Hello, markov chains. Notes include basic definitions, computational classes, stationary states, notes on google page rank, random walk. it covers L_n normalization, diagonalization, spectral theorem. other names this subject may be called are time discrete evolutions.

1 Notes

A **Markov chain** is a mathematical model that describes a sequence of possible events (states) where the outcome of each event depends only on the state of the previous event. This property is called the *Markov property*, meaning that the future state depends only on the present state, not on the sequence of events that preceded it.

2 Definitions

I will define markov chains more in tune with the definitions found in classical automata theory. (Why? bc why not)

- **State Space**: The set of all possible states in the Markov chain, denoted by S. For the sake of simplicy, let us assume that (for now) $|S| = n < \infty$. Or there is only a finite amount of states.
- **Transition Matrix***: A matrix δ where the element δ_{ij} is the probability of transitioning from state i to state j. Because this is in the space of probabilites, the sum of the columns must be 1.
- **Initial Distribution**: A vector $\underline{\mathbf{x_0}}$ representing the probabilities of starting in each state.

Thus let us represent markov chains with a 3-tuple (S, δ, x_0). We can define a computation up until t to be the lists of the vectors $x_0, x_1, x_2, ..., x_t$.

Note: $*$ The easiest way to verify that the transition matrix is valid is to add up the values of the columns. For example, given this matrix

$$\delta = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n} \\ p_{2,0} & p_{2,1} & \cdots & p_{2,n} \\ p_{3,0} & p_{3,1} & \cdots & p_{3,n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n,0} & p_{n,1} & \cdots & p_{n,n} \end{bmatrix}$$

$$\forall a \mid \sum_{i=0}^{n} p_{i,a} = p_{0,a} + p_{1,a} + ... + p_{n,a} = 1$$

A more algorithmic way to check this is through multiplying the n-dimensional one vector transposed $\underline{1_n}^T \delta = \underline{1_n}^T$. The math is the same, but the latter way is for more computational checking easily implemented in numpy.

We can notice that the following is true:

$$P(X_{t+1} = s \in S \mid \underline{x_t}, \underline{x_{t-1}}, \ldots, \underline{x_0}) = P(X_{t+1} = s \in S \mid \underline{x_t})$$

Or in other words, the **P**, or probability, of state s conditional on all past computation is actually equivalent to just the last computation. This property is called **memoryless**.

2.1 Basic Example

Let us have the following problem. *On a given day in winter, the probability that it will rain depends on the last day. If it rains today we know that there is a 75% it will rain tomorrow. If it doesn't rain today, then there will be a 25%, it rains on the tomorrow. Given that today it rains, what is the chance that it rains a week from today?*

2.1.1 Naive method

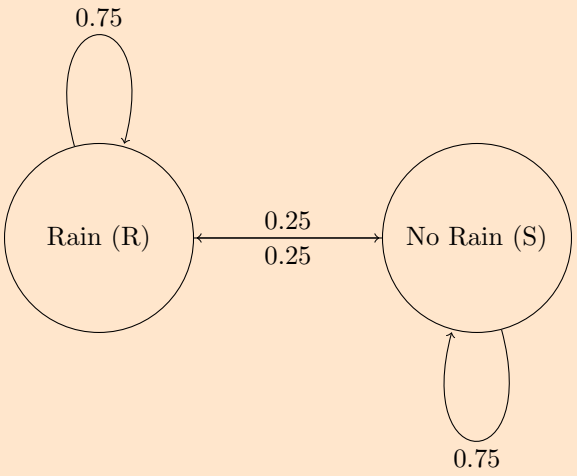
To solve this problem without markov chains, lets try to solve for the first few days:

1. On the day it is $P(\text{rains on day } 0) = 1$. This is the inital state.
2. On the first day after(or tomorrow), we know the chances are equivalent to a $P(\text{rain}) = 0.75$ and $P(\text{no rain}) = 0.25$. Easy
3. On the second day, we know the chances are equivalent to $P(\text{rains} \mid \text{rains on day } 1) = 0.75 * 0.75$ and $P(\text{rains} \mid \text{no rain on day } 1) = 0.25 * 0.25$. Adding the two together we get the following probaility of rain being 0.625
4. Repeating the process for the third day, we have to calculate a total of four combinations: [RRR, RRS, RSR, RSS] out of the total eight combinations of [RRR, RRS, RSR, RSS, SRR, SRS, SSR, SSS] where R means it rains and S means it did not rain. We can sum up the following to get a probability of 0.

