

D3. Random walks on graphs.

Notes by Malcolm Brooks and Adam Piggott,
expanded from notes of Pierre Portal
with influences from Judy-anne Osborn .

Unfortunately, Random walks are not covered in our text by Epp, nor in the books by Johnsonbaugh or Kolman *et. al.*

Random walks: idea

Let G be a digraph with n vertices $V = V(G) = \{1, \dots, n\}$ and (directed) edge set $E = E(G)$.

[We will stick to these meanings for G, n, V and E throughout this final section of the notes.]

Imagine that you are walking on this digraph.

Travelling through an edge takes you one unit of time.

At time 0, you are at vertex x .

At time 1 you are at a vertex $y \in V$, with $(x, y) \in E$.

At time 2 you are at a vertex $z \in V$ with $(y, z) \in E$.

At time k you are at a vertex t and there is a walk of length k from x to t .

Before each step, you choose where to go next probabilistically:

If you are at a vertex i you go to vertex j with probability p_{ij} .

[If $(i, j) \notin E$, then, of course, $p_{ij} = 0$.]

Walks on graphs

A graph can model many things

Anything that can be modelled with an (undirected) graph can be modelled with a digraph, simply by replacing each (undirected) edge with two directed edges.

Depending on the nature of the system your digraph models, you may be interested in describing a walk on the graph.

Recall that a walk on a directed graph is a sequence $v_0, e_1, v_1, e_2, \dots, e_n, v_n$ of vertices alternating with edges with the property that $e_i = (v_{i-1}, v_i)$ for each i such that $1 \leq i \leq n$.

Depending again on the nature of the system your digraph models, you may be interested in a walk in which the decision of where to go next is made probabilistically (because this may model something that happens in your system)...

Random walks: definition

Let G be a digraph with n vertices $V = V(G) = \{1, \dots, n\}$ and (directed) edge set $E = E(G)$.

A square matrix with non-negative entries such that the entries in each row sum to 1 is called a **stochastic matrix**.

Associated with an n -vertex directed graph G , let $T = (t_{ij})_{1 \leq i, j \leq n}$ be an $n \times n$ stochastic matrix satisfying the rule $\forall (i, j) \notin E(G) \ t_{ij} = 0$.

For any given n , let B_n denote the set of **basis vectors** $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$ where \mathbf{e}_i is the $n \times 1$ vector with 1 as the i -th entry (i.e. in row i) and all other entries zero. E.g, for $n = 3$: $\mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

For $X_0 = \mathbf{e}_i \in B_n$ the sequence $(X_k)_{k \in \mathbb{N}^*}$ (called a **Markov chain**) specified by G and T is called the **random walk** on G starting at vertex i (or “at \mathbf{e}_i ”), with transition matrix T .

Then $X_k = (T')^k \mathbf{e}_i = (q_j)_{1 \leq j \leq n}$ say gives, for $1 \leq j \leq n$, the probability q_j of being at the vertex j after k steps, starting from vertex i .

Steady States

Let $S = (q_j)_{1 \leq j \leq n} \in \mathbb{Q}^n$ be a probability vector (all entries are non-negative and they sum to 1), which is a steady state vector for T , i.e.

$$T'S = S.$$

What does S represent in relation to our random walk? We have seen that a Markov process does not necessarily approach a steady state from every initial state. However it can be proved that, for any initial probability vector X ,

$$\frac{1}{N} \sum_{k=0}^{N-1} (T')^k X$$

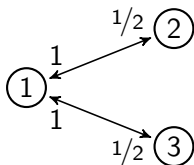
approaches S as N gets large, so, **on average, the walk is at vertex j with probability q_j , i.e. is at j $100q_j\%$ of the time.**

Example 1

Consider a graph G with adjacency matrix A and a random walk on G with transition matrix T , where

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$



Our random walker alternates between vertex 1 and the other other two vertices, with neither of these two vertices being favoured over the other.

On average, the walker is at 1 half of the time and at 2, 3 a quarter of the time each, so the steady state vector is $S = \begin{bmatrix} 1/2 \\ 1/4 \\ 1/4 \end{bmatrix}$.

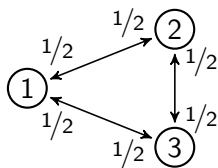
This is confirmed by checking that $T'S = S$.

