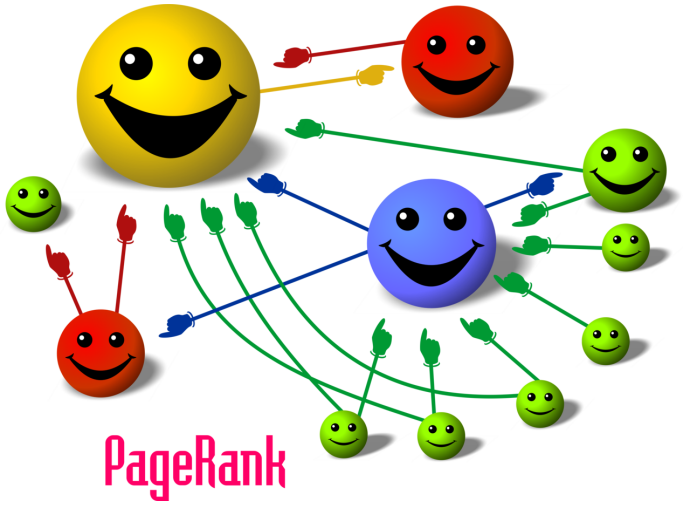
The Page Rank Algorithm
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/pagerank/pagerank.pdf

---



The size of each node reflects its importance, and is related to the importance of the nodes pointing to it. [Image from Wikipedia]

1 PageRank: a Machine Learning Algorithm

PageRank is an standard example of a machine learning algorithm. The task is to assign a ranking to every web page, which reflects some measure of the page's significance or value to a user. Such rankings were originally produced by very patient human readers, but as the Internet exploded in size, this became impractical. The PageRank algorithm simulates the human ranking process, but does so in a way that is fully automatic, extendible to arbitrarily large sets of data, and does not require in any way understanding the content of the pages, which could just as well be written in the Martian language.

2 The Search Problem

We are used to the experience of using an Internet browser to find answers to questions instantly. We don't think about the fact that, essentially, we are seeing a miracle every time. When we enter a search phrase like "banana cake", a search engine

1. out of all the various concepts that include "banana" and "cake", selects those about the popular dessert, and ignores the movie "Bananas", the rock band "Cake", the expression "going bananas";
2. out of all the web pages in the universe, identifies those that seem to have something to do with your search phrase.
3. out of all those "hits", determines a ranking of which web pages are the best, and points to the first few such pages;

4. retrieves and displays the page you finally select;

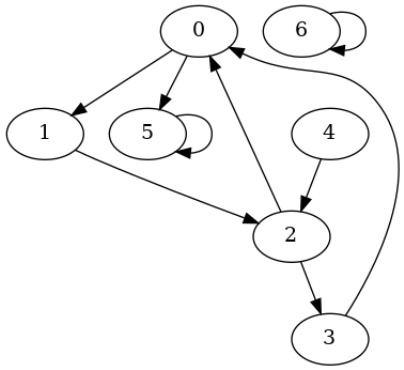
Today, we will only talk about part of item #3: Once the search engine has a list of all the hundreds of thousands of web pages that may be relevant to your search, how does it choose which pages to show first? This is the **page rank problem**.

So now let's imagine a mass of web pages all having something to do with banana cake. The browser does not read or understand this text, and yet it must decide which pages are the most likely to satisfy the user. What techniques can be used to do this?

While the ranking process is more complicated, it relies fundamentally on observing the pattern of links from one page to another. A web page may have *outlinks* that point to other pages, and it may have *inlinks*, that is, links on other pages that point to it. We might think of the inlinks as a kind of recommendation or vote of approval; some web page creator is saying that this page h as useful information.

Can we convert this vague idea into a scheme for producing rankings?

