MA1691 | Summer project 2020.

1 Project background

This project builds upon the work done in the second half of Term 2, where we learned the mathematics underlying the Google PageRank algorithm. Most of the key learning materials needed to implement this project can be found on the Blackboard, in folders for Project 2 (weeks 22–25). More specifically

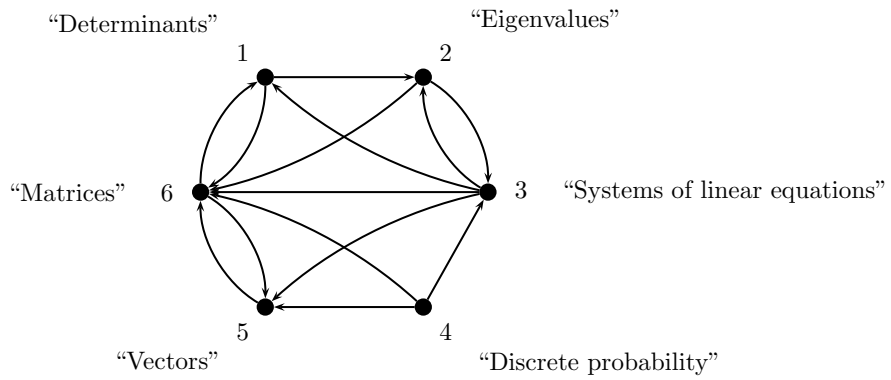
- Seminar 0 (Week 22) explained the motivation for ranking nodes on a network and outlined the imaginary computer simulation that underpins the PageRank algorithm. During the Plenary Session 1 (Week 23) we introduced the key concepts needed to represent networks mathematically: adjacency matrices, as well as in- and out-degrees. We learned how to handle adjacency matrices in Matlab during Lab 1 (Week 23).

- The simulation that motivates the PageRank algorithm describes surfers who browse a network according to some set rules. The simulation of individual surfers is possible, but very computationally inefficient, so direct simulation needs to be replaced by studying the probabilities for surfers to be on every particular node within the network, and studying how these probabilities change from iteration to iteration. Seminar 1 (Week 23) was concerned with hands-on calculations of these probabilities.

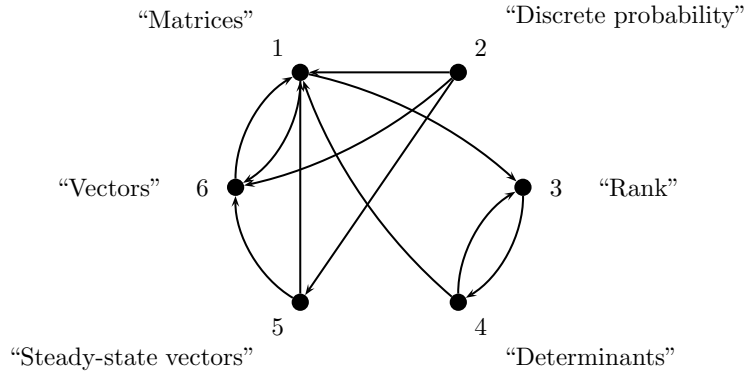
- In Plenary Session 2 (Week 24) we showed how the effect of iterations on probabilities can be reproduced by multiplying probabilities vector by the conditional probability matrix. Lab 2 (Week 24) was devoted to computing this matrix from a given adjacency matrix, see `cmat1.m` in Lab 2 (Week 24). This allowed us to reproduce the effect of simulations more directly, see `iterations1.m` in Lab 2 (Week 24).

- Experimentation with script `iterations1.m` in Lab 2 (Week 24) showed that probabilities often become stable after a few iterations of the PageRank algorithm. Plenary Session 3 (Week 25) argues why this ought to be expected for many networks of interest and shows how to reduce the problem of finding the limiting probabilities to the solution of a system of linear equations. This transformation was exemplified for a tiny network considered during Seminar 3 (Week 25); a general computer implementation of the same algorithm was covered in Lab 3 (Week 25). As the result, we wrote the script `direct1.m` which takes the adjacency matrix for a network and determines the limiting probabilities for surfers to be on any particular node. The nodes are then ranked in the order of decreasing limiting probabilities (higher probabilities correspond to higher rankings).

- Last, but not least, we also needed examples of networks to study, so Seminar 2 (Week 24) was devoted to the examples of nodes and edges that would be appropriate to a network showing relationships between mathematical topics.