Example 2

Consider a graph G with adjacency matrix A and a random walk on G with transition matrix T , where

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix}$$



We see that no one vertex is favoured over any other.

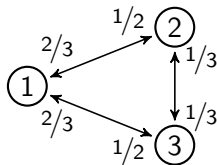
On average, the walker will spend equal time at each vertex, so the steady state vector is $S = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$.

Again, this is confirmed by checking that $T'S = S$.

Example 3

Consider a graph G with adjacency matrix A and a random walk on G with transition matrix T , where

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad T = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 2/3 & 0 & 1/3 \\ 2/3 & 1/3 & 0 \end{bmatrix}$$



Now our random walker slightly favours vertex 1 over the other two, with neither of these other two being favoured over the other.

So the steady state vector should have the form $S = \begin{bmatrix} p \\ q \\ q \end{bmatrix}$.

But what should the probabilities p and q be?

Can you guess?

Example 3 (concluded)

To find p and q we could use the method we used when investigating Markov chains earlier in the course.

That is, we could use the computer to solve the system of linear equations represented by the matrix equation

$$(T' - I)S = 0$$

modified by replacing all entries in the last row of both $T' - I$ and 0 by 1.

However, in this case the equations boil down to:

$$\begin{aligned} -p + \frac{4}{3}q &= 0 \\ p + 2q &= 1 \end{aligned}$$

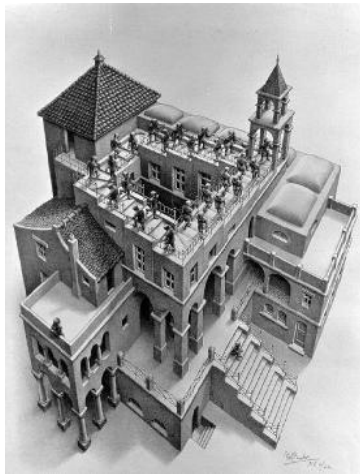
Thus $\begin{bmatrix} -1 & \frac{4}{3} \\ 1 & 2 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, so $\begin{bmatrix} p \\ q \end{bmatrix} = \frac{1}{-10/3} \begin{bmatrix} 2 & -(\frac{4}{3}) \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4/10 \\ 3/10 \end{bmatrix}$.

On average, our walker spends 40% of the time at vertex 1 and 30% at each of the other two vertices. Is that what you guessed?

The steady state vector is $S = \begin{bmatrix} 4/10 \\ 3/10 \\ 3/10 \end{bmatrix}$. As you can check, $T'S = S$.

A story about old people who lived long ago

Old timey people (such as myself) had it tough when we were young. Not only did we have to walk to school, and it was uphill both ways...



"Ascending and Descending," by M.C. Escher (1960). Lithograph.

Story continued

...but internet search engines did not sort hits very well. We often had to click "next page" many times to find the good hits to any search we ran.

Imagine a search for "Andrey Markov" returning 15,000 hits (relevant webpages), but the first page of hits is filled with 7th grade history projects and sneaky advertisements that list keywords irrelevant keyword so they can show up in searches. (I am exaggerating, but not as much as you think).

A new company named Google was launched by Larry Page and Sergey Brin in 1998 to market a new search engine called Google Search. The competitive advantage of Google Search, at least for this old timer, was the effective way it sorted the list of search hits before displaying them to the user. The "best" hits were invariably on the first or second page of hits. What was the new idea that Page and Brin used to outperform the existing search engines...they used some discrete mathematical modelling.

How can we decide which webpages are most awesome?

IDEA: Let the structure of an internet (the hyperlinks between pages) tell us which pages are most awesome.

HYPOTHESES OF THE MODEL: For a webpage W :

1. Each link to W (from some other page) is saying that W is a bit awesome;
2. A link to W from an awesome page is saying more than a link to W from a less awesome page.

The hard part...how to make this self-referential criteria operational.

First we model the structure of an internet

Represent an internet as a digraph called a **webgraph**. The vertices represent pages of the web, and there is a directed edge from vertex X to vertex Y if and only if there is a hyperlink from the page corresponding to X to the page corresponding to Y .

