PageRank Explained with Javascript

Clear Graph

Damping Factor: 0.825

Page 1

[x]

0.033

Page 2

[x]

Page 3

0.384

0.384

Page 3

[x]

Page 2

0.344

0.344

Page 4

[x]

Page 1

0.019

Page 2

0.019

0.039

Page 5

[x]

Page 2

0.027

Page 4

0.027

Page 6

0.027

0.080

Page 6

[x]

Page 2

0.019

Page 5

0.019

0.039

Page 7

[x]

Page 2

0.008

Page 5

0.008

0.016

Page 8

[x]

Page 2

0.008

Page 5

0.008

0.016

Page 9

[x]

Page 2

0.008

Page 5

0.008

0.016

Page 10

[x]

Page 5

0.016

0.016

Page 11

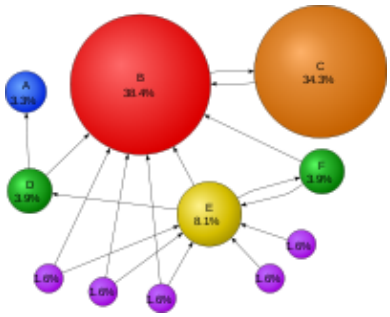
[x]

Page 5

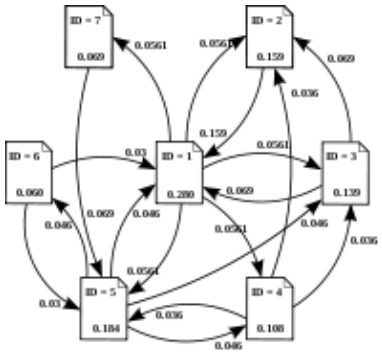
0.016

0.016

- **Wikipedia Example 1**



- **Wikipedia Example 2**



The above graphic describes a network of web pages. Each page lists its links to other pages. You can load example networks based on images from the Wikipedia article on PageRank or you can click Clear Graph and then add your own pages and links.

Page 7

Page 2

0.008

Page 5

0.008

0.016

To add a page, click on any empty space in the above *page space*. This creates a page represented by the **page icon** you see to the left.

To link from one page to another, click on the page icon that you want to create a link from, followed by the page icon you want to create a link to.

Pages are draggable, so feel free to move them around.

Double-clicking on a page will remove it from the graph.

You'll notice that there are numbers at the bottom of each page icon. This represents the PageRank of each page. It is updated in real time to reflect the changing state of the network. In addition, the rest of the article updates to reflect the changing state of the graph.

If you scroll down the page the bottom portion of the graph remains and sticks to the top of the browser viewport. This allows you to quickly show the graph at any point by clicking Show Graph. This button only appears once the graph has scrolled out of view.

Keep this dynamic functionality in mind while reading through the article as it may help you visualize what is going on.

What is PageRank?



PageRank relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. In essence, Google interprets a link from page A to page B as a vote, by page A, for page B. But, Google looks at more than the sheer volume of votes, or links a page receives; it also analyzes the page that casts the vote. Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important".



Before we start...

We will need to familiarize ourselves with some of the language used herein.

What is a graph?

A graph is an abstract representation of a set of objects that are connected by links. The objects are referred to as **nodes** (or vertices) and their connections as **edges**.

There are two main types of graphs. Undirected graphs and directed graphs. An undirected graph is one where the edges between two nodes are considered the same. We are not interested in that type of graph for our discussion.

We are concerned with **directed graphs,** that is, graphs where the connections have a specific direction, like in the image to the left.

Nodes that have no outgoing edges are called **sinks**, and nodes that have no incoming edges are called pumps.

In our specific case, we are considering ***web pages*** *to be the* ***nodes*** and the ***links*** *from one page to another to be the* ***edges***. We will also deal with pages that act as sinks, that is, pages that have no outbound links.

Our graph is a representation of a network of web pages and how they are linked to one another.

And what does this have to do with PageRank?

