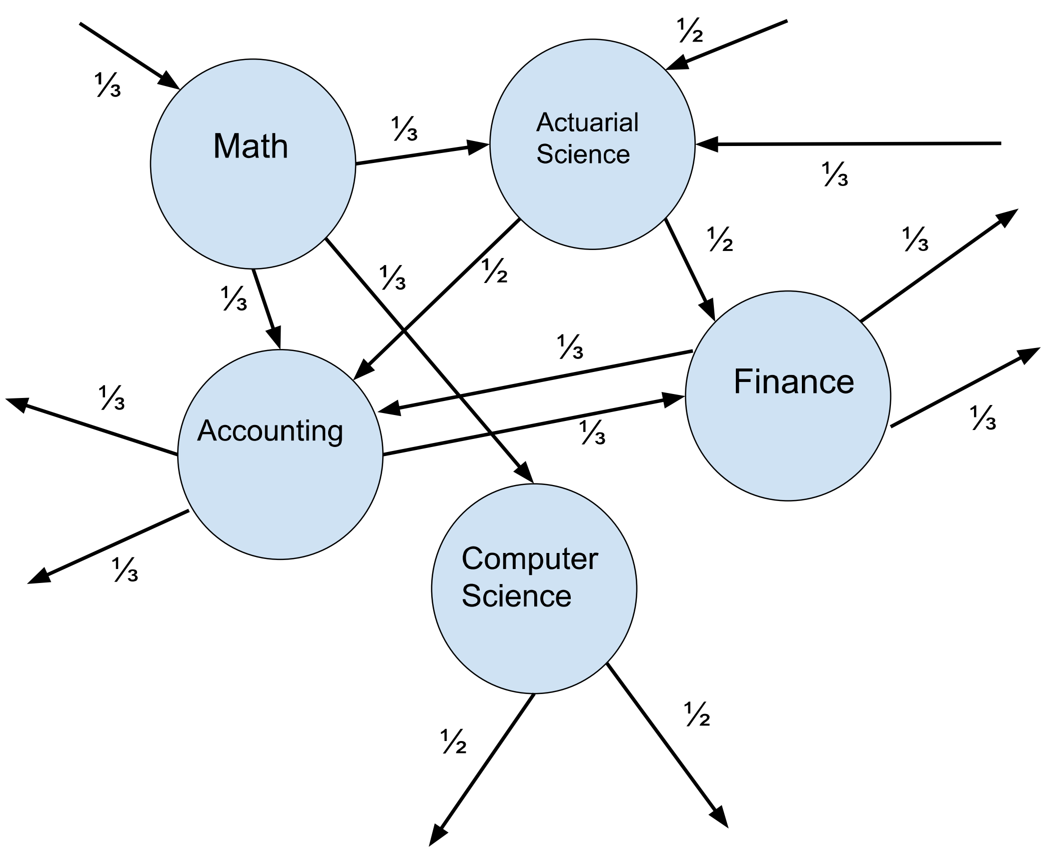
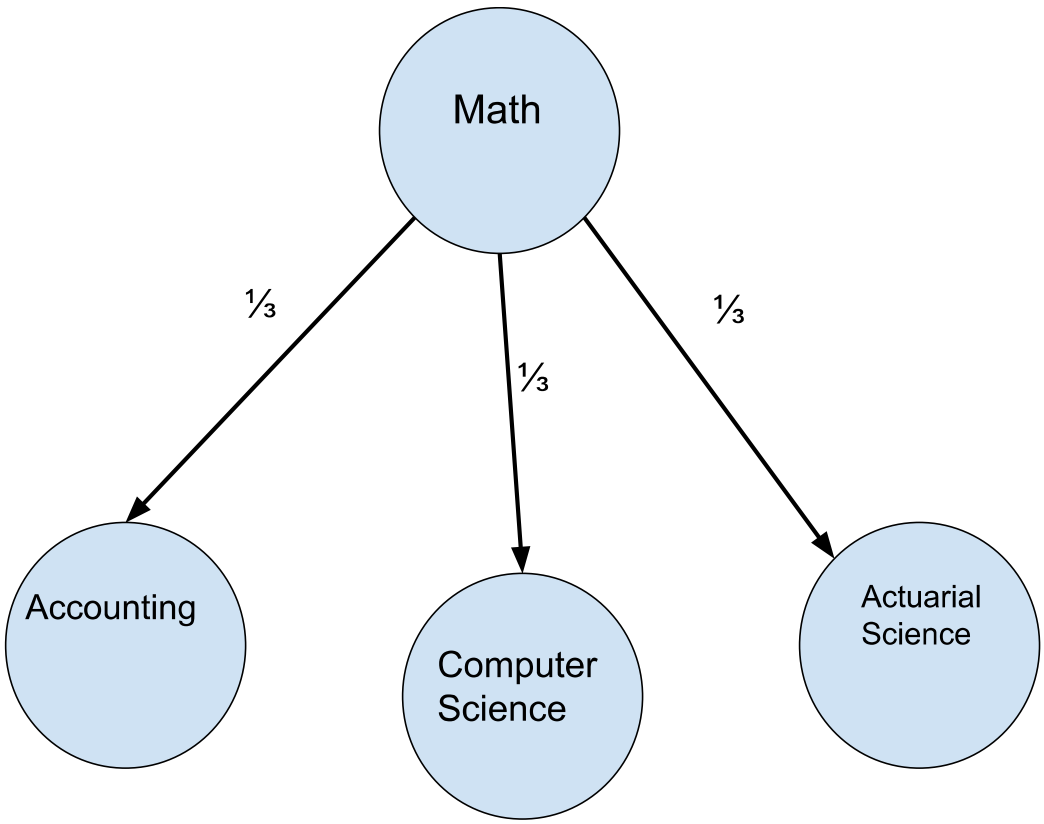
Google’s Pagerank Algorithm