$$\begin{aligned}
P(R_3) &= P(R_3|R_2,R_1)P(R_2 \cap R_1) + P(R_3|R_2,S_1)P(R_2 \cap S_1) \\
&\quad + P(R_3|S_2,R_1)P(S_2 \cap R_1) + P(R_3|S_2,S_1)P(S_2 \cap S_1) \\
&\text{by memoryless principal or only the the day before effects the current} \\
&= P(R_3|R_2)P(R_2 \cap R_1) + P(R_3|R_2)P(R_2 \cap S_1) \\
&\quad + P(R_3|S_2)P(S_2 \cap R_1) + P(R_3|S_2)P(S_2 \cap S_1) \\
&= 0.75P(R_2 \cap R_1) + 0.75P(R_2 \cap S_1) + 0.25P(S_2 \cap R_1) + 0.25P(S_2 \cap S_1) \\
&= 0.75^3 + 0.75 * 0.25 * 0.25 + 0.25 * 0.25 * 0.75 + 0.25 * 0.75 * 0.25 \\
&= 0.5625
\end{aligned}$$

2.1.2 With markov chains

Following is the state diagram



The three tuple (S, δ, x_0) are as follows:

1. $S = \{R, S\}$
2. $\delta = \begin{bmatrix} P(R_t \mid R_{t-1}) = 0.75 & P(S_t \mid R_{t-1}) = 0.25 \\ P(R_t \mid S_{t-1}) = 0.25 & P(S_t \mid S_{t-1}) = 0.75 \end{bmatrix}$
3. $\underline{x_0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ where $x_0[0]$ means rain and $x_0[1]$ means no rain

Then to calculate the days as follows:

1. **Day 1:** To calculate, we can multiply the probaility matrix.

$$\delta * \underline{x_0} = \begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.625 \\ 0.375 \end{bmatrix}$$

   The interpretation of the resulting vector x_1 is there is a 0.75 chance it will rain and a 0.25 chance it will not rain.

2. **Day 2:** We can apply the same idea,

$$\underline{x_2} = \delta * \underline{x_1} = \begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}$$

   The interpretation of the resulting vector x_2 is there is a 0.625 chance it will rain and a 0.375 chance it will not rain.

3. **Any day:** However, on the topic of computation, we can apply subsitution because we know that $\underline{x_1} = \delta \underline{x_0}$ to get

$$\underline{x_2} = \delta * \underline{x_1} = \delta(\delta * \underline{x_0}) = \delta^2 \underline{x_0}$$

   So we can find the t-th day, via jus the formula

$$\underline{x_t} = \delta^t \underline{x_0}$$

   The proof is trivial because it is just applying the subsitution over an over again. But checking if this indeed works for the third day,

$$\begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix}^3 \underline{x_0} = \begin{bmatrix} 0.5625 & 0.4375 \\ 0.4375 & 0.5625 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5625 \\ 0.4375 \end{bmatrix}$$

We can analyze how much easier this is for calculating probalities with more complexity. As the number of possible states increases, say instead of rain or no rain there were levels of precipitation, the naive calculations becomes harder and harder to compute. But for markov chains, the complexity scales with an increase in dimensions. To calculate t-th iteration, in this method it will only take $O(n^3t + n^2) = O(n^3t)$ where n is the size of the matrix or the the number of states($n = |S|$) and t is the desired time from the two steps

1. The first step is multiplying the matrix t times. While speed ups exists through algorithms like strassens, it is trivially useful here. Compute $n-$square matrix multiplication is equivalent to computing the individual positions in the matrix dot producted with the respective row and column. There are n^2 positions to compute and the dot product is done in linear time.

2. The extra n^2 calculation is trivial to the class, but it should be noted. This is the last computation where you multiply the resulting matrix with the start state vector. This multiplication is equivalent to multiplying $(n \times n)(n \times 1)$ or quadratic time.

For asymptotics, this is linear assuming a large enough t as n never changes.