If we are only going to think about links, then we can model our problem as the study of directed graphs (often called "digraphs"). We can picture a digraph as a set of nodes, connected by directed edges. Small examples are easy to display. Here we consider a case involving just seven nodes:



Notice that there is a sort of cycle involving nodes 0, 1, 2, and 3, that node 4 has only an outlink, node 5 has only an inlink, and node 6 has no links at all.

Our first guess at how to rank these pages would be to simply count the inlinks. In that case, pages 0 and 2 have a rank of 2, pages 1, 3 and 5 have rank 1, and pages 4 and 6 have rank 0.

However, we might like to refine our rankings. After all, page 0 has two inlinks, but one is from a low rank page (3) and one is from a high rank page (2). On the other hand, page 2 is pointed to by pages of rank 0 and 1. Perhaps we should rerank pages, using the sum of the rankings from the inlinks. In that case, our rankings would now be:

```
0 1 2 3 4 5 6
3 2 1 2 0 2 0
```

and suddenly page 3 looks to have the most importance. Naturally, we think we could improve this ranking by repeating the adding procdess, but each time we do it, the numbers involved grow larger in a somewhat unruly way. Moreover, we completely ignore pages 4 and 6, which might actually have some value, however small.

We will consider three approaches:

1. a mathematical ranking process using the transition matrix of the directed graph in a power method iteration.

2. the random surfer technique, a model of how a user would actually browse the web.
3. a modification of the transition matrix to include the possibility of random jumps

All these methods try to assign a numerical ranking to every web page, but they face problems that arise when some pages are dead ends (no outlinks), some pages have no inlinks, and pages may form isolated islands of information with no connection to other islands.

3 The power method

Many physical processes can be described by a state vector x which is subject to a repeated process of change, that can be represented by multiplication by a linear operator defined by a matrix A, so that from an initial state x_0 we next have the state $x_1 = A * x_0$, then $x_2 = A * x_1 = A^2 * x_0$ and so on.

While matrix multiplication typically scrambles the values of the input vector to form the output vector, there are often some special input vectors for which this multiplication returns a copy of the input, multiplied by some value; in other words, for this input, matrix multiplication becomes simple multiplication by some scalar value λ. If we have a result like $A * v = \lambda v$, we say that v is an eigenvector of A with associated eigenvalue λ.

A matrix must have at least one such eigenvector, eigenvalue pair, and can have as many as n (the order of the matrix). These quantities can be complex. If A is symmetric, however, we are guaranteed that there will be a full set of n eigenpairs, the eigenpairs will be real, and in general the eigenvectors will be orthogonal.

While there are well known techniques, algorithms, and software for computing eigen information, we will be interested in a simple approach called the *power method*, which is used to estimate the eigenvalue of largest magnitude (called the *dominant eigenvalue* and its associated (dominant) eigenvector. The reason that this value is important is that, in general, the long term behavior of a system $x_{i+1} = A * x_i$ is that the state vector will look more and more like a multiple of the dominant eigenvector. So computing this eigenvector in advance tells us where our system is going.

If we are going to program this process, we want to be able to estimate the eigenvalue. If v is our current approximation to the dominant eigenvector, the Rayleigh quotient will estimate the corresponding eigenvalue:

$$R(A, v) = \frac{v^T A x}{v^T v}$$

which can be programmed as

```
def rayleign ( A, v ):
  lam = np.dot ( v, np.dot ( A, v ) ) / np.dot ( v, v )
  return lam
```

So a procedure for estimating the dominant eigenvalue λ and eigenvector v would be something like:

```
v = np.random.random ( size = n )
for it in range ( 0, it_max ):
  v = np.dot( A * v )
  lam = rayleigh ( A, v )
```

Assuming our matrix A is symmetric, this iteration is generally guaranteed to converge. Here is a sample computation for a random 5x5 matrix, showing the rapid convergence:

```
k     lam

0   3.881137345004205
1   4.004578900091497
```

```
     2   4.013249189004849
     3   4.014006258508431
     4   4.014080402039246
     5   4.014087983253743

(exact)  4.01408886119406
```

4 The adjacency and transition matrices for a digraph

A *directed graph* can be represented by something called the *adjacency matrix*, which we will write as A. Entries of A will be zero, unless there is a directed edge from node i to node j, in which case $A(i,j) = 1$.