A webgraph is a discrete mathematical model of an internet.

How can we decide which webpages are most awesome?

Now consider a random walker (or random surfer) RS who surfs the web forever as follows:

1. RS chooses a page at random to start surfing.
2. At each step, the random surfer moves to a page on the web according to the following rule
 - 2.1 With probability α , the RS will type in the URL of a randomly chosen page (possibly even the current page)—we will call this “(unforced) teleporting.”
 - 2.2 With probability $(1 - \alpha)$ the RS will proceed as follows; if there is at least one hyperlink on the current page, the RS will choose at random one of these hyperlinks and click on it; if there is no hyperlink on the current page, then the random surfer will choose a *new* page at random and teleport there (“forced teleporting”).

The value $(1 - \alpha)$ is called the **damping factor**. A damping factor of 0.85 ($\alpha = 0.15$) is often used.

Some intuitive observations about the walk taken by the RS

Consider a webpage W :

1. The more hyperlinks that point to W , the more likely it is that the RS will be visiting W often;
2. If the RS is likely to visit a page X often, then a hyperlink from X to W will have a significant effect on how often the RS will visit W ; if the RS is unlikely to visit a page X often, then a hyperlink from X to W will not have a significant effect on how often the RS will visit W .

These observations appear to align well with our idea of how we might decide which webpages are most awesome. On average, the RS will visit an awesome page more often than a less awesome page.

PageRank algorithm

PageRank is a link analysis algorithm, named after Larry Page, co-founder of Google, used by the Google Internet search engine.

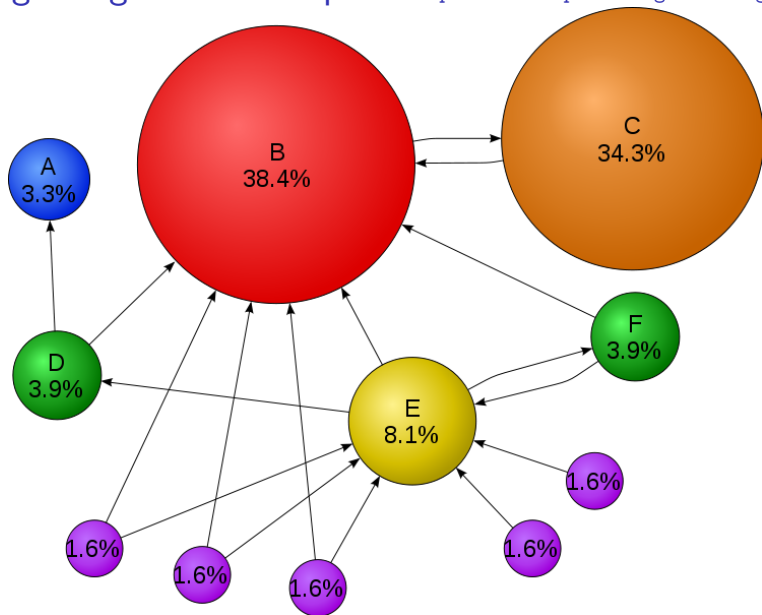
Let G be a webgraph for some internet with vertices $V(G) = \{1, 2, \dots, n\}$. Let M be the transition matrix corresponding to the movement of the random surfer (with damping factor α chosen from the interval $(0, 1)$). Let $P = (p_1, p_2, \dots, p_n)$ be the steady state vector for the random walk on W using the transition matrix M . For each $i \in \{1, \dots, n\}$, p_i is the **PageRank** of page i .

The name “PageRank” is a trademark of Google, and the PageRank process has been patented (U.S. Patent 6,285,999).

[**Source:** <http://en.wikipedia.org/wiki/PageRank>]

Google PageRank Example

<http://en.wikipedia.org/wiki/PageRank>



Building the transition matrix M

We shall build M by combining a “basic transition matrix T ” with a matrix that encodes what happens when the RS decides to move about the webgraph using links or forced teleportation, and another matrix that encodes what happens when the RS decides to move by unforced teleportation.

Basic transition probabilities

Suppose that that RS never teleports unless there are no hyperlinks on the current page. Then the RS is equally likely to follow any link on a page, and, if there are no links, is equally likely to ‘teleport’ to any other page on the web.