Network 1 (for students with odd u, where u is the penultimate digit of your student number).



Network 2 (for students with even u, where u is the penultimate digit of your student number).

2 Project tasks (Part 1)

Let u, v be the last two digits of your student number (as read from left to right). Some of the tasks in Parts 1 and 2 of the project will use these digits to personalize your instructions.

In this first group of tasks you will need to compute and analyse the ranking of nodes for a pre-defined network, describing the dependencies between 6 mathematical topics. Each node represents a topic, and each edge represents the dependence. The direction of the edge goes **from** topic A **to** topic B if topic B underlies or is employed in some way in topic A; for instance the edge from "Systems of Linear Equation" to "Matrices" indicates that any analysis of systems of linear equations can be performed by studying the corresponding system matrices. If u (the penultimate digit of your student number) is odd, you will be working with the Network 1, otherwise, you will be working with the Network 2, see the diagrams above.

Task 1(a): Create script `task1a.m`, based upon script `direct1.m`, to compute the vector of limiting probabilities for your network. These are the probabilities that a random surfer following the PageRank algorithm will visit each node. The PageRank will then rank the node with the largest probability first, the node with the second largest probability second, etc (10%).

Task 1(b): Write a short section of your report to describe your findings. It must include the mathematical definition of your adjacency matrix (typed using the equation editor), as well as a brief statement of what changes to `direct1.m` were actually

needed. Also write a paragraph to state what are the top three ranked nodes for your network, what are the values of their limiting probabilities and their ranks, and why do you think the associated topics got the highest ranks (10%).

For the next group of tasks you will need to add 3–5 nodes to your network, as well as at least 7 more edges. The resulting network should contain not fewer than 9 and not more than 11 topics. The following are some suitable examples of mathematical topics that can become new nodes, as well as possible dependencies between them:

- Systems of linear equations and matrix algebra: the latter can be used to solve the former, so if nodes *systems of linear equations* and *matrix algebra* are included in your network, there would be a directed edge pointing from *systems of linear equations* to *matrix algebra*.
- *Infinite series* $\longrightarrow$ *limits*: the theory of limits is needed to evaluate infinite series.
- *Definite integrals* $\longleftarrow$ *area formulas in geometry*: definite integrals can be used to calculate the area under a curve. Is there a link in the opposite direction?
- There is a dependence between discrete distributions (in probability) and infinite series: the moments of a discrete distribution are often calculated using an infinite series. Hence, if nodes *discrete distributions* and *infinite series* are included in your network, there would be a directed edge pointing from *discrete distributions* to *infinite series*.

You may use one of these examples in your network, but not more than one. These are tasks that you need to perform for the newly created network:

Task 2(a): Prepare a graphical representation of your network, with each node numbered and labelled by a mathematical topic. Do not forget to clearly indicate which nodes and edges have already been there, and which ones you contributed. People usually draw their networks using Microsoft Word or Microsoft Powerpoint, but you are free to use any other software if you prefer (10%).

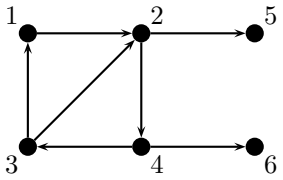
Task 2(b): For each edge that you added to the network to connect a pair of nodes, provide a brief justification in your report (1-2 sentences) of why do you believe these nodes must be connected (20%).

Task 2(c): Create script `task2c.m` based upon your earlier created script `task1a.m` to compute limiting probabilities for your extended network. Try running it. Write a few sentences in your report to briefly describe the results you obtain (10%).

Depending on the configuration of the network you just created, your script from Task 2(c) may fail to report the limiting probabilities (or it may give you warnings about the matrix `M`). To understand why this may be the case, you need to learn some additional theory.