The matrix.pdf file is located in the project’s directory. The matrix shows a list of all 89 majors not including minors. Each column in the matrix adds up to 1 which makes this a stochastic matrix.

When clicking through different web pages, we notice that some majors point to 3 majors at most. Some majors point to 1, some to 2, and some to 3. Below shows a small diagram of what this would look like.



From the diagram above, each major points to different majors and passes values depending on what other majors it's also pointing to. For example, we see that Math points to three other majors which are Actuarial Science, Accounting, and Computer Science. We pass 1/3 of a point to each webpage to make it even.



To be able to find how fair or unfair this ranking is, we will need to find the steady-state vector that the Markov Chain converges to by applying the formula below where A is the 89 x 89 matrix and x_k is our initial vector x_0 to start off with.

$$x_{k+1} = Ax_k$$

Our initial vector will have a total of 89 elements and each element will have a value of 1/89 to make it even.

We can find the steady-state vector another way by finding the eigenvalues of the matrix using the formula below and solving for λ where A is the matrix and I is the identity matrix.

$$A - \lambda I$$

We can find the eigenvalues of the 89 x 89 matrix by computing it below.

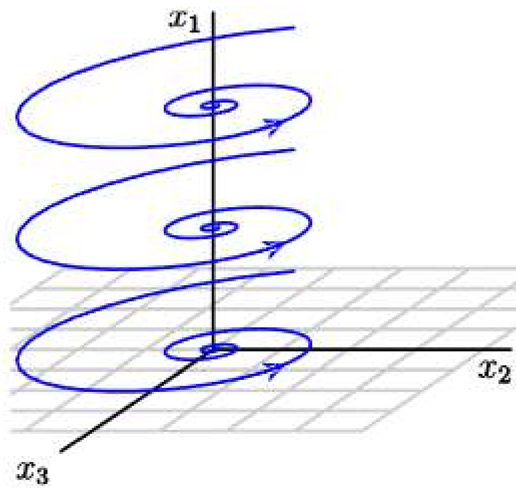
```
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1/3,    0,
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,

x0 = vector([1/89,  1/89,   1/89,   1/89,   1/89,   1/8
#markov_chain(A, x0, 70)
show(A.eigenvalues())
```

Evaluate (Sage)

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

We see that one of the eigenvalues is equal to 1 which is what we need in order to find the steady-state vector. According to Perron-Frobenius theorem, if the matrix is a positive stochastic matrix, then the eigenvalues satisfy $\lambda_1 = 1$ and $|\lambda_j| < 1$ where j > 1 which is true for the eigenvalues that we received.

Below shows a graph from the textbook of what it would look like when every eigenvector with $\lambda < 1$ converges to the eigenvector with $\lambda = 1$. Whenever the eigenvector with $\lambda = 1$ is being multiplied by the matrix, the vector remains unchanged.