For each vertex i , let n_i be the number vertices to which i is linked:

$$\begin{aligned} n &= |V(G)| \\ n_i &= |\{j : (i, j) \in E(G)\}| \end{aligned}$$

Then the basic probability t_{ij} of a transition from vertex i to j is given by

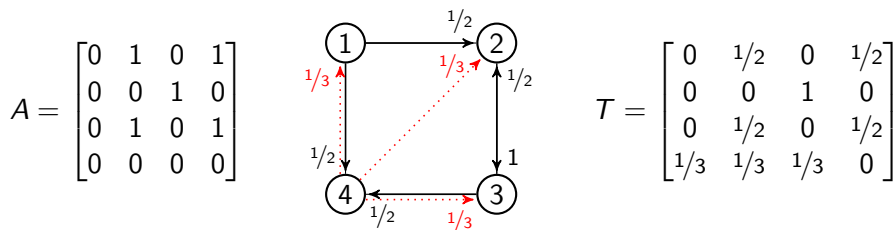
$$t_{ij} = \begin{cases} 1/n_i & \text{if } n_i \neq 0 \text{ and } (i, j) \in E(G) \\ 1/(n-1) & \text{if } n_i = 0 \text{ and } i \neq j \quad (\text{but see footnote})^1 \\ 0 & \text{otherwise} \end{cases}$$

The basic transition matrix is $T = (t_{ij})_{1 \leq i, j \leq n}$.

¹When n is large (as in the WWW) this line can be simplified to: $1/n$ if $n_i = 0$.

Example 4A

Here is a tiny example of basic transition probabilities - with just $n = 4$ vertices :



Solving, by computer, $(T' - I)S = 0$ with the usual replacement last equation, gives steady state solution $S = \frac{1}{13} \begin{bmatrix} 1 \\ 4 \\ 5 \\ 3 \end{bmatrix} \approx \begin{bmatrix} .08 \\ .31 \\ .38 \\ .23 \end{bmatrix}$.

So on this basis, vertex 3 is most important and vertex 1 least.

The damping factor $(1 - \alpha)$

Recall that the PageRank algorithm assumes that, at any time k , there is a small probability α that, irrespective of what links are available at the current page, the surfer chooses to teleport randomly to any page on the web (including the current page).

The probability that the surfer takes the unforced teleport option and lands on any particular page is α/n .

Thus the modified probability for transition from vertex i to j is

$$m_{ij} = \alpha/n + (1 - \alpha)t_{ij}.$$

In practice, Google uses a damping factor of 85%, i.e. $\alpha = 0.15$.

The modified transition matrix M and PageRank vector \mathcal{PR}

The modified transition probabilities lead to a modified transition matrix

$$\begin{aligned} M = (m_{ij})_{1 \leq i, j \leq n} &= (\alpha/n + (1 - \alpha)t_{ij})_{1 \leq i, j \leq n} \\ &= (\alpha/n)U + (1 - \alpha)T, \end{aligned}$$

where U is the $n \times n$ all-1's matrix and T is the basic transition matrix.

As indicated earlier, the PageRank algorithm defines the rank of page i of the webgraph to be the i -th entry in the **PageRank vector** \mathcal{PR} , which in turn is defined as the steady state vector for the random walk on the webgraph with transition matrix M .

Thus \mathcal{PR} is defined as the probability vector solution to the equation

$$M' \mathcal{PR} = \mathcal{PR}.$$

Calculating \mathcal{R}

Expanding the defining equation $M'\mathcal{R} = \mathcal{R}$ gives

$$\begin{aligned}\mathcal{R} &= ((\alpha/n)U + (1 - \alpha)T')\mathcal{R} && \text{since } U' = U \\ &= (\alpha/n)U\mathcal{R} + (1 - \alpha)T'\mathcal{R} && (\star\star)\end{aligned}$$

Now each entry of the product $U\mathcal{R}$ is the sum of all the entries in \mathcal{R} , and since \mathcal{R} is a probability vector that sum is 1. Hence $U\mathcal{R} = \mathbf{1}$ where $\mathbf{1}$ is the $n \times 1$ vector of all 1's. So equation $(\star\star)$ can be rearranged as

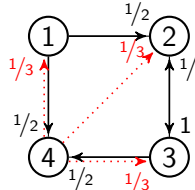
$$(I - (1 - \alpha)T')\mathcal{R} = (\alpha/n)\mathbf{1}$$

For small to moderate n this equation can be solved directly (by computer).

