings

- Let's define the **page rank** of node i as the steady state of the Markov chain:

$$x(i) = \lim_{t \rightarrow \infty} x_t(i).$$

- If \mathbf{x} exists, then taking the limit as $t \rightarrow \infty$ of both sides of the Markov chain gives the following:

$$\lim_{t \rightarrow \infty} \mathbf{x}(t+1) = \lim_{t \rightarrow \infty} \left[\epsilon \hat{A} \mathbf{x}(t) + (1 - \epsilon) \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{x}(t) \right]$$

$$\mathbf{x} = \epsilon \hat{A} \mathbf{x} + (1 - \epsilon) \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{x}$$

$$\left(I - \epsilon \hat{A} - (1 - \epsilon) \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) \mathbf{x} = \mathbf{0}$$

$$\left(I - \epsilon \hat{A} \right) \mathbf{x} = \frac{1 - \epsilon}{n} \mathbf{1}$$

$$\mathbf{1} \mathbf{1}^T \mathbf{x} = \mathbf{1} \text{ since } \sum_i x_t(i) = 1$$

\rightsquigarrow a system of linear equations!

- Alternatively, setting $\bar{A} = \epsilon \hat{A} + \frac{1-\epsilon}{n} \mathbf{1}\mathbf{1}^T$

$$(I - \bar{A}) \mathbf{x} = \mathbf{0}$$

$$\bar{A}\mathbf{x} = \mathbf{x}$$

- \mathbf{x} is an eigenvector of \bar{A} corresponding to the eigenvalue $\lambda = 1$.
- since the columns of \bar{A} sum to 1, and because the entries of \bar{A} are strictly positive, Perron's theorem guarantees that $\lambda = 1$ is the unique eigenvalue of \bar{A} of largest magnitude, and that the corresponding eigenvector \mathbf{x} is unique up to scaling.
- \mathbf{x} can be scaled so that each of its entries are positive, meaning $\mathbf{x}/\|\mathbf{x}\|$ is the desired PageRank vector.

An Iterative Method

- Solving the system of linear equations above or finding the eigenvalues/eigenvectors is feasible for small networks, but they are not efficient strategies for very large systems.
- Iterative technique (Power Method):

1. Start with $t = 0$ and an initial guess \mathbf{x}_0

2. Compute \mathbf{x}_{t+1} with

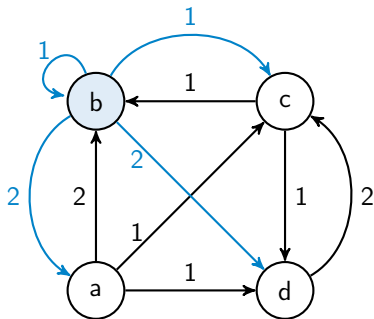
$$\mathbf{x}_{t+1} = \left(\epsilon \hat{A} + (1 - \epsilon) \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) \mathbf{x}_t$$

and set $t \leftarrow t + 1$

3. if $\| \mathbf{x}_t - \mathbf{x}_{t-1} \|$ is sufficiently small stop, otherwise go to 2.

PageRank on Weighted Graphs

If hyperlinks to page **a** are clicked on more frequently than hyperlinks to page **b**, the edge from node **a** should be given more weight than the edge to node **b**.



$$A = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix} \end{matrix}$$

$$\hat{A} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1/4 & 0 & 0 \\ 1/2 & 1/4 & 1/3 & 0 \\ 1/4 & 1/4 & 0 & 1 \\ 1/4 & 1/4 & 2/3 & 0 \end{bmatrix} \end{matrix}$$

The columns of \hat{A} still sum to 1. Thus $\bar{A} = \epsilon \hat{A} + \frac{1-\epsilon}{n} \mathbf{1}\mathbf{1}^T$ is still positive stochastic, so we can expect a unique **x** to exist.

Python: Networkx

- It represents graphs internally with dictionaries, thus taking full advantage of the sparsity in a graph.
- The base class for directed graphs is called `nx.DiGraph`.
- Nodes and edges are usually added or removed incrementally with the following methods.

Method	Description
	Add a single node. <code>add_node()</code>
	Add a list of nodes. <code>add_nodes_from()</code>
	Add an edge between two nodes, adding the nodes if needed. <code>add_edge()</code>
	Add multiple edges (and corresponding nodes as needed). <code>add_edges_from()</code>
	Remove a single edge (no nodes are removed). <code>remove_edge()</code>
	Remove multiple edges (no nodes are removed). <code>remove_edges_from()</code>
	Remove a single node and all adjacent edges. <code>remove_node()</code>
	Remove multiple nodes and all adjacent edges. <code>remove_nodes_from()</code>

Example

```
>>> import networkx as nx
```

```
# Initialize an empty directed graph.
```

```
>>> DG = nx.DiGraph()
```

```
# Add the directed edges (nodes are added automatically).
```

```
>>> DG.add_edge('a', 'b', weight=2)      # a --> b (adds nodes a and b)
```

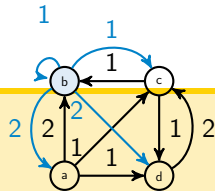
```
>>> DG.add_edge('a', 'c', weight=1)      # a --> c (adds node c)
```

```
>>> DG.add_edge('a', 'd', weight=1)      # a --> d (adds node d)
```

```
>>> DG.add_edge('c', 'b', weight=1)      # c --> b
```

```
>>> DG.add_edge('c', 'd', weight=2)      # c --> d
```

```
>>> DG.add_edge('d', 'c', weight=2)      # d --> c
```



Networkx

- `nx.Digraph` object can be queried for information about the nodes and edges.
- Dictionary-like indexing to access node and edge attributes, such as the weight of an edge.

Method	Description		
	Return if <code>node</code> is a node in the graph. <code>has_node(node)</code>	True	A
	Return if there is an edge from <code>u</code> to <code>v</code> . <code>has_edge(u,v)</code>	True	
	Iterate through the edges. <code>edges()</code>		
	Iterate through the nodes. <code>nodes()</code>		
	Return the number of nodes. <code>number_of_nodes()</code>		
	Return the number of edges. <code>number_of_edges()</code>		

Example

```
# Check the nodes and edges.
>>> DG.has_node('a')
True
>>> DG.has_edge('b', 'a')
False
>>> list(DG.nodes())
['a', 'b', 'c', 'd']
>>> list(DG.edges())
[('a', 'b'), ('a', 'c'), ('a', 'd'), ('c', 'b'), ('c', 'd'), ('d', 'c')]

# Change the weight of the edge (a, b) to 3.
>>> DG['a']['b']["weight"] += 1
>>> DG['a']['b']["weight"]
3
```

PageRank in Networkx

- NetworkX efficiently implements several graph algorithms.
- The function `nx.pagerank()` computes the PageRank values of each node iteratively with sparse matrix operations.
- This function returns a dictionary mapping nodes to PageRank values

```
# Calculate the PageRank values of the graph.  
>>> nx.pagerank(DG, alpha=0.85)      # alpha is the damping factor (epsilon).  
{ 'a': 0.08767781186947843,  
  'b': 0.23613138394239835,  
  'c': 0.3661321209576019,  
  'd': 0.31005868323052127 }
```


Summary

1. Eigenvalue Theory Applications

2. Page Rank Algorithm