The only difference is that the graph is for a 3 x 3 matrix from the textbook. This visualization is similar to an 89 x 89 matrix where every eigenvector will spiral to the eigenspace E_1.

To find the eigenvector of the eigenvalue 1, we will apply the eigenvalue in the formula below to find the null space where λ is the eigenvalue 1.

$$Nul(A - \lambda I)$$

When computing it to get the eigenvectors of every eigenvalue, we get the results below.

```
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1/5,   0,
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,

x0 = vector([1/89,  1/89,   1/89,   1/89,   1/89,   1/8
#markov_chain(A, x0, 70)
#show(A.eigenvalues())
show(A.eigenvectors_right())
```

Evaluate (Sage)

$$\left[\left(1, \left[\left(1, \frac{2105}{33}, \frac{10525}{88}, \frac{79953}{2530}, \frac{6315}{88}, \frac{1736727}{2024}, \frac{18222297}{2024}, \frac{23732}{2530}\right.\right.\right.\right.$$

We are focusing on the first eigenvector E_1 with an eigenvalue of 1. To find the probability vector in E_1, we need to find the appropriate scalar multiple of v.

To find the first value of the probability vector, we have to divide the first value by the total number of every element in the vector.

An appropriate way to find the steady-state vector with a large matrix. We will use the markov chain algorithm to see what probability vector it converges to.

$$x_{k+1} = Ax_k$$

Below shows the result of the vector it is converging to when computing it.

```
def markov_chain(A, x0, N):
    for i in range(N):
        x0 = A*x0
        print (x0.n())
## define the matrix A and x0
A = matrix([[0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
[1/3,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
[1/3,   1/2,    0,  0,  1/3,    0,  0,  0,  0,  0,  0,
```

Evaluate (Sage)

```
(0.00374531835205993, 0.0131086142322097, 0.0224719101123595, 0.011
(0.00249687890137328, 0.00936329588014981, 0.0190387016229713, 0.00
(0.00166458593424886, 0.00749063670411985, 0.0168539325842697, 0.00
(0.000832292967124428, 0.00430018033014288, 0.0117214592870024, 0.0
(0.000462384981735793, 0.00309797937762981, 0.00836916816941786, 0.
(0.000339082319939582, 0.00296119048719964, 0.00652926126292751, 0.
(0.000272293378133300, 0.00261825495907893, 0.00554508767333015, 0.
(0.000238042638745464, 0.00251726879466511, 0.00504540149572979, 0.
(0.000195942771581248, 0.00235428344028127, 0.00466541451671921, 0.
(0.000174892837999140, 0.00218766935087808, 0.00434495603632177, 0.
(0.000157216443678496, 0.00199775291562590, 0.00400234755519174, 0.
(0.000144725891438391, 0.00185376672024106, 0.00370002762945833, 0.
(0.000132793961886411, 0.00171897586531383, 0.00342352332861554, 0.
(0.000122162053298797, 0.00161303366461825, 0.00319259689836627, 0.
(0.000111403525648526, 0.00151496105909165, 0.00298836534991036, 0.
(0.000101595762528507, 0.00143189936593895, 0.00281295182829518, 0.
(0.0000920282400940738, 0.00135453434972282, 0.00265501813697538, 0
(0.0000834039821658944, 0.00128756273446635, 0.00251603000679482, 0
(0.0000753325619143812, 0.00122637493414985, 0.00239091404155632, 0
(0.0000681676172734367, 0.00117355327949481, 0.00228089342529133, 0
```

A Markov Chain function is created and prints the new vector after each iteration. After 20 iterations we noticed a problem. We can see that the vector is decreasing and slowly going to 0, but it’s not converging to a single vector no matter how many iterations we run. The markov chain function runs a for-loop and updates the vector x0 after multiplying with the matrix A.

We have noticed earlier that some webpages are not linked from other webpages which may be the issue. This disrupts the issue of convergence

In order to fix this problem, we have to modify the 89 x 89 matrix before we use Markov Chain again by using the formula below where G is the google matrix and a is alpha.

$$G^{'} = aG + (1 - a)H_{n}$$

We will set alpha to be 0.85.

$$G^{'} = 0.85G + 0.15H_{n}$$

This means that a user will follow a link randomly from one page to another 85% of the time and will randomly go to another page 15% of the time.