- Notes:
- When $\alpha = 0$ this equation reverts to the basic steady state equation.
 - When $\alpha \neq 0$ the fact that we have used $U\mathcal{R} = \mathbf{1}$ means that it is no longer necessary, nor appropriate, to replace the last row of this matrix equation by all 1's, as in the case of $\alpha = 0$.

Example 4B

For example 4A we had:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$


$$T = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 & 0 \end{bmatrix} \quad S = \begin{bmatrix} .08 \\ .31 \\ .38 \\ .23 \end{bmatrix}$$

Let's see what happens if we apply a damping factor of 90%; *i.e.* $\alpha = 0.1$.

We need to solve the equation $(I - (1 - \alpha)T')PR = (\alpha/n)\mathbf{1}$ where:

$$(I - (1 - \alpha)T') = I - (0.9)T'$$

$$= \begin{bmatrix} 1 & 0 & 0 & -0.3 \\ -0.45 & 1 & -0.45 & -0.3 \\ 0 & -0.9 & 1 & -0.3 \\ -0.45 & 0 & -0.45 & 1 \end{bmatrix} \quad \parallel \quad (\alpha/n)\mathbf{1} = \frac{0.1}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.025 \\ 0.025 \\ 0.025 \\ 0.025 \end{bmatrix}$$

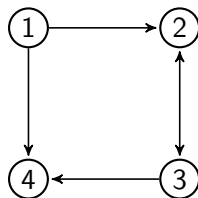
Example 4B (cont.)

Solving the equation

$$\begin{bmatrix} 1 & 0 & 0 & -0.3 \\ -0.45 & 1 & -0.45 & -0.3 \\ 0 & -0.9 & 1 & -0.3 \\ -0.45 & 0 & -0.45 & 1 \end{bmatrix} \mathbf{PR} = \begin{bmatrix} 0.025 \\ 0.025 \\ 0.025 \\ 0.025 \end{bmatrix}$$

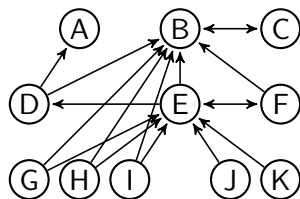
on the computer gives, to two decimal places,

$$\mathbf{PR} = \begin{bmatrix} .10 \\ .30 \\ .37 \\ .23 \end{bmatrix} \quad \text{compared to} \quad \mathbf{S} = \begin{bmatrix} .08 \\ .31 \\ .38 \\ .23 \end{bmatrix} \quad \text{without damping.}$$



The PageRank of vertex 1 increases because it now has teleporting ‘inputs’ from all vertices, not just vertex 4. This increase is at the expense of the stronger vertices 2 and 3. Vertex 4 gains about as much as it loses. This is what you expect with damping.

Example 5



At left is the Wikipedia example we saw earlier of a miniweb of 11 pages and 17 hyperlinks. Colours, variable sizes and PageRanks have been removed and the bottom five vertices have been labelled G to K, following the given labelling of the top six vertices. The layout is similar to that in the original.

Using the steady state method we have been discussing, we will derive the PageRanks given on the Wikipedia diagram.

Step 1: Compile the adjacency matrix A .

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Example 5 (cont.)

Step 2: Compile the basic transition matrix T .

In each row i of A , count the number n_i of 1's in the row then:

- If $n_i \neq 0$ replace each 1 with $1/n_i$.
- If $n_i = 0$ then
 - if n (the total number of pages) is small (less than 10 say) replace all but the i -th (diagonal) entry by $1/(n-1)$ (we did this in Example 4B)
 - but for $n \geq 10$ (as here and for WWW) replace every entry by $1/n$.

For our Wikipedia example we get

$$T = \begin{bmatrix} 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 & 1/11 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Example 5 (cont.)

Step 3: Compile the matrix $(I - (1 - \alpha)T')$

For each row i of T

1. multiply each entry by $(1 - \alpha)$ and put the *negative* of this in the corresponding position in the i -th *column* of $(I - (1 - \alpha)T')$;
2. add 1 to each diagonal entry of the resulting matrix.