3 Maybe speed up from $O(n^3t)$ to $O(log(t)n)$

This section is more relevant in the future sections about random walks. Not all probability matrices are real symmetric, but all simple graphs are real symmetric.
On the topic of speeding up computation, we can acutally calculate this even faster with some clever understanding of square matrices. More specifically, it requires the understanding of **diagonalization** and fast ways to exponentiate with integer values.

3.1 Real Symmetric Matrices

We should recall the following fact, that all real square symmetric matrices are diagonalizable. In more simple terms:

Note: A matrix is diagonalizable M implies there exists two matrices P, D such that it satisfies the following

$$M = PDP^{-1}$$

Where

1. P is the change of basis matrix. It is also the matrix of eigenvectors.
2. D are the eigenvectors in diagonal form or it must look as the following

$$\begin{bmatrix} \lambda_1 & 0 & 0 & ... & 0 \\ 0 & \lambda_2 & 0 & ... & 0 \\ 0 & 0 & \lambda_3 & ... & 0 \\ 0 & 0 & 0 & ... & \lambda_n \end{bmatrix}$$

3. P^{-1} exists because eigenvectors span the existing space so it must invertible.

Multiplication is a lot easier now. If we want to find the square of a matrix it is

$$M^2 = (PDP^{-1})^2 = (PDP^{-1})(PDP^{-1}) = PD^2P^{-1}$$

Because the inner $P^{-1}P$ cancels. We can see raising to any power of t is just this method repeated over and over again to get

$$M^t = (PDP^{-1})^t = (PDP^{-1})(PDP^{-1})(PDP^{-1})^{t-2} = PD^2P^{-1}(PDP^{-1})^{t-2} \\ = ... = PD^tP^{-1}$$

In a section, we will see how fast this could be done.

A basic fact about all $n-$dimensional square matrices is that there must exists n eigenvalues or $\lambda_1, \lambda_2, ...\lambda_n$ must all exist. This simple fact can be derived by how determinants are calculated. Recall to calculate determinants, is equivalent to finding when the determinant of $\det(M - \lambda) = 0$ for all λ's. We will get a polynomial(aka characteristic polynomial) and by some fact in algebra, it states that all polynomials of degree n must n complex-roots with multiplicity. **Ok let us assume you have checked that the matrix can be decomposed into the following form then speed ups are immeasurable(for example on graphs). Turning a linear problem into a logarithmic problem.**

Note: Real symmetric matrices have real eigenvalues
This is easy to prove with a basic understanding of inner products(for example dot product). We know the following must be true for any non-zero vector:

1. The inner product*(take for example a dot product) of a vector on itself must be positive**. This is just by definition of a valid inner product. We can express a matrix acting on an inner product in the following fashion of an inner product

$$\begin{aligned}\langle \underline{v} \mid M\underline{v} \rangle = \overline{\underline{v}}^T (M\underline{v}) = \overline{(\overline{M}^T \overline{\underline{v}})}^T \underline{v} = \langle \overline{M}^T \underline{v} \mid \underline{v} \rangle \\ = \langle \overline{M}\underline{v} \mid \underline{v} \rangle = \overline{\lambda}|v| \text{ by symmetric} \\ = \langle M\underline{v} \mid \underline{v} \rangle = \lambda|v| \text{ by real}\end{aligned}$$

   Thus, $\underline{v}$ is defined to be arbitary, so if we plug in all the eigenvectors of said space, then we will only get a scalar factor of $|v|^2$. The particular scalar is λ for the given eigenvalue. But we know that $\lambda|\underline{v}| = \overline{\lambda}|\underline{v}|$, so λ must be real.

This is also relates to the **spectral theorem for real symmetric matrices** which states that all real symmetric matrices are diagonalizable.
* Skipping a lot of the basic definitions for inner products
** This fact can be realized as a inner product defines a size of a vector. It does not make sense to say vector has negative size.

3.2 Speeding up exponentiation

Assume you are given a number and a exponent(integer), to calculate a^n with a the naive method of $a^n = a(a^{n-1}) = a * a * a^{n-2} = ...$ would require $O(n)$ linear time. However, we see that a better way by binary exponentiation exists.

(1) Binary Exponentiation given a^n:
BinEXP(a,n)

1. If $n = 1$ or $n = 0$: return a or 1
2. Return BinEXP(a,n//2) * BinEXP(a,n//2) + residual if n is odd

So for example the calculation of a^{125} are the following functional calls

1. $125 \rightarrow 62$
2. $62 \rightarrow 31$
3. $31 \rightarrow 15$
4. $15 \rightarrow 7$
5. $7 \rightarrow 3$
6. $3 \rightarrow 1$

A total of $O(log(n))$ time.