PageRank is a certain mathematical property of the graph that describes a network of pages and their links. We can't really understand this concept unless we are directly dealing with the mathematics involved.

There are a number of mathematically equivalent ways to solve for the PageRank of a graph representing the link structure of a network of web pages. A convenient approach is to use matrices.

When using matrices to compute PageRank we are measuring the eigenvector centrality of a modified matrix based on the network.

Whoa, that just got pretty dense. Don't worry, we'll wrap our heads around this stuff pretty quickly. Let's make sure we don't water anything down or shy away from using the proper *language*.

Adjacency Matrix

In order to do any sort of computation we have to describe our network of interconnected web pages in some sort of mathematical language. Amongst others, we will be using the language of **matrices**. The branch of mathematics that deals with matrices is called linear algebra.

If you are so inclined, MIT has graciously made available a great video lecture series on linear algebra taught by Gilbert Strang.

What is a matrix?

Simply put, *a matrix is a **rectangular array of elements***. It has a number of **rows** and a number of **columns**. Matrices are used for a number of different purposes and are a cornerstone of computational mathematics. Almost every advanced topic in computer science, from 3D graphics, to AI, to network analysis like PageRank involves matrices.

Matrices started out as a way to solve systems of linear equations but once the language itself was developed and detached from this original application, it expanded and grew, allowing for other applications to arise, as people saw fit.

One of the beauties of mathematics is the constant dance between **applied mathematics** and **pure mathematics**. We could get in to a debate about which one came first, but there is no doubt that sometimes numbers and patterns are used to describe the world and sometimes numbers and patterns are used to describe other numbers and patterns. The wonderful part is when rules that were developed under the auspices of pure mathematics find themselves once again applicable to a situation in the real world.

This is definitely the case with the PageRank algorithm.

What does a matrix look like?

Basically, a rectangle.

To the left we have a **2x2 matrix**, containing four elements. *It is a 2-dimensional square matrix.*

We'll come back to this matrix when we are discussing a very important property of matrices, their **eigenvectors**.

Let's construct a matrix of our own that is specific to our network, where the rows and columns represent the pages that we are analyzing.

Here is the resulting 11x11 square matrix:

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0	0	0
5	0	1	0	1	0	1	0	0	0	0	0
6	0	1	0	0	1	0	0	0	0	0	0
7	0	1	0	0	1	0	0	0	0	0	0
8	0	1	0	0	1	0	0	0	0	0	0
9	0	1	0	0	1	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	0	0
11	0	0	0	0	1	0	0	0	0	0	0

For example, Page 1 has no links to other pages. Page 2 has a link to Page 3.

This grid is what we will call the adjacency matrix of the graph that describes our network of web pages.

Now that we have described this network in the language of linear algebra, we need to further translate it in to a computer programming language. Luckily, the majority of languages have linear algebra libraries. In this example we're using a Javascript library called Sylvester. It has a number of classes that allow us to model matrices and vectors in any number of dimensions.

In programming terms, a matrix is similar to an **array of arrays**.

?

```
1 adjacencyMatrix = $M([
2     [0,0,0,0,0,0,0,0,0,0,0],
3     [0,0,1,0,0,0,0,0,0,0,0],
4     [0,1,0,0,0,0,0,0,0,0,0],
5     [1,1,0,0,0,0,0,0,0,0,0],
6     [0,1,0,1,0,1,0,0,0,0,0],
7     [0,1,0,0,1,0,0,0,0,0,0],
8     [0,1,0,0,1,0,0,0,0,0,0],
9     [0,1,0,0,1,0,0,0,0,0,0],
10    [0,1,0,0,1,0,0,0,0,0,0],
11    [0,0,0,0,1,0,0,0,0,0,0],
12    [0,0,0,0,1,0,0,0,0,0,0]
13]);
```

For the remainder of this article, we will be adding functions to Sylvester's Matrix class.

Row Stochastic Matrix

Since PageRank can also be thought of as a representation of how likely it is that a person randomly clicking on links well end up on a certain page, we need to convert our matrix in to a representation that represents that probability. "Stochastic" and "probability" are interchangeable terms within the realms of our discussion.

