DM561 – Linear Algebra and Applications
DM562 – Scientific Programming

Obligatory Assignment 5, Autumn 2018

Submission Deadline: Wednesday, January 9, 2018, at noon

In the period between Christmas and New Years Eve, issues with the automatic grading or other type of assistance is not guaranteed.

In this assignment it is not allowed to use the module Networkx.

This document is associated with the files `asg5.py` and the data files:
`web_stanford.txt`, `psh-uefa-2017-2018.csv`, which are available in the git repository. The file `asg5.py` is the only one that needs to be edited and submitted.

Task 1

Subtask a. Write a class for representing directed graphs via their adjacency matrices. The constructor should accept an $n \times n$ adjacency matrix A and a list of node labels (such as [a, b, c, d]) defaulting to `None`. Modify A into $\widetilde{A}$ as shown in the slides so that there are no sinks in the corresponding graph, then calculate the $\widehat{A}$:

$$\widehat{A}_{ij} = \frac{\widetilde{A}_{ij}}{\sum_{k=1} \widetilde{A}_{kj}}.$$

Save $\widehat{A}$ and the list of labels as attributes of the DiGraph objects. Use $[0, 1, \ldots, n-1]$ as the labels if none are provided. Finally, raise a `ValueError` if the number of labels is not equal to the number of nodes in the graph. (Hint: use array broadcasting to compute $\widehat{A}$ efficiently. The Numpy function `where` could help you.)
In the docstring example you find the same graph that was used in slides.

Subtask b. Add the following methods to your class from the previous Subtask. Each should accept a damping factor ϵ (defaulting to 0.85), compute the PageRank vector $\mathbf{x}$, and return a dictionary mapping label i to its PageRank value x_i.

1. `linsolve()`: solve for $\mathbf{x}$ in slide 32 after substituting the fact that $\mathbf{1}\mathbf{1}^\mathsf{T}\mathbf{x} = \mathbf{1}$ since $\sum_i x_t(i) = 1$:

$$\begin{aligned} \left(I - \epsilon\widehat{A} - (1-\epsilon)\tfrac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}\right)\mathbf{x} &= \mathbf{0} \\ \left(I - \epsilon\widehat{A}\right)\mathbf{x} &= \tfrac{1-\epsilon}{n}\mathbf{1} \end{aligned}$$

2. `eigensolve()`: solve $\bar{A}\mathbf{x} = \mathbf{x}$ from slide 33 for $\mathbf{x}$. Normalize the resulting eigenvector so that its entries sum to 1.

3. `itersolve()`: in addition to ϵ, accept an integer `maxiter` and a float `tol`. Iterate until $||\mathbf{x}_t - \mathbf{x}_{t-1}||_1 <$ `tol` or $t>$ `maxiter`. Use $\mathbf{x}_0 = [\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]^\mathsf{T}$ as the initial vector (any positive vector that sums to 1 will do, but this assumes equal starting probabilities).

In the docstring example you can test that the three methods yield the same results. For the graph of the lecture with $\epsilon = 0.85$, you should get the following dictionary mapping labels to PageRank values.

```
{'a': 0.095758635, 'b': 0.274158285, 'c': 0.355924792, 'd': 0.274158285}
```

Subtask c. Write a function that accepts a dictionary mapping labels to PageRank values, like the outputs in the previous Subtask and returns a list of labels sorted **from highest to lowest** rank. (Hint: look for the built-in function `sorted()`.)
For the graph in the slides with $\epsilon = 0.85$, the list is [c, b, d, a] (or [c, d, b, a], since b and d have the same PageRank value).
Looking at the graph from the slides, it is easy to see why a has the lowest PageRank value: the only other node that points to it is b. It also makes sense that c has the highest ranking, since c and d both have edges from the other three nodes pointing to them, but d only has one edge (pointing to c), while c points to both b and d. In other words, at each step d distributes all of its importance to c, while c splits its importance between b and d.
Of course, constructing rankings is much more difficult to do by hand when there are more than just a few nodes in the graph.

Task 2

The file `web_stanford.txt` contains a subset of the information on Stanford University webpages and the hyperlinks between them, gathered in 2002. Each line of the file is formatted as `a/b/c/d/e/f...`, meaning the webpage with ID `a` has hyperlinks to webpages with IDs `b`, `c`, `d`, and so on.
Write a function that accepts a damping factor ϵ defaulting to 0.85. Read the data and get a list of the n unique page IDs in the file (the labels). Construct the $n \times n$ adjacency matrix of the graph where node j points to node i if webpage j has a hyperlink to webpage i. Use your class from Subtask 1.a and its `itersolve()` method from Subtask 1.b to compute the PageRank values of the webpages, then rank them with your function from Subtask 1.c. Return the ranked list of webpage IDs. (Hint: after constructing the list of webpage IDs, make a dictionary that maps a webpage ID to its index in the list. The values are the row/column indices in the adjacency matrix for each label.)
With $\epsilon = 0.85$, the top three ranked webpage IDs are 98595, 32791, and 28392.

Task 3

The file `psh-uefa-2018-2019.csv` contains data for men's football teams in Europe for the current season from the main leagues in Italy, England, France, Scotland, Spain, Germany, Greece, Belgium, Holland, Portugal, Turkey, together with Champions League and Europa League. Each line represents a different game, formatted `home_team,away_team,home_goals,away_goals`. Write a function that accepts a filename and a damping factor ϵ defaulting to 0.85. Read the specified file and get a list of the n unique teams in the file. Construct the $n \times n$ adjacency matrix of the graph where node j points to node i with weight w if team j was defeated by team i in w games. That is, **edges point from losers to winners**. Ignore draw games. Use your class from Subtask 1.c and its `itersolve()` method to compute the PageRank values of the teams, then rank them with your function from Subtask 2.c. Return the ranked list of team names.
Using `psh-uefa-2018-2019.csv` with $\epsilon = 0.85$, the top three ranked teams in the current season should be Liverpool, Athletic Madrid, Paris SG.
The damping factor ϵ acts as an "upset" factor: a larger ϵ puts more emphasis on win history; a smaller ϵ allows more randomness in the system, giving underdog teams a higher probability of defeating a team with a better record.
Note that we did not take into account weights derived from home-away situations, goal difference, importance of the match and aggregate results from Europa and Champions league.
It is also worth noting that the sink-fixing procedure is still reasonable for this model because it gives every other team **equal** likelihood of beating an undefeated team. That is, the additional arcs don't provide an extra advantage to any one team.