Google uses an 85% damping factor, so we set $(1 - \alpha) = 0.85$.

$$(I - (1 - \alpha)T') =$$

[illegible]

Example 5 (cont.)

Step 4: Compile vector $(\alpha/n)\mathbf{1}$.

Since $(1 - \alpha) = .85$ use $\alpha = .15$.

Thus $\alpha/n = \frac{.15}{11} = .01\overline{36}$ and hence

$$(\alpha/n)\mathbf{1} = \begin{bmatrix} .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \\ .01\overline{36} \end{bmatrix}$$

Step 5: Use a computer to solve $(I - (1 - \alpha)T')PR = (\alpha/n)\mathbf{1}$

For example 'Gauss-Jordan Elimination' in the *Matrix Reshish* online matrix calculator, gives

$$PR = \begin{bmatrix} 0.032919 \\ 0.384644 \\ 0.343127 \\ 0.039112 \\ 0.080937 \\ 0.039112 \\ 0.016180 \\ 0.016180 \\ 0.016180 \\ 0.016180 \\ 0.016180 \end{bmatrix} \approx \begin{bmatrix} 3.3\% \\ 38.4\% \\ 34.3\% \\ 3.9\% \\ 8.1\% \\ 3.9\% \\ 1.6\% \\ 1.6\% \\ 1.6\% \\ 1.6\% \\ 1.6\% \end{bmatrix}$$

Iterative Approximation method

When n is huge, as it is with the WWW, solving the the $n \times n$ linear system $(I - (1 - \alpha)T')\mathcal{R} = (\alpha/n)\mathbf{1}$ becomes computationally infeasible.

A computationally simpler method starts from the defining equation:

$$M'\mathcal{R} = \mathcal{R}.$$

Recall that $M = (\alpha/n)U + (1 - \alpha)T$ is the modified transition matrix.

As with Markov chains in general we can attempt to find \mathcal{R} by iteratively calculating the chain of probability vectors P_0, P_1, \dots where P_0 is arbitrary and $P_k = M'P_{k-1}$ for $k \geq 1$ (so $P_k = (M')^k P_0$).

A steady state is reached when $P_k \approx P_{k-1}$. Then declare that $\mathcal{R} \approx P_k$.

$$\begin{aligned} \text{Now } P_k &= M'P_{k-1} = [(\alpha/n)U + (1 - \alpha)T']P_{k-1} \\ &= (\alpha/n)\mathbf{1} + (1 - \alpha)T'P_{k-1} \end{aligned}$$

and it is natural to start with all ranks equal. So the iterative scheme is

$$P_0 = (1/n)\mathbf{1}; \quad P_k = \alpha P_0 + (1 - \alpha)T'P_{k-1}, \quad k \geq 1.$$

Each iteration takes a weighted average of teleporting and hyperlinking.

Example 4C

In Example 4B we used the equation-solving method to find \overline{PR} .

For $T = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$ and $\alpha = 0.1$ we found $\overline{PR} = \begin{bmatrix} .10 \\ .30 \\ .37 \\ .23 \end{bmatrix}$ to 2d.p.

Let's try the same problem using iterative approximation:

$$P_0 = \begin{bmatrix} .25 \\ .25 \\ .25 \\ .25 \end{bmatrix} \quad P_k = \begin{bmatrix} .025 \\ .025 \\ .025 \\ .025 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & .3 \\ .45 & 0 & .45 & .3 \\ 0 & .9 & 0 & .3 \\ .45 & 0 & .45 & 0 \end{bmatrix} P_{k-1}$$

Results of the first ten iterations, rounded to 2d.p. Calcs used 15d.p.

k	1	2	3	4	5	6	7	8	9	10
P_k	.10	.10	.09	.10	.09	.10	.09	.10	.09	.10
	.33	.29	.31	.30	.31	.30	.31	.30	.30	.30
	.33	.39	.35	.38	.36	.37	.36	.37	.37	.37
	.25	.22	.25	.22	.24	.23	.24	.23	.24	.23

Pretty good after just 2 iterations! Within 1%-point after 4 iterations.

END OF SECTION D3