For example, since Page 5 has 3 links on it, *there is a **1 in 3 chance** that we will randomly click on any of one of the links.*

This type of matrix is known as a stochastic matrix, and in our case, we are looking for a row stochastic matrix. That is, we want every row to add up to 1. We are turning our representation of a network of links to a form that takes in to account the probabilities that a random user will browse to a certain page.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1.00	0	0	0	0	0	0	0	0
3	0	1.00	0	0	0	0	0	0	0	0	0
4	0.50	0.50	0	0	0	0	0	0	0	0	0
5	0	0.33	0	0.33	0	0.33	0	0	0	0	0
6	0	0.50	0	0	0.50	0	0	0	0	0	0
7	0	0.50	0	0	0.50	0	0	0	0	0	0
8	0	0.50	0	0	0.50	0	0	0	0	0	0
9	0	0.50	0	0	0.50	0	0	0	0	0	0
10	0	0	0	0	1.00	0	0	0	0	0	0
11	0	0	0	0	1.00	0	0	0	0	0	0

However, what happens if there is a page that acts as a sink, that is, it has no outbound links? How do these **dangling pages** affect the calculation?

If we randomly stumble across the rare web page that has no links to other pages we will assume that we have equal probability to end up on any of the other web pages in the network. That is, we will assume that there are links to every page from a page that has no links.

In order to be fair to the other pages that do have links, these random transitions are added to all pages in our network, with an additional residual probability called a **damping factor**.

This damping factor is included to increase the probability that we'll end up at a page with more network centrality and counter some of the abuses that spammers could use to game this algorithm.

Neither adjustments made for dangling pages or damping factors are necessary for computing network centrality, but they are indeed properties of PageRank and are what differentiates and makes the algorithm unique.

Here is our function that computes the row stochastic matrix:

```
?
1
2
3 Matrix.prototype.row_stochastic = function(damping_factor) {
4
5     var row_length = this.elements[0].length;
6     var d = (1 - damping_factor) / row_length;
7
8     var row_total = [];
9
10    for (var x = 0; x < row_length; x++) {
11        row_total.push(0);
12        for (y = 0; y < row_length; y++) {
13            row_total[x] += this.elements[x][y];
14        }
15    }
16
17    var a1 = this.elements.clone();
18
19    for (var x = 0; x < row_length; x++) {
          for (var y = 0; y < row_length; y++) {
```

```
20                  a1[x][y] = a1[x][y]/row_total[x] + d;
21              }
22              else {
23                  a1[x][y] = (1/row_length) + d;
24              }
25          }
26      }
27
28      return $M(a1);
29
30  }
31
```

As we can see on line 4, we compute our additional probabilities of moving from one page to another as a variable called d, which is based on a damping factor and the total number of web pages.

Next, we iterate through each row and store an array of the sum of all elements in the row, as shown on lines 8 - 13.

Finally, we again iterate through each row, this time dividing each element by its row's total, fulfilling our stochastic requirements. We also add our variable d to not only pages that have no outbound links (and therefore are assumed to link to all pages), but to every page, in order to be fair to the pages that do have links. This is done on lines 17 - 26.

When using a damping factor of 0.825 we get the resulting matrix:

	1	2	3	4	5	6	7	8	9	10	11
1	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
2	0.02	0.02	1.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
3	0.02	1.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
4	0.52	0.52	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
5	0.02	0.35	0.02	0.35	0.02	0.35	0.02	0.02	0.02	0.02	0.02
6	0.02	0.52	0.02	0.02	0.52	0.02	0.02	0.02	0.02	0.02	0.02
7	0.02	0.52	0.02	0.02	0.52	0.02	0.02	0.02	0.02	0.02	0.02
8	0.02	0.52	0.02	0.02	0.52	0.02	0.02	0.02	0.02	0.02	0.02
9	0.02	0.52	0.02	0.02	0.52	0.02	0.02	0.02	0.02	0.02	0.02
10	0.02	0.02	0.02	0.02	1.02	0.02	0.02	0.02	0.02	0.02	0.02
11	0.02	0.02	0.02	0.02	1.02	0.02	0.02	0.02	0.02	0.02	0.02

Things have very suddenly become very abstract. We've gone from one representation, our graph of a network of web pages, through to a matrix representation of that graph, and ended up with a matrix that takes in to account all sort of peculiar probabilities related to the likelihood of a user ending up on a given page.

