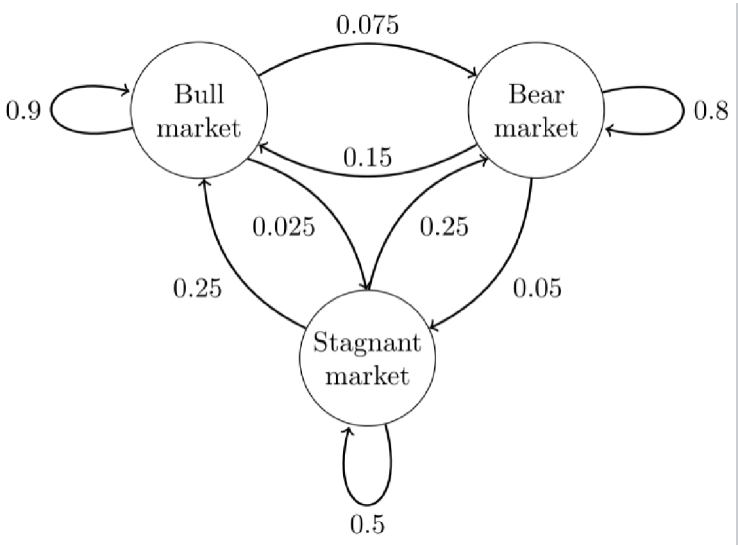
Markov Chains and Google's PageRank

Suppose that we have been observing a (hypothetical) stock market for some time and notice that the state of the market appears to be driven (almost entirely) by the current state of the market. Naturally, there will be other factors at play, but this might nevertheless provide a semi-decent model to study.

The states represent whether the market is a "bull market" (going up), a "bear market" (going down), or is stagnant. Label the state space **1 = bull, 2 = bear and 3 = stagnant.** *Each time step in this model will represent a week.*



Question 1. Input the transition matrix for this system. Make sure that your columns add to one. Call your matrix T.

	bull	bear	stagnant
bull	0.9	0.15	0.25
bear	0.075	0.8	0.25
stagnant	0.025	0.05	0.5

```
(*Define the transition matrix T*)
T = {{0.9, 0.15, 0.25}, {0.075, 0.8, 0.25}, {0.025, 0.05, 0.5}};
```

Question 2. Suppose that the current/initial state of the market is "stagnant." Find the probability of being in each state when *time = 2* (weeks). *Note:* To multiply a matrix use '.' rather than *.

```
(*Define the initial vector timeZero*)
timeZero = {{0}, {0}, {1}};

(*Perform matrix multiplication*)
timeOne = T.timeZero;
timeTwo = T.timeOne;

(*Display the results*)
timeZero
timeOne
timeTwo
```

Out[•]=
```
{{0}, {0}, {1}}
```

Out[•]=
```
{{0.25}, {0.25}, {0.5}}
```

Out[•]=
```
{{0.3875}, {0.34375}, {0.26875}}
```

Question 3. Consider larger and larger values of *time.* (Use **MatrixPower[A, n]**) What do you observe?
We get the same results!

In[•]:=
```
(*Repeat calculation with MatrixPower*)
timeOneWithPower = MatrixPower[T, 1].timeZero
timeTwoWithPower = MatrixPower[T, 2].timeZero
```

Out[•]=
```
{{0.25}, {0.25}, {0.5}}
```

Out[•]=
```
{{0.3875}, {0.34375}, {0.26875}}
```

Question 4. Find the eigenvalues and eigenvectors of T. You can use the function **Eigenvalues[A]** and **Eigenvectors[A]**. If you only want to consider the first element of the array, you can put **[[1]]** at the end; for instance **Eigenvectors[A][[1]]**.

Which eigenvector will give you the steady state?
The steady state is associated with the eigenvector corresponding to the eigenvalue of 1.

In[•]:=
```
(*Find eigenvalues and eigenvectors*)
eigenvalues = Eigenvalues[T];
eigenvectors = Eigenvectors[T];

(*Display eigenvalues and eigenvectors*)
eigenvalues
eigenvectors
```

Out[•]=
```
{1., 0.741421, 0.458579}
```

Out[•]=
```
{{0.890871, 0.445435, 0.0890871},
 {0.73658, -0.673392, -0.0631886}, {-0.275692, -0.527733, 0.803425}}
```

Re-scale this eigenvector (i.e., choose another eigenvector in the eigenspace/span) so that it represents a probability vector (dividing by the sum of the entries will do). If you want to calculate the sum of a vector v you can use **Norm[v, 1]**.