Here is the adjacency matrix for the set of 7 web pages:

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

We will also be interested in something called the *transition matrix*. Given an adjacency matrix A, the transition matrix T is computed as follows:

1. copy $T \leftarrow A$;
2. if row i is completely zero, set $T_{i,i} = 1$;
3. divide each row by the sum of its entries;
4. transpose the resulting matrix.

For our problem, we have:

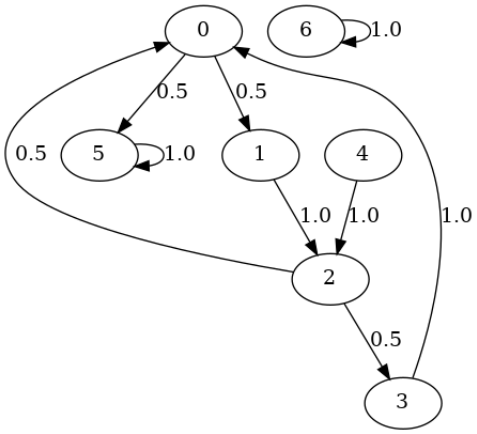
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{normalize} \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{transpose} \begin{bmatrix} 0 & 0 & \frac{1}{2} & 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = T$$

The transition matrix has several properties we will need later:

1. it is square;
2. all entries are nonnegative;
3. each column sums to 1

A matrix with these properties is often called a *stochastic matrix* or a *Markov matrix*, because it has important uses in probability, statistics, and simulation. In particular, the value of $T(i,j)$ can be thought of as the probability that a traveler making random steps along the edges of the digraph will next move from node i to node j.

Note that in cases where a node had no outlink, the transition matrix puts a 1 in the diagonal, which corresponds to a link from the node back to itself. This corresponds to the role of such a node as a dead end. Here is a plot of the transition digraph, with the transition probabilites labeling each edge.



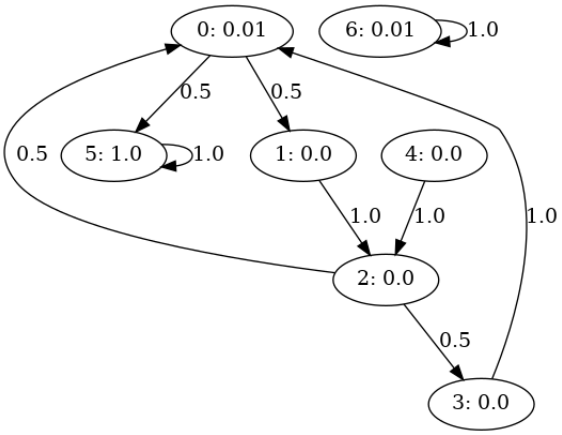
5 Applying the power method to the transition matrix

We are trying to compute an importance ranking for our example set of web pages. The transition matrix T allows us to represent a sort of flow of importance from one page to the next, where each link represents a vote of confidence by the source for the destination page. If we can find an eigenvector vector v such that $Tv = v$, then we may have found what we are looking for: v_i would be the relative importance or ranking of page i.

In doing this, we are relying on a version of the Perron-Frobenius theorem:

The Perron-Frobenius Theorem (strictly positive transition matrices):
If A is a square matrix whose entries are all strictly positive, and each of whose columns sums to 1, then the largest eigenvalue is 1. This eigenvalue is simple. All other eigenvalues are strictly smaller in norm. There is a corresponding eigenvector whose entries are all strictly positive.

Does it matter that some of the entries of T are not "strictly" positive? We try the power method, using a random starting vector, taking just 20 iterations, and see what happens. In the following plot, each node is now labeled with its "importance" or ranking, which is simply the corresponding eigenvector value, normalized so that the maximum value is 1. What can we conclude?