It gets confusing because we're representing something that we can think of in a number of ways, as a graph, a matrix, or a set of probabilities. In turn, our use of language jumps between **graph language**, **matrix language**, and **probability language** quite rapidly.

This aspect of mathematics has always been the most difficult to deal with. A lot of the times the principles that are being discussed are easy to understand, but the language that mathematicians use to talk about these principles are arcane to say the least. Don't feel bad. A lot of mathematicians are only fluent in whatever specific branch they are familiar with and have just as hard a time figuring out the languages of branches they are unfamiliar with!

Eigenvector Centrality

This is where things start to get magical. It as also the hardest part to visualize. Every step so far has made a good amount of sense.

PageRank is a measure of network centrality with some tweaks made that are specific to web browsing, as we've discussed above. Since we are pretty much only dealing with matrices at this point we'll refer to this as eigenvector centrality.

This eigenvector centrality is a property of our initial graph of the network of web pages. Our previous steps, along with quick transposition of our matrix, lead us to our penultimate step towards computing PageRank.

What is a vector?

At the most abstract level, a vector is nothing more than a member of a vector space.

A vector in 2D vector space, called a **euclidean vector**, is a geometric object that has a magnitude and a direction. In the case of 2D computer graphics, the relationship of pixels to one another in an image can be considered vectors.

Vectors are **acted upon** by matrices. We can also think of a matrix as being somewhat like a function that we can apply to a vector space.

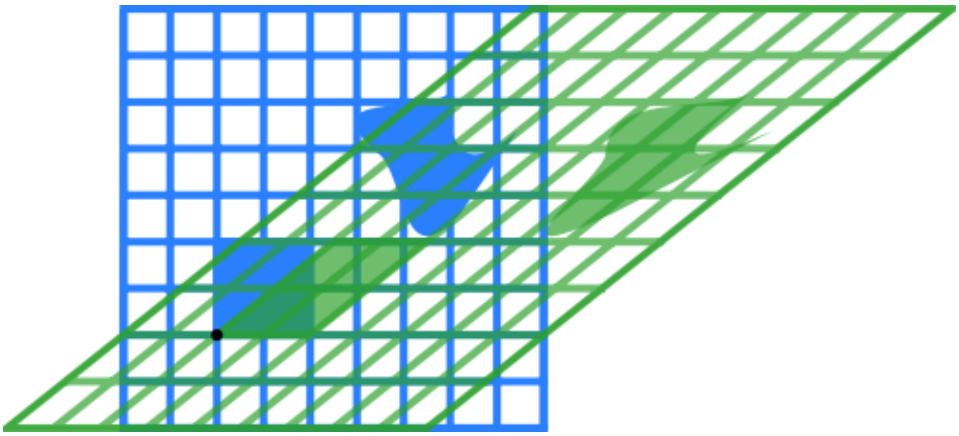
Here is the same 2x2 matrix that we saw above. It is an example of a ***shear mapping***.

Matrices are used for a lot of different purposes. They are heavily used in the realms of 2D and 3D graphics.

Italic fonts are also an example of a shear mapping!

One way that matrices are used is for performing **linear transformations**.

If you've ever used the transform tool in Photoshop, *you are watching linear algebra in operation.*



This is the result of applying the above 2-dimensional matrix to a set of pixels in a **vector space**. Matrices are always applied to vector spaces.

What is an eigenvector?

When we apply our matrix to a set of pixels, it scales all of the pixels to the right by a factor of 1.25. Notice that the horizontal relationships of the pixels do not change at all. Or, in other words, the image is scaled by 1.25 along the x-axis and left the same in the y-axis.