In[•]:=
```
(*The index of eigenvalue equal to 1*)
index = 1;

(*Steady state eigenvector*)
steadyStateEigenvector = eigenvectors[[index]];

(*Rescale the eigenvector to represent a probability vector*)
normalizedSteadyStateEigenvector =
  steadyStateEigenvector / Norm[steadyStateEigenvector, 1];

(*Display the normalized steady state eigenvector*)
normalizedSteadyStateEigenvector
```

Out[•]=
```
{0.625, 0.3125, 0.0625}
```

Okay, now let's talk about Google.

Google went online in the late 1990s, and when it did, the thing that set it apart was that the search result listings always seemed to give what you wanted toward the top (with other search engines, you'd have to flip through pages and pages). Part of what made Google so powerful was the PageRank algorithm. It rates the importance of each webpage, and from there present the user with the most relevant pages first.

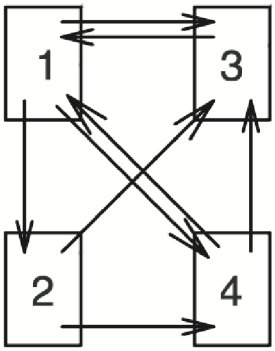
The PageRank algorithm operates from the following belief about how pages should be ranked:

The Set Up: *Suppose that there is a "web surfer" who is crawling across the Internet and, when they are*

at a webpage, they look at all of the links on the page and randomly click one of them. (So if there are links to three other webpages, they would click each one with probability 1/3.) The score of the webpage should be the probability that, in the long run, the surfer is on that page. That is, the user is taking a random walk on the Internet graph.

That is, the PageRank vector is the steady state of a Markov chain/process.

Suppose that we have an internet with only four pages (1, 2, 3, 4) with the following links (see Figure).



Question 5. Find the Markov transition matrix for the "Internet" represented by the above link diagram. For instance, if you are at page 2 at time t, at time t + 1, you are at page 3 with probability 1/2 and page 4 with probability 1/2.

	p1	p2	p3	p4
p1	0	0	1	1/2
p2	1/3	0	0	0
p3	1/3	1/2	0	1/2
p4	1/3	1/2	0	0

In[•]:=
```
MTM = {{0, 0, 1, 1 / 2}, {1 / 3, 0, 0, 0}, {1 / 3, 1 / 2, 0, 1 / 2}, {1 / 3, 1 / 2, 0, 0}}
```

Out[•]=

$$\left\{\left\{0, 0, 1, \frac{1}{2}\right\}, \left\{\frac{1}{3}, 0, 0, 0\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, \frac{1}{2}\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, 0\right\}\right\}$$

Question 6. Suppose that the "random web surfer" starts at a random webpage; so the initial state vector would be: u = (.25, .25, .25, .25). Find the probability vector for the surfer after *t = 3* clicks/transitions.

In[•]:=
```
(*Define the initial state vector*)u = {0.25, 0.25, 0.25, 0.25};

(*Calculate the probability vector after t=3 transitions*)
u3 = MatrixPower[MTM, 3].u;

(*Display the result*)
u3
```

Out[•]=
```
{0.354167, 0.145833, 0.291667, 0.208333}
```

Question 7. Find the steady state by finding the eigenvector with eigenvalue 1; and rescale it so that it is a probability vector. If you want to find the sum of the entries in a vector, you can type **Norm[v, 1]**.

In[•]:=
```
(*Find eigenvalues and eigenvectors*)
eigenvalues = Eigenvalues[MTM];
eigenvectors = Eigenvectors[MTM];

(*Find the index of the eigenvalue 1*)
index = Position[eigenvalues, 1][[1, 1]];

(*Steady state eigenvector*)
steadyStateEigenvector = eigenvectors[[index]];

(*Rescale the eigenvector to form a probability vector*)
steadyStateProbabilityVector = steadyStateEigenvector / Norm[steadyStateEigenvector, 1];

(*Display the steady state probability vector*)
steadyStateProbabilityVector
```

Out[•]=

$$\left\{\frac{12}{31}, \frac{4}{31}, \frac{9}{31}, \frac{6}{31}\right\}$$

Question 8. The steady state provides your PageRanking (as it provides the long-term probability that the surfer is on that page). Which webpage has largest PageRank?
Just looking at our probability vector. Page One with the rank: 12/31. Intuitively, this is reasonable, as looking at the matrix (see question 5), the value of the links pointing to page 1 is more than any other page's.