3.3 In conclusion

So let us assume that the markov probability matrix is diagonalizable, we only have to run a decomposition of the matrix into it's diagonal form(P, D) and then exponentiate all the eigenvalues(the λ's) through our method to have a total of only $O(log(t))$ time.

Note: We should definitely note that not all matrices are diagonalizable just because they are square. They exist. For example, the following have no diagonalization

$$\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}, \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

The reasons are different in every case. For example, the first case is the lack of linear independence and the second is the lack of dimensions.

3.4 Previous Example

In the previous example withe probability matrix

$$\delta = \begin{bmatrix}0.75 & 0.25\\ 0.25 & 0.75\end{bmatrix}$$

This matrix is diagonalizable(because it is real symmetric). The following is the decomposition

$$\delta = PDP^{-1} = \begin{bmatrix}-1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}0.5 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-0.5 & 0.5\\ 0.5 & 0.5\end{bmatrix}$$

So to calculate the t-th day we just have to do the following

$$x_t = \delta^t x_0 = (PDP^{-1})^t x_0 = PD^tP^{-1}x_0 = P\begin{bmatrix}0.5^t & 0\\ 0 & 1^t\end{bmatrix}P^{-1}x_0$$

This takes $log(t)$ time.

4 Stationary Distribution

In probability, there is the basic idea of long runs. For example, when gambling, you always lose because the house has an edge. Some people do not understand: see `https://www.instagram.com/mcmeatrocket/reels/`. The proper definition for stationary distribution is

Note: The **Stationary Distribution** π is $\underline{\pi} = \lim_{n\to\infty}\delta^n\underline{x_0}$ or when $\delta\underline{\pi} = \underline{\pi}$

The idea is that after a long time, one more time evolution from $t \to t+1$, there is no effect on what x_{t+1} is. This is a statement about convergence and not always a statement about exact values about finite times. There can exist situtations where $\delta^n x_0$ never equals said distribution but converges to it. Other names for the following property are **stable state** or **steady state** depending on the domain.

4.1 Solving for stationary distributions

We can solve for the steady state through the following intuition. We know that $\delta\underline{\pi} = \underline{\pi}$ then

$$\delta\underline{\pi} - \underline{\pi} = \underline{0} \leftrightarrow (\delta - I_{n\times n})\,\underline{\pi} = \underline{0}$$

Now there are a lot of ways to calculate this.

1. **Naive method** Just multiply the matrix over and over again until is numerically looks stable. Obviously multipying the matrix once will not work, but like 1000 might work or 10^{10}. At some point it would look stable.

2. **Eigenvector method** Ok so recall that if $\underline{v}$ is an eigenvector then, $M\underline{v} = \lambda\underline{v}$ for a given eigenvalue λ. λ is an eigenvalue if and only if $det(M - \lambda I_{n\times n}) = 0$. So from the above we know that $\underline{\pi}$ must one of the eigenvalues of said space generated by the δ matrix. Then we calculate all the eigenvalues and check each of them one by one to see if the corresponding eigenvalue is 1. If it is 1, then we have found the distribution.

Here is the example from above with the diagonalization decomposition.

$$\delta = PDP^{-1} = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

We can see that 1 is an eigenvalue associated with the corresponding eigenvector $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. Thus, we know as we continue to do this matrix multiplication, it will eventually reach the stable state of $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. Or after a long time, the chances of rain is only a fair coin flip. However, we should not that winter does not last forever.

Note: If you don't believe it, here are the following calculation of $\delta^{t=i}\underline{x_0}$ with wolfram alpha.

1. **t=10**

$$\begin{bmatrix} 0.500488 & 0.499512 \\ 0.499512 & 0.500488 \end{bmatrix} \underline{x_0} \tag{1}$$

2. **t=100**

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \underline{x_0} \tag{2}$$

The floating points get too small to even track at t=100.