The result is rather odd. Almost all the nodes have zero importance, except for the nodes formerly known as 5 and 6. These two nodes are distinguished by the fact that they are dead end nodes. Node 5 seems to

be the most important, while 6 is rather important, and the others have no importance at all. Can we talk this through and see what the power method is telling us?

Imagine that nodes are rooms in an art gallery, and that we start with 100 people in each room. Every minute, the viewers get restless and move to an adjoining room. That's all we need to visualize in order to explain what happens. The people in room 6 can never leave, and no one can enter, so that population stays constant. The people in room 5 can also never leave, but new people might enter. And in fact, the people in all other rooms will wander aimlessly for a while, but eventually happend to choose room 5, so they are stuck. The people in six rooms all end up in room 5, so eventually, the power method reveals that room 5 is the most important, followed by room 6.

6 Adding random surfer jumps

The power method has played the game the way we asked it, but the answer is not what we want. We really didn't expect this result, and we can't do much with it. Instead of thinking in terms of an art gallery, let's be a little more realistic and suppose that an Internet user is browsing the pages. Certainly, it's likely that for a while, the user will move by clicking on a link on the current page, and we might as well assume these choices are random. But if a user reaches a page with no outlinks, do they simply sit and stare at it for eternity? No, of course not. They eventually pick another page to look at, and we might as well assume that that is also a random choice. And in fact, we might as well assume that, after viewing any of the pages, there is some probability p that the viewer will decide to jump to a random page.

That means that on each step of the viewing process, the user will either

1. with probability $(1 - p)$, choose an outlink at random;
2. with probability p, jump to one of the n pages at random;

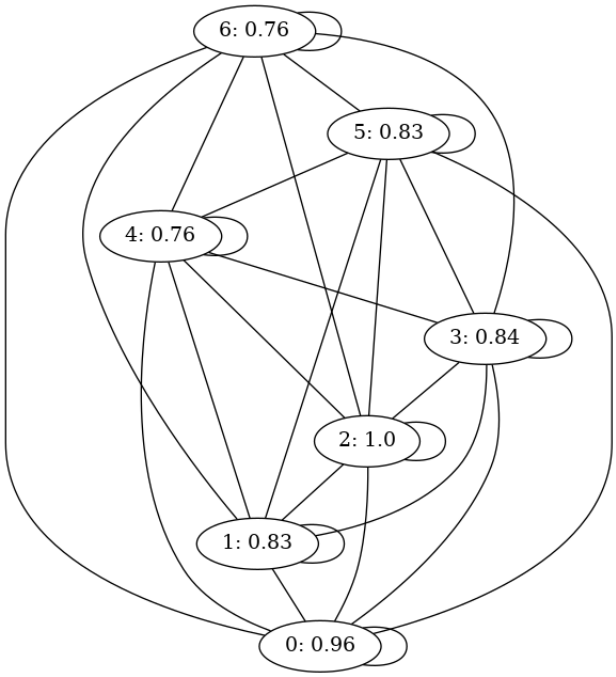
To model this behavior, we need to make a new "surfer" matrix S

1. compute T as before;
2. compute E, an $n \times n$ matrix, all of whose values are $\frac{1}{n}$;
3. compute $S = (1 - p) * T + p * E$;

7 The power method with jumps

It is common to use the value $p = 0.15$ for the probability of jumping. Noticed that now our matrix has no zero entries. Therefore, it satisfies the strong version of the Perron-Frobenius theorem, which requires that there is a path connecting every pair of nodes in the digraph:

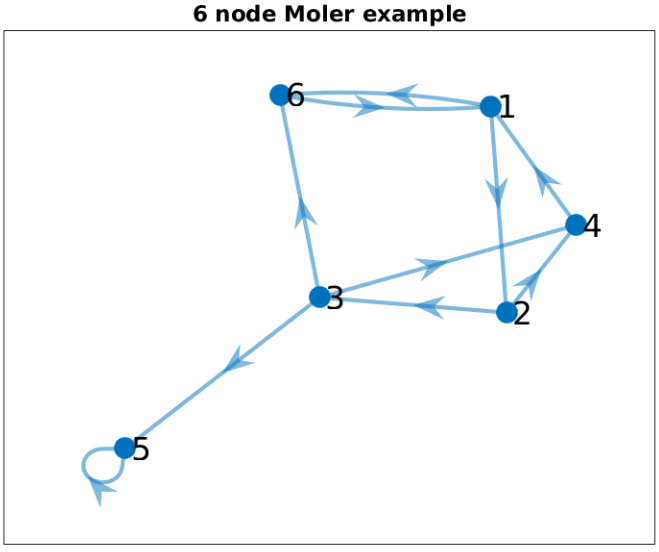
The Perron-Frobenius Theorem (connected digraph):
If A is a square matrix whose entries are all nonnegative, and the matrix is irreducible, and each of the columns sums to 1, then there is an eigenvalue equal to 1, and no eigenvalue is greater than 1 in absolute value. The eigenvalue equal to 1 is simple. There is a corresponding eigenvector whose entries are all nonnegative.



Now we see a more reasonable set of rankings for the nodes. Adding random jumps means every page has some chance of being viewed. The arrangement of outlinks means that some nodes, such as 0 and 2, get the benefit of extra viewers being directed there.

8 A tiny network

Consider this 5-node system studied by Cleve Moler:



A 5-node (not simply connected!) network.

Using the transition matrix T, we take 100 power method steps, and then look at the next iterates.

	r100	r101	r102
1:	0.011	0.011	0.010
2:	0.005	0.005	0.005
3:	0.003	0.003	0.003
4:	0.004	0.004	0.004
5:	0.970	0.971	0.972
6:	0.007	0.006	0.006

The ranking for node #5 is increasing, and the others are dropping, again, because this node has no outlinks. But if we add random jumps to our matrix, we get a more balanced behavior:

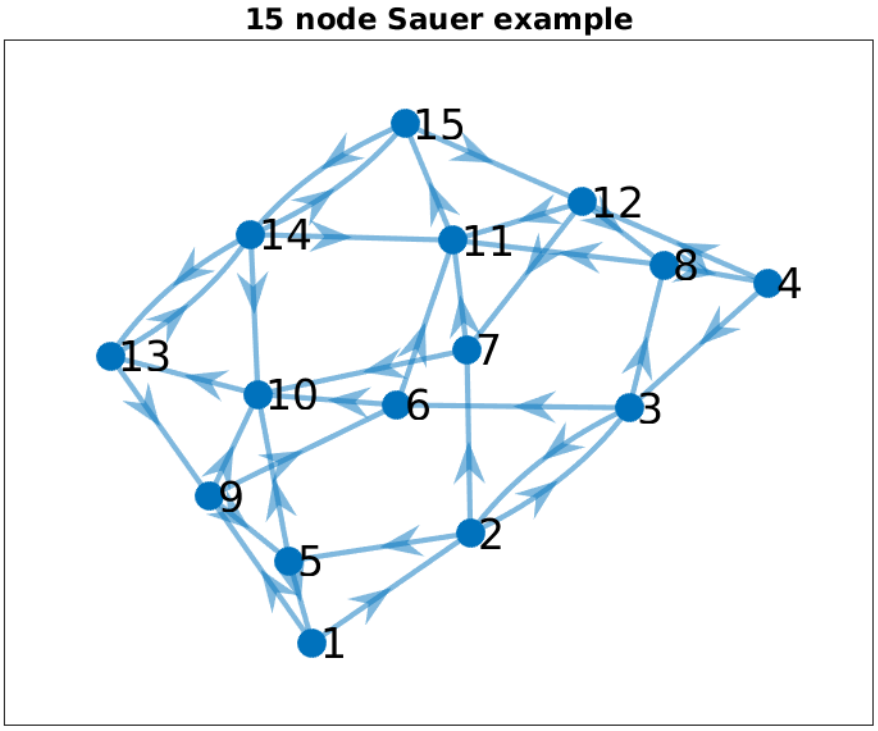
	r100	r101	r102
1:	0.235	0.235	0.235
2:	0.124	0.124	0.124
3:	0.078	0.078	0.078
4:	0.100	0.100	0.100
5:	0.314	0.314	0.314
6:	0.147	0.147	0.147