In[•]:=
```
(*Find the index of the maximum probability in the steady state vector*)
maxProbabilityIndex =
  FirstPosition[steadyStateProbabilityVector, Max[steadyStateProbabilityVector]][[1]];

(*Display the webpage with the largest PageRank*)
webpageWithLargestPageRank = maxProbabilityIndex;

webpageWithLargestPageRank
```

Out[•]=

```
1
```

Some practicality and the insight for PageRank. No one is going to search the internet for an "infinite" amount of time. Since the Internet is very large, allowing unreasonably long surfing times could lead to undesired results.

For instance:
Suppose that there's a corner of the Internet that is only accessible by, say, one webpage. But then once you're in that corner, it is very difficult to leave (since there is only perhaps one bridge back).

If we let our surfer to click around indefinitely, it's not unreasonable to think that they'll end up in such a corner of the internet. (Which isn't what we want to consider being "important" anyway.)

So, to make the model more realistic, we letting the surfer "sign off/stop surfing and begin again at a random webpage."

So we suppose that, at each time, the random walker will continue on their typical random walk (taking a random link) with probability .85, and with probability 0.15 will hop to a random webpage. Lots of studies have shown that this is a reasonable dampening factor. Since the probability of signing off at a given moment is 0.15, the expected value for the number of jumps before signing off is 1/0.15 = 6.67. (This is a geometric distribution from probability, if you want to know more, feel free to ask.) The new transition matrix would then be this: (If you changed the name of your transition matrix to something other than T, fill it in.)

Note:
(* Define the teleportation probability *)
teleportationProbability = 0.15;

(* Calculate the dampening factor matrix *)
dampeningFactorMatrix = teleportationProbability * ConstantArray[1, {4, 4}]/4;

(* Calculate the new transition matrix PageRankTransition *)
PageRankTransition = (1 - teleportationProbability) * MTM + dampeningFactorMatrix;

(* Display the new transition matrix PageRankTransition *)
PageRankTransition

In[•]:=
```
PageRankTransition =
 0.85 * MTM + 0.15 * {{1 / 4, 1 / 4, 1 / 4, 1 / 4}, {1 / 4, 1 / 4, 1 / 4, 1 / 4},
    {1 / 4, 1 / 4, 1 / 4, 1 / 4}, {1 / 4, 1 / 4, 1 / 4, 1 / 4}}
```

Out[•]=
```
{{0.0375, 0.0375, 0.8875, 0.4625}, {0.320833, 0.0375, 0.0375, 0.0375},
 {0.320833, 0.4625, 0.0375, 0.4625}, {0.320833, 0.4625, 0.0375, 0.0375}}
```

Question 9. Find the associated eigenvalues and eigenvectors of PageRankTransition. You'll likely want to use the [[1]] to just focus on the first eigenvector. If you don't like the imaginary part (which will always turn out to be zero for the eigenvector with eigenvalue 1), you can use the function **Re[ ]** to just grab the real part.

In[•]:=
```
(*Find eigenvalues and eigenvectors*)
eigenvalues = Eigenvalues[PageRankTransition];
eigenvectors = Eigenvectors[PageRankTransition];

(*Select the first eigenvector and grab the real part*)
firstEigenvector = Re[eigenvectors[[1]]];

(*Display the eigenvalues and the real part of the first eigenvector*)
eigenvalues
firstEigenvector
```

Out[•]=
$$\{1. + 0.\,i,\ -0.30653 + 0.349329\,i,\ -0.30653 - 0.349329\,i,\ -0.23694 + 0.\,i\}$$

Out[•]=
$$\{0.696483,\ 0.268281,\ 0.544778,\ 0.3823\}$$

Question 10. You're the owner of webpage 3 and are a bit peeved that your webpage isn't ranked as highly as #1. So you decide to create a webpage, #5, with links to and from webpage #3. Write down the new transition matrix. Then compute the PageRanks of the webpages. What happened to the ranking of webpage 3?