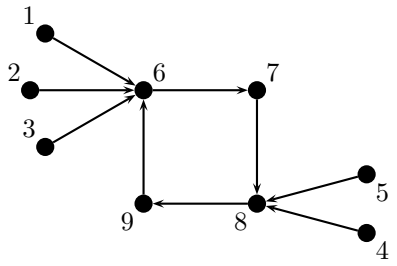
3 The extended version of the PageRank algorithm

Our script `direct1.m` works by setting up a system of linear equations for the probabilities p_i that a random surfer is found at any particular node i of the network after being allowed to surf for some arbitrarily long time. Unfortunately, for some networks, it may be the case that the simplified version of PageRank simulations we discussed thus far ("choose one node with equal probability from among those pointed to by links from the current node") will fail to produce reasonable rankings. There are two key breakdown mechanisms. One is exemplified by the following network:



If the surfer reaches a node with the out-degree equal to zero (i.e. node 5 or node 6), our set of rules simply breaks down, as the surfer has nowhere to go next. Every surfer will end up on one of these two "dead end" nodes after a while, with the limiting probabilities for all other nodes being equal to zero, which certainly does not seem like a reasonable ranking! The way out is to allow the surfer who reached the dead end to jump to any other page in the network, with equal probability. Since there is a total of $N-1$ available pages (excluding the one page the surfer is currently visiting), the probability to go to any particular page is $1/(N-1)$.

The second mechanism is more subtle, exemplified by the following network:



A closed cycle with nodes 6, 7, 8 and 9 means that the surfer gets stuck forever moving along a sequence of these 4 nodes, unable to reach any other node in the network. These two breakdown mechanisms are special cases of a general feature of many directed networks: a group of nodes such that there are no links pointing from any of them to any of the nodes outside the group. Such a group of nodes creates a "trap", and defeats the purpose of letting a surfer explore the whole of the network to investigate the relative importance of various nodes.

An ingenious way out of this situation is to "break the rules": instead of "obediently" following the links, we allow the surfer to jump to an arbitrarily chosen node every once in a while, irrespective of any connections. One way to create an algorithm out of this "once in a while" idea is to associate a probability to it: when deciding on the next move the surfer either follows the links as per our original algorithm with probability $1-\alpha$, or jumps to any node (other than itself) in the network with probability α. The value of α is usually set to 0.15–0.25. A part of your assignment will be to investigate how the rankings depend on the value of α.

This rule frees us from the need to determine whether a node belongs to a "trap" (this may be a non-trivial and computationally challenging task for large networks). Instead, the surfer performs this rule-breaking jump with probability α regardless of whether this is genuinely needed to escape a "trap" or not, i.e., even when the surfer is at any node with non-zero out-degree. To apply the algorithm, one has to distinguish between nodes with zero out-degree ("dead ends") and nodes with out-degree equal to a finite number. However, unlike searching for traps, calculating the out-degree is a simple matter of summing the entries in a column of the adjacency matrix, so implementing the algorithm is computationally straightforward.

These new, more general surfing rules can be succinctly reproduced by appropriately generalizing definition of the conditional probability matrix. Once this is done, we would have

the following general formula for the elements of C:

$$c_{ij} = \begin{cases} 0, & i = j; \\ \dfrac{\alpha}{N-1} + (1-\alpha)\dfrac{a_{ij}}{d_j}, & d_j > 0; i \neq j; \\ \dfrac{1}{N-1}, & d_j = 0; i \neq j, \end{cases} \tag{1}$$

where d_j is the out-degree of the node j.

4 Implementation of the full conditional probability matrix

Let us now implement the modified surfing algorithm in software. Our old Matlab function `cmat1()` assumed that all elements of the conditional probability matrix had the form $c_{ij} = a_{ij}/d_j$. This assumption made its computation very simple:

```
% this nested loop goes through all elements of C
% i.e. it looks at every possible combination of "from"
% and "to" nodes
for from=1:N
    for to=1:N
        % probability to go from "from" node to "to" node
        % depends on whether there is an edge going from
        % "from" to "to" and also on the out-degree of
        % the "from" node
        C(to,from) = A(to,from)/outdeg(from);
    end
end
```

However, this simplicity also made the code vulnerable to breakdown if presented with a network in which some nodes have vanishing out-degrees. To avoid such breakdowns and implement the algorithm encoded in Equation (1), we need to examine each pair of `from` and `to` variable more closely, to catch the cases of “sink" nodes with zero out-degrees. This is done by adding an `if`-statement that explicitly checks if the out-degree of a node is equal to zero or not. Depending on the result, the relevant branch of the formula in equation (1) is chosen to compute the corresponding element in **C**. We also need to add an `if`-statement that ensures the surfer moves *somewhere* by setting the diagonal elements of **C** to zero. (This happened automatically in `cmat1()` because the diagonal elements of the adjacency matrix A are equal to zero, but now we need to cover this case explicitly.) Putting everything together, we have:

```
for from=1:N
    for to=1:N
        if from==to
            % never stay at the same node
            C(to,from) = 0;
        elseif outdeg(from)>0
            % jump to a random node (with probability
            % alpha) or follow one of the outgoing
            % links (with probability (1-alpha))
            C(to,from) = alpha/(N-1) ...
```

```
                    +(1-alpha)*A(to,from)/outdeg(from);
            else
                % jump to a random node
                C(to,from) = 1/(N-1);
            end
        end
    end
```