Eigenvectors are the vectors in a space that when acted upon by a matrix have only their magnitude affected, and not their direction. In the case of our sheer mapping, vectors that are perfectly horizontal are eigenvectors, as their direction is not changed. Again, in terms of pixels, the relationship between pixels are not affected in a horizontal direction, but are in all other directions.

Technically, an eigenvector is a vector that satisfies the equation . It is a vector x, that when acted upon by a matrix A, results in the vector x multiplied by a scalar λ. The vector x is the **eigenvector** and the scalar λ is the associated **eigenvalue**.

There are an infinite amount of eigenvectors in this case, as all horizontal vectors are eigenvectors. For the remainder of the article we will refer to all eigenvectors with the same direction as just a single eigenvector, or a unit eigenvector. We are interested in finding the unit eigenvectors, of which there will sometimes be many.

However, the matrix that we've constructed is not a 2x2 matrix and it doesn't exist in a 2D vector space. *In inhabits a vector space that is purely abstract*, so the concept of an eigenvector is harder to visualize.

A vector in a stochastic vector space is called a stochastic or **probability vector**. It is a vector that adds up to 1 and describes the likelihood of certain events taking place. The eigenvector that we are going to compute is a probability vector. It describes the probabilities of ending up at certain web pages based on the link structure of our network, no matter what page we start on. The concept of *direction* that we discussed when referring to the above linear transformation loses it's ordinary meaning.

Instead of, as in a 2D space where applying a matrix to a set of vectors alters the literal direction that they point in, we can think of this matrix as "directing" people clicking on links towards certain pages. The **eigenprobability** of our modified adjacency matrix is the PageRank.

Phew!

How do we calculate the eigenvector?

There are a number of approaches that we can use to solve for the sought after eigenvector of our matrix. Some are more accurate, some are faster, some will find more eigenvectors than others.

We are only looking for the **eigenvector associated with the dominant eigenvalue**. We don't need to know about any of the other unit eigenvectors. We want to be able to find our eigenvector as *fast as possible*.

Luckily, there's a perfect algorithm that meets our criteria..

Power Iteration

The **power iteration** method is a simple yet powerful algorithm.

1. We start with a vector. It can be any vector with at least one non-zero element.
2. We multiply this vector by our matrix.
3. We take the resulting vector, normalize it, and then multiply it by our matrix again.
4. We repeat step 3 until we've reached an acceptable amount of tolerance for error.
5. The resulting vector is an approximation of the eigenvector that we were looking for.

It is deceptively simple, but can we see how it relates to our concept of an eigenvector?

We start with a vector, any vector. We subject it to the matrix, and slowly but surely, it starts to point in a direction. It starts to converge. We're finding what vector the matrix directs all vectors towards.

What do our vectors represent? They represent the likelihood of being on a web page and ending up on another web page. Why? Because that's how we set everything up. We started with a description of our network. We made a graph showing what links to what. We then modified that graph to represent the probabilities of clicking on links randomly. We're now taking that entire structure, and all at once, figuring out *where it is pointing*.

Why do we want to normalize our vectors? We want them to add up to 1 so we can talk about them as fractions of a single whole, something that we tend to find very familiar.

How do we normalize a vector?

A **normalized vector** is a vector that is divided by its length creating a **unit vector**. This is the same as saying that *all elements in the vector add up to 1*.

```
Vector.prototype.normalize = function() {

    var row_length = this.elements.length;
    var t = 0;

    for (var i = 0; i < row_length; i++) {
        t += this.elements[i];
    }

    return this.multiply((1.0/t));
}
```

Why use power iteration?

The power iteration method will only find the eigenvector associated with the largest eigenvalue in absolute terms. Also, it will fail unless the matrix is sufficiently sparse. Luckily, those terms match our current needs perfectly. Compared to other **eigensolvers**, the power method is less computationally intensive than performing a full decomposition.

Also, it is very easy to implement and to explain.