	p1	p2	p3	p4	p5
p1	0	0	1/2	1/2	0
p2	1/3	0	0	0	0
p3	1/3	1/2	0	1/2	1
p4	1/3	1/2	0	0	0
p5	0	0	1/2	0	0

```
(*The tampered matrix*)
newMTM = {{0, 0, 1 / 2, 1 / 2, 0}, {1 / 3, 0, 0, 0, 0},
  {1 / 3, 1 / 2, 0, 1 / 2, 1}, {1 / 3, 1 / 2, 0, 0, 0}, {0, 0, 1 / 2, 0, 0}}
```

Out[•]=
$$\left\{\left\{0, 0, \frac{1}{2}, \frac{1}{2}, 0\right\}, \left\{\frac{1}{3}, 0, 0, 0, 0\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, \frac{1}{2}, 1\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, 0, 0\right\}, \left\{0, 0, \frac{1}{2}, 0, 0\right\}\right\}$$

In[•]:=
$$\left\{\left\{0, 0, \frac{1}{2}, \frac{1}{2}, 0\right\}, \left\{\frac{1}{3}, 0, 0, 0, 0\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, \frac{1}{2}, 1\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, 0, 0\right\}, \left\{0, 0, \frac{1}{2}, 0, 0\right\}\right\}$$

```
(*DAMPEN THE MATRIX*)

(*Define the teleportation probability*)
teleportationProbability = 0.15;

(*Calculate the dampening factor matrix*)
dampeningFactorMatrix = teleportationProbability * ConstantArray[1, {5, 5}] / 4;

(*Calculate the new transition matrix PageRankTransition*)
newPageRankTransition = (1 - teleportationProbability) * newMTM + dampeningFactorMatrix;

(*Display the new transition matrix PageRankTransition*)
newPageRankTransition
```

Out[•]=
$$\left\{\left\{0, 0, \frac{1}{2}, \frac{1}{2}, 0\right\}, \left\{\frac{1}{3}, 0, 0, 0, 0\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, \frac{1}{2}, 1\right\}, \left\{\frac{1}{3}, \frac{1}{2}, 0, 0, 0\right\}, \left\{0, 0, \frac{1}{2}, 0, 0\right\}\right\}$$

Out[•]=
```
{{0.0375, 0.0375, 0.4625, 0.4625, 0.0375},
 {0.320833, 0.0375, 0.0375, 0.0375, 0.0375}, {0.320833, 0.4625, 0.0375, 0.4625, 0.8875},
 {0.320833, 0.4625, 0.0375, 0.0375, 0.0375}, {0.0375, 0.0375, 0.4625, 0.0375, 0.0375}}
```

In[ ]:=

```
(*CALCULATE THE NEW PAGE RANKS*)

(*Find eigenvalues and eigenvectors*)
eigenvalues = Eigenvalues[newPageRankTransition];
eigenvectors = Eigenvectors[newPageRankTransition];

(*The index of the eigenvalue 1 is the first*)
index = 1;

(*Steady state eigenvector. Grab the real part and find the absolute value *)
steadyStateEigenvector = Abs[Re[eigenvectors[[index]]]];

(*Rescale the eigenvector to form a probability vector*)
steadyStateProbabilityVector = steadyStateEigenvector / Norm[steadyStateEigenvector, 1];

(*Display the steady state probability vector*)
steadyStateProbabilityVector
```

Out[ ]=

```
{0.235476, 0.100451, 0.345002, 0.1416, 0.177471}
```

The new ranking of page 3 is now larger than the ranking of page 1.
So the SEO manipulation worked!

But... how can we compute an eigenvector of a matrix which is the size of the Internet? We don't. Actually, we can just approximate it by taking a few matrix powers of some sub-internet (iterating the system a relatively short amount of time). It turns out that this is not too bad of an approximation because the gap between the eigenvalue of 1 and that of the next one is quite large. (Remember when we diagonalized and took powers of the diagonal in the Fibonacci lab and those terms very quickly went to zero? This is exactly what's happening here.) There is always a fairly decent "eigenvalue gap" in practical PageRank matrices. So, we can get a decent approximation of the steady state "fairly quickly" (given the size of the data we're working with).

Want to learn more? *The Wikipedia page for Page Rank is quite good!* In general, math Wikipedia is a great place to learn a bit about various topics. E.g., Markov chains, networks, network centrality, ...