This code makes use of the variable `alpha`, which must be given a value in order for the calculation to work. The best option would be to add this variable as the second input parameter of the function `cmat2()`, so that its value can be modified by a user externally. With this final adjustment, the full version of the function `cmat2()` is as follows:

File `cmat2.m`:

```
function C = cmat2(A, alpha) % A is the original adjacency matrix

    N = size(A,1); % A is square; 1 means "the number of rows"
    outdeg=sum(A); % compute vector of out-degrees

    C = zeros(N,N); % C has the same dimensions as A

    % this nested loop goes through all elements of C
    % i.e. it looks at every possible combination of "from"
    % and "to" nodes
    for from=1:N
        for to=1:N
            % probability to go from "from" node to "to" node
            % depends on whether there is an edge going from
            % "from" to "to" and also on the out-degree of
            % the "from" node
            if from==to
                % never stay at the same node
                C(to,from) = 0;
            elseif outdeg(from)>0
                % jump to a random node (with probability
                % alpha) or follow one of the outgoing
                % links (with probability (1-alpha))
                C(to,from) = alpha/(N-1) ...
                    +(1-alpha)*A(to,from)/outdeg(from);
            else
                % jump to a random node
                C(to,from) = 1/(N-1);
            end
        end
    end
end
```

To use so extended version of PageRank algorithm, the main code (`direct1.m` or any of the scripts you derived from it) has to call function `cmat2()` rather than `cmat1()`. The only

adjustment is that `cmat2()` has to be given *two* arguments, the first of which is the adjacency matrix, and the second argument is the value of the parameter α. E.g., we replace the line

```
C = cmat1(A); % matrix C is computed by the function cmat1
```

with the line

```
C = cmat2(A,0.25); % matrix C is computed by the function cmat2
```

to simulate what would happen when $\alpha = 0.25$.

5 Project tasks (Part 2)

The remaining tasks are all about applying the extended version of the PageRank algorithm to your network and comparing the results with your observations from Task 2(a–b).

Task 3(a): Create script `task3a.m`, based upon script `task2a.m`, to compute the vector of limiting probabilities for your network using an extended version of the PageRank algorithm. This will require you to modify `task1a.m` to use function `cmat2()` instead of `cmat1()`. Use the value of $\alpha = 0.15 + 0.0v$, where v is the last digit of your student number (10%).

Task 3(b): Add a brief section to your report and describe what are the top three ranked topics for your modified network, what are their limiting probabilities and ranks, and how these results are different (if at all) from your results in Task 2(c) (10%).

The remaining two challenge tasks are designed to stretch the strongest students in the class and, therefore, are much harder to do.

Task 4(a): Create Matlab script `task4a.m` based upon script `task3a.m`, to generate a plot showing how limiting probabilities depend on the parameter α (α should change between 0 and 1). Make sure all your modifications are properly commented in the script (10%).

Task 4(b): Write a section in your report including the plot resulting from Task 4(a), explain what specific modifications to `task3a.m` were needed to generate such a plot and also describe how limiting probabilities change for $\alpha \in [0, 1]$ (10%).

6 Project summary

Overall, the key deliverables for this project are as follows

- The report that contains answers to Tasks 1(b), 2(b), 3(b) and 4(b), which should be submitted as a .PDF file. The overall report should not be longer than 3 sides of A4 (assuming 2cm margins, at least 11pt font and 1.5 line spacing).
- The network diagram as specified in Task 2(a), which should be submitted either as a part of the report, or as a separate .PDF document.
- All Matlab scripts that were produced to answer Tasks 1(a), 2(c), 3(a) and 4(a), which should be submitted within a single .ZIP file. If you have incomplete answers to some of the tasks, please submit incomplete solutions as well, because partial marks are often possible. Use your student ID as the name of the .ZIP file.