4.1.1 On the topic of normalization

We will see in later examples that sometimes a stable state does not seem to match up with the corresponding eigenvalue. In this section, it will cover what L_m normalization means. We define the norm, magnitude, or inner product, of a vector $v \in R^n$ in respect of some L_m.

$$\text{dist}(v) = |v| = \langle v \mid v \rangle = \left(\sum_{i=0}^{n} |v_i^m| \right)^{\frac{1}{m}}$$

Note: We all know what **euclidean distance** is. Euclidean distance is just L_2 or given a vector we compute the square root of the square of each term. For example, given the following vector in 2D space the norm is as follows:

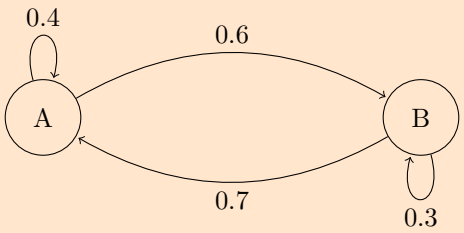
$$\left| \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right| = \left(2^2 + 3^2\right)^{\frac{1}{2}}$$

In terms of markov chains, we will be using the norm defined by L_1 or it is just equivalent to adding up all the components. For example:

$$\left| \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right| = \left(2^1 + 3^1\right)^{\frac{1}{1}} = 5$$

This is equivalent to **manhattan distance**. We choose to use L_1 or manhattan distance because of the way probabilities work. If we are given a set of probabilities, we expect the sum to be 100%. However, sometimes we are given a vector probability such that the vector does not add up to one. Take for example the following markov chain:

Following is the state diagram



The three tuple (S, δ, x_0) are as follows:

1. $S = \{A, B\}$
2. $\delta = \begin{bmatrix} P(R_t \mid R_{t-1}) = 0.4 & P(S_t \mid R_{t-1}) = 0.3 \\ P(R_t \mid S_{t-1}) = 0.6 & P(S_t \mid S_{t-1}) = 0.7 \end{bmatrix}$
3. $\underline{x_0}$ trivial here so not defined

If we start crunching down the numbers for the example above we get the following PDP decomposition.

$$\delta = \begin{bmatrix} 0.4 & 0.3 \\ 0.6 & 0.7 \end{bmatrix} = PDP^{-1} = \begin{bmatrix} -1 & 0.5 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix} P^{-1}$$

But if we were to run a numerical method on δ^n where n is really big, we get the following

$$\delta^{100} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{2}{3} \end{bmatrix}$$

As we can tell, the stable state does not seem to equal the corresponding eigenvector of the eigenvalue 1.

$$\begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \neq \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$$

But we can see that the first vector from the eigenvector is actually a scalar multiple of the actual stable state calculated through the numerical method. The multiple is $\frac{1}{1.5}$ which also happens to be the the L_1 norm of said vector. Hence the stable state is just the said eigenvector divided by its L_1 norm.

$$\text{stable state} = \frac{v_{\lambda=1}}{|v_{\lambda=1}|}$$

For our example, after normalization the method equal each other.

$$\text{stable state} = \frac{v_{\lambda=1}}{|v_{\lambda=1}|} = \frac{1}{1.5} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$$

One way to reason this is to recall how to calculate probabilities of one event given a space of probabilities. The vector encodes each event. Say for example, we have a box with the following objects: 4 red balls, 6 green ball, 10 blue balls. Then we can encode the space with the following vector:

$$\underline{v} = \begin{bmatrix} \text{number of red balls} \\ \text{number of green balls} \\ \text{number of blue balls} \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 10 \end{bmatrix} \rightarrow |\underline{v}| = 20$$

The chances of picking each color is just the vector divided by it's L_1 norm here.

4.1.2 On the topic of multiple stable states

There are possible occurances of multiple stable states. In other words, given a probability distribution, a markov chain may have multiple stable states. At this point, it really depends on the x_0 or the initial state configuration. There are also some knowledge of periodicity:

$$\delta = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} P^{-1}$$

The following are the behaviors

$$\delta(t) = \begin{cases} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & t = \text{ even} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2\times 2} & t = \text{ even} \end{cases}$$

This has the behavior of periodcity of two. We can only ensure it reacts the stable state if it is not periodic.

5 Google PageRank

Google Page Rank is a unique application of markov chains combined with some interpretations of eigenvector. The basic problem that PageRank is trying to solve is that given a set of pages that the user might want to see, how should they rank it? For example, if the user searches up Markov Chain, should a random blog be shown at the top or the wikipedia page.