Since $G^{'}$ is a positive stochastic matrix, the Perron-Frobenius theorem says that the Markov Chain will converge to a steady-state vector.

Below shows the result for the Markov Chain using the modified matrix.

```
def modified_markov_chain(A, x0, N):
    r = A.nrows()
    A = 0.85*A + 0.15*matrix(r,r,[1.0/r]*(r*r))
    for i in range(N):
        x0 = A*x0
        print (x0.numerical_approx(digits=3))
    #print(x0.numerical_approx(digits=3))
## Define original Google matrix G and initial vector x0.
```

Evaluate (Sage)

```
(0.00487, 0.0128, 0.0208, 0.0112, 0.00964, 0.0335, 0.0272, 0.0128, 0.0256
(0.00397, 0.0101, 0.0183, 0.00718, 0.0130, 0.0308, 0.0387, 0.00832, 0.025
(0.00346, 0.00897, 0.0170, 0.00564, 0.0112, 0.0291, 0.0448, 0.00723, 0.02
(0.00302, 0.00731, 0.0143, 0.00519, 0.0103, 0.0280, 0.0498, 0.00790, 0.02
(0.00286, 0.00677, 0.0128, 0.00526, 0.00884, 0.0259, 0.0517, 0.00788, 0.0
(0.00281, 0.00672, 0.0121, 0.00521, 0.00819, 0.0249, 0.0535, 0.00797, 0.0
(0.00279, 0.00661, 0.0118, 0.00522, 0.00797, 0.0242, 0.0544, 0.00793, 0.0
(0.00278, 0.00658, 0.0117, 0.00520, 0.00784, 0.0237, 0.0551, 0.00782, 0.0
(0.00277, 0.00655, 0.0116, 0.00517, 0.00778, 0.0234, 0.0557, 0.00775, 0.0
(0.00277, 0.00651, 0.0115, 0.00514, 0.00774, 0.0233, 0.0561, 0.00766, 0.0
(0.00276, 0.00648, 0.0114, 0.00512, 0.00771, 0.0232, 0.0565, 0.00760, 0.0
(0.00276, 0.00646, 0.0114, 0.00510, 0.00768, 0.0231, 0.0568, 0.00754, 0.0
(0.00276, 0.00644, 0.0114, 0.00508, 0.00766, 0.0230, 0.0570, 0.00750, 0.0
(0.00276, 0.00643, 0.0113, 0.00507, 0.00765, 0.0230, 0.0572, 0.00747, 0.0
(0.00276, 0.00643, 0.0113, 0.00506, 0.00763, 0.0230, 0.0574, 0.00745, 0.0
(0.00276, 0.00642, 0.0113, 0.00505, 0.00763, 0.0230, 0.0575, 0.00743, 0.0
(0.00276, 0.00641, 0.0113, 0.00505, 0.00762, 0.0229, 0.0576, 0.00742, 0.0
(0.00276, 0.00641, 0.0113, 0.00505, 0.00761, 0.0229, 0.0577, 0.00741, 0.0
(0.00276, 0.00641, 0.0113, 0.00504, 0.00761, 0.0229, 0.0577, 0.00741, 0.0
(0.00276, 0.00641, 0.0113, 0.00504, 0.00761, 0.0229, 0.0578, 0.00740, 0.0
```

We can see that after 20 iterations, the Markov Chain is slowly converging to a single probability vector.

In the modified markov chain function, it uses the modified google matrix formula and uses the modified matrix in the for-loop.

After running 10 more iterations, the lowest value in the vector is 0.00169.

There is more than one major that has the same value. Below is a list of majors with the lowest benefit. These majors have the lowest value of 0.00169.

Lowest Benefit

- Eco Art
- Photography
- Criminal Justice - Social Sciences
- Psychology - Industrial
- Marketing - Professional Sales
- Professional Communication
- Mechanical Engineering
- Health Science Pre-Physical Therapy
- Health Science Pre-Dental
- Health Science Pre-Med
- Health Science Pre-Occupational Therapy
- Health Science Pre-Optometry
- Health Science Pre-Pharmacy
- Medical Imaging
- Interdisciplinary Studies
- Bible

The major with the highest benefit is shown below with the highest value of 0.0694

Highest Benefit

- Digital Marketing

From the google sheets, we can see that there are 6 majors that point to Digital marketing which makes it the highest ranking.

The majors that have the lowest value, don't have any majors that points to them which makes them have the lowest ranking.