The first thing to note is that the iteration has converged. Node #5 no longer traps all users. The random jumps allow them to leak away to other nodes. Secondly, node #5 still has very high rank, but node #1 has comparable ranking, based roughly on the sum of the rankings of nodes #4 and #6. While having no outlinks seems to favor node #5, it is no longer the black hole that it represented before. Now we have a more uniform sampling of the network, and a mathematically-based estimate of the page rankings.

9 Sauer's internet

In the textbook, *Numerical Analysis* by Timothy Sauer, the following adjacency graph is shown:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1:	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2:	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0
3:	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
4:	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
5:	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
6:	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
7:	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
8:	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
9:	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0
10:	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
11:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12:	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
13:	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
14:	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1
15:	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0



A plot of Sauer's network.

Now we can apply several methods to assign a rank to each of the nodes. From the adjacency matrix A, we can compute the transition matrix T, and the random surfer matrix S, using a random jump probability of $p = 0.15$.

Sauer's matrix does not have any dead-end nodes, so if we prefer, we can first try computing the rankings using the transition matrix. Here are the results using 100 steps of the power method on T:

```
#        power

 1:      0.0154
 2:      0.0115
 3:      0.0115
 4:      0.0154
 5:      0.0308
 6:      0.0308
 7:      0.0308
 8:      0.0308
 9:      0.0810
10:      0.1100
11:      0.1100
12:      0.0810
13:      0.1467
14:      0.1467
```

```
15:       0.1467
```

If we add the surfing option, then we should expect that this will smooth out the results, reducing the highs and lows in the rankings.

```
             power        surf

 1:       0.0154     0.0259
 2:       0.0115     0.0303
 3:       0.0115     0.0301
 4:       0.0154     0.0271
 5:       0.0308     0.0392
 6:       0.0308     0.0395
 7:       0.0308     0.0391
 8:       0.0308     0.0403
 9:       0.0810     0.0738
10:       0.1100     0.1056
11:       0.1100     0.1070
12:       0.0810     0.0745
13:       0.1467     0.1247
14:       0.1467     0.1164
15:       0.1467     0.1265
```

From these results, we can see that the power and surfer methods essentially agree on the rankings, although the surfer method reduces the variation.

We have analyzed both methods by constructing a matrix and applying the power method to seek an eigenvector. To deal with the Internet, we cannot afford to create an actual matrix. Instead, we have to try to approximate the behavior of the power method using a much simpler approach.

Actually, we can do a sort of surfer simulation, in which we pretend to be a very very patient web surfer, who intends to try to see everything on the web, one random page at a time. Pages that come up more often in the random search will be assumed to be more important, because more outlinks pointed to them.

In other words, start at a random page and set its count to 1. If the page has no outlinks, then jump to another page. Otherwise, with probability p take a random number or with probability (1-p) chose an outlink. Increment the count on this page, and repeat the process. By taking a very very large number N of such steps, you can browse the Internet. You will return to some pages many times, and to others not at all. Estimate the ranking of each page by dividing its count by N. This is essentially what the PageRank algorithm does.

As we said at the beginning, ranking web pages is simply one part of a search engine's job. But when this algorithm was invented, and made effective, it gave Google an initial boost past all the search engine companies whose ranking methods were far inferior and limited.