Here's the code:

```
Matrix.prototype.eigenvector = function() {

    var tolerance = 0.000001;

    var row_length = this.elements[0].length;

    var a = [];

    for (var i = 0; i < row_length; i++) {
        a.push(1);
    }

    var x = $V(a);

    var c_old = 0;

    for (var i = 0; i < 100; i++) {
        var x_new = x.normalize()
        var c_new = x_new.elements[0];

        var e = 100 * (c_new - c_old)/c_new;
        if (Math.abs(e) < tolerance) {
            break;
        }

        x = this.multiply(x_new);
        c_old = c_new;
    }
```

```
30      return $V(x);
31  }
32
33
```

On line 3 we set our tolerance, which we use to in our iteration as our margin of error.

Next, we construct out initial vector, by first filling an array a with 1s and then converting it to our Vector object, x, as shown on lines 5 - 13.

Then, we set up our variable c_old, which we use in relation to our margin of error.

On lines 17 - 28, we loop through the meat of the algorithm. We loop 100 times in case the eigenvector doesn't converge to within our margin of error, which sometimes happens. Line 26 contains the most important aspect, the iterative multiplicationof our matrix times our arbitrary vector. Lines 19 - 24 check to see if we are within our margin of error, and if so, halt the loop.

Finally, we return our vector, which has at this point converged to within a reasonable tolerance of our sought after eigenvector.

Which leads us to...

Our final equation, which brings all of the separate steps together:

```
1
2  Matrix.prototype.pagerank = function() {
3      var damping_value = Pages.dampingFactor;
4      var row_stochastic_matrix = this.row_stochastic(damping_value);
5      var transposed_matrix = row_stochastic_matrix.transpose();
6      var eigenvector = transposed_matrix.eigenvector();
7      var normalized_eigenvector = eigenvector.normalize();
8      return normalized_eigenvector.elements;
9  }
```

On line 2, we get our damping factor from Pages.dampingFactor. This variable is set whenever the examples are loaded as well as whenever the slider in the accompanying graph is moved.

On line 4, there is one additional step, that of transposing our matrix. We could have just set up our matrix in a column based manner instead of a row based manner and skipped this step, but it works better for illustrativepurposes as well as being the most common way that stochastic matrices are presented.

Wait, what? What just happened here? Why do we need to use this language? Seriously, why all the talk of graphs and vectors?

Mathematics is made up of a large number of languages. Each language carves out it's own prioritiesand it's own set of rules. We've got algebra, trigonometry, differentialsand integrals, logarithms, and these are just a few of the familiar languages.

The amazing thing is that they can be used with one another. We can find the integral of a function that comes from trigonometry. We can solve for a variable described by an exponent. We can mix and match and reframe our question in an infinite number of ways and the entire time, if we're careful, we can be sure that techniques used and the inherent truths of each language still apply.

Mathematics is almost in one way an unraveling of a tautology. Of course there is a 1/3 chance of picking out any random apple from a group of three of them. Of course there is a direct relationship between the surface area of a sphere and the volume contained by that surface.

Starting with these principles, these almost outright obvious truths, we've discovered certain key principles to the systems that we've invented. We've discovered many ways to manipulate our patterns within these systems, promoting or ignoring certain characteristics to meet the desired goals.

Mathematics is our language of truth. It doesn't lie, yet at the same time, it can be made to model just about anything we can throw at it. It is important to understand that.

Wait, isn't PageRank a search algorithm?

No, its not!

All that PageRank does is order pages by the combined authority of all pages on the Internet. It doesn't have anything to do with keyword lookups, text indexing, or anything else that is pretty much a requirement for searching for things in the way we're used to.

PageRank is just an important aspect of Google's search algorithm and is one of the primary methods related to ordering search results.

How can I use this to drive traffic to my website?

If you take anything from this article take with it the spirit of sharing knowledge. Knowledge is the only type of content that is truly valuable and there should be no doubt that search engines now and of the future will try their best to make sure it ends up at the top of their search results.

Thanks

Let me know what you need help building.

