MATH280
Case 1: PageRank

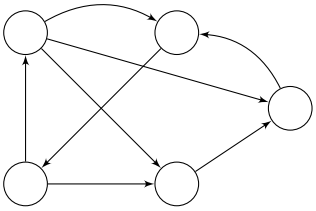
If you haven't done so already, read section 10.1 and 10.2 in *Linear Algebra and Its Applications* (available on Canvas).

Note: In this note, indices start at 0, to correspond with NumPy arrays. If you are using Matlab, vectors and matrices there are 1-indexed, and you have to adjust accordingly.

1 Directed graphs

A directed graph is a set of vertices with directed edges between the vertices.

Example (A directed graph)**.**

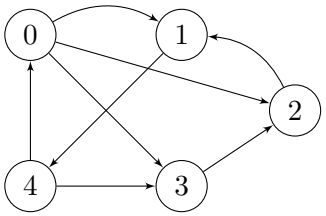


In the PageRank algorithm, each vertex represents a web page, and each edge a link between webpages.

A directed graph can be represented on a computer in several ways. For our purposes, we will focus on the *adjacency matrix*. To form the adjacency matrix of a graph, we first label the vertices of the directed graph with indices $\{0, 1, \ldots, n-1\}$, where n is the number of vertices.

The adjacency matrix is an $n \times n$-matrix $\mathbf{A}$, with entries

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge } i \to j, \\ 0 & \text{if not.} \end{cases}$$



Example (Adjacency matrix)**.** We label the vertices in the graph
To set up the adjacency matrix, we put a 1 at position (i, j) if there is an edge $i \to j$, otherwise we put zero.

$$\mathbf{A} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

2 Transition matrix

The PageRank Algorithm uses a transition matrix $\mathbf{P}$ for the random walk across the graph. (See Chapter 10.2 i Lay). This matrix is a stochastic matrix where p_{ij} is the probability of going from state j to state i.

$$p_{ij} = \begin{cases} \frac{1}{m_j} & \text{if there is an edge } j \to i \\ 0 & \text{otherwise} \end{cases},$$

where m_j is the number of edges pointing out of node j.

Note: The adjacency matrix has $a_{ij} \neq 0$ if there is an edge $i \to j$, while the transition matrix has $p_{ij} \neq 0$ if there is an edge $j \to i$.

To compute $\mathbf{P}$ from $\mathbf{A}$ we need to do three things:

1. Compute the number of edges going out from each node.
   $m_i = \sum_{j=0}^{n-1} a_{ij}$.
2. Divide each row of $\mathbf{A}$ by the corresponding number of edges, resulting in a matrix $\mathbf{C}$.
   $c_{ij} = \frac{a_{ij}}{m_i}$.
3. Transpose the resulting matrix.
   $\mathbf{P} = \mathbf{C}^\top$.

Example (Transition matrix)**.** For our example graph, the numbers of edges pointing out of each node are, respectively

$$\mathbf{m} = \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 3 & 1 & 1 & 1 & 2 \end{pmatrix} \end{array}$$

After dividing each row of $\mathbf{A}$ and transposing, we end up with the transition matrix $\mathbf{P}$.

$$\mathbf{P} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 1 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 1 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Task 1. Write a function that takes an adjacency matrix $\mathbf{A}$ as input, and returns the corresponding transition matrix $\mathbf{P}$.

3 The power method

The PageRank of each node is calculated from the steady-state vector $\mathbf{q}$, defined by

$$\mathbf{Pq} = \mathbf{q}, \quad \text{and} \quad \sum_{i=0}^{n-1} q_i = 1.$$

As shown in the lectures,

$$\mathbf{q} = \lim_{k\to\infty} \mathbf{x}_k,$$

where $\mathbf{x}_k$ is a sequence of probability vectors defined by

$$\mathbf{x}_{k+1} = \mathbf{Px}.$$

The idea of the power method is to compute the sequence $\mathbf{x}_0, \mathbf{x}_1 = \mathbf{Px}_0, \mathbf{x}_2 = \mathbf{Px}_1, \ldots$, and use the approximation

$$\mathbf{q} \approx \mathbf{x}_N,$$

for some large N.

Note: Here we have assumed that the initial vector $\mathbf{x}_0$ is a probability vector. If it is not, the sequence will converge to some scalar multiple of $\mathbf{q}$ instead.

Task 2. Implement the power method. Your function should take as arguments the transition matrix $\mathbf{P}$, an initial probability vector $\mathbf{x}_0$, and the number of iterations N.

Example. For our example graph, $\mathbf{q}$ can be found by solving the homogenous linear equation $(\mathbf{P} - \mathbf{I})\,\mathbf{q} = 0$ and choosing the free parameter such that $\sum_{i=0}^{n-1} q_i = 1$.

$$\mathbf{q} = \left[\frac{1}{8}, \frac{1}{4}, \frac{5}{24}, \frac{1}{6}, \frac{1}{4}\right]^T.$$

You can use this to test your method.

Task 3. In this task, you are going to test your implementation on a set of webpages and links. This set consists of webpages with an url starting with www.nmbu.no/fakultet/realtek and internal links between them (In January 2022). Dangling nodes have been removed.

The adjacency matrix is stored in 'adjacency_realtek.txt', and the urls represented by each index in 'keyvals.txt'.

What are the top 5 webpages by PageRank on the RealTek webpages?

4 Additional challenges

In the power method implemented above, we specify the number of iterations. It would be better to iterate until

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq Tol$$

for some specified tolerance Tol.

Task 4 (a)**.** Implement a new function that does this.

For more efficient code, you should not compute the matrix-vector product $\mathbf{P}\mathbf{x}_{k-1}$ more than once per iteration. It is good practice to implement a maximum number of iterations, so your code doesn't get stuck in an infinite loop.

The actual PageRank algorithm is more robust than what we have done here. By making some adjustments, it can handle dangling nodes (webpages without links) and other complications that may arise.

Task 4 (b)**.** Implement Adjustment 1 and 2 described in *Linear Algebra and its Applications* Chapter 10.2.

5 Some additional remarks

Our implementation only works for graphs with less than a few thousand nodes. For larger graphs, better implementations are needed. Adjacency matrices are usually sparse, that is, most of the entries are zero. Efficient matrix algorithms take advantage of this. In Python, the most used implementation of sparse matrices is scipy.sparse. See `https://docs.scipy.org/doc/scipy/reference/sparse.html`.

Really efficient implementations of the PageRank algorithm never explicitly store the adjacency matrix or the transition matrix. It is possible to compute a matrix-vector product $\mathbf{P}\mathbf{v}$ using only the adjacency list (`https://en.wikipedia.org/wiki/Adjacency_list`) of the graph.