5.1 Example

For the stake of this example, let us only have the following pages in our space, [Wikipedia page on Markov Chains(**W**), Blogpost(**B**), University PDF(**P**), Amazon Textbook(**A**), Useless pages(**X,Y,Z**)]. We can represent each of these as a state in out markov chain or $S = \{W, B, P, A, X, Y, Z\}$ and $|S| = 7$. The problem here is transition δ probability matrix needs to be defined. *We will use this later, but an easier example is given first for the following sections.*

5.2 Probability Matrix

Each of these pages is hyper linked to a few other links. Now let us define the behavior of a user. A user may do the following:

1. **Possibility 1:** User clicks a hyper link up until the user hits a dead end. For example, if the user is on wikipedia, he or she may choose to click a random link on the page. Let's say the user clicks a pdf with no links, then the user resets his session by going on a different website in the state space.

2. **Possibility 2:** User chooses click off the website and go on an entirely different website.

5.2.1 Probability Matrix of first possibility

Define the δ to be as the following

$$\delta_{i,j} = \begin{cases} \frac{1}{N_j} & \text{if there is a link between page j to i} \\ \frac{1}{n} & \text{page j does not have an outgoing edge or outdegree} = 0 \\ 0 & \text{otherwise} \end{cases}$$

We technically break it down into two case depending on the outdegree of the website, or vertex. The N_j is equivalent to the out degree of the website.

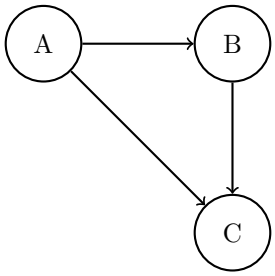
Note:

1. **Outdegree $>$ 0** We assume that there exists equal probability that the user could click on any of the websites.
2. **Outdegree = 0** We assume a uniform distribution so the column associated with the website is uniformly distributed. If there exists n websites, the column is as follows:

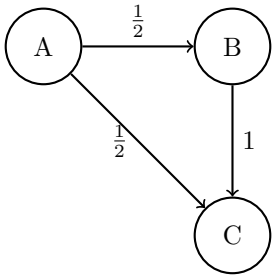
$$j\text{-column of } \delta = \begin{bmatrix} \frac{1}{n} \\ \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix}$$

The reasoning behind this is that because the user has no where to click, he resets. His current surf session cannot go anywhere. So he can now choose any website his state space. *This part seems more engineering than truly mathematical. See note below in section 5.2.3*

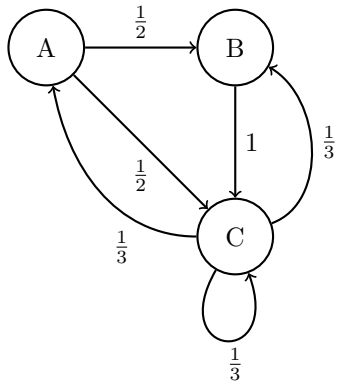
For example, given the following simpler graph on three vertices:



We can translate it into the following graph with weights:



However, we should notice that the following is not a proper markov chain because sum of the outgoing weights of all vertices must be 1. C however has a total outgoring probability of 0. To fix this we use case two. We get the following proper markov chain:



Which will result in the following probability matrix of

$$\delta_{\text{Possibility 1}} = \begin{bmatrix} 0 & 0 & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{3} \\ \frac{1}{2} & 1 & \frac{1}{3} \end{bmatrix}$$

5.2.2 Probabilty of second possibility

Because out assumption is that the user may choose to reset his session at any time and choose any random website, this is equivalent to all the columns being equally distributed. Hence, the matrix for any case on a web/state space of n to be equivalent to

$$\delta_{\text{Possibility 2}} = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \dots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \dots & \frac{1}{n} \\ \dots & \dots & \dots & \dots \\ \frac{1}{n} & \frac{1}{n} & \dots & \frac{1}{n} \end{bmatrix} = \frac{1}{n} J_{n \times n}$$

Note: The matrix J is the matrix with all 1 entries.

So in our case on the smaller example above with $S = \{A, B, C\}$ we have

$$\delta_{\text{Possibility 2}} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \frac{1}{3} J_{3 \times 3}$$

5.2.3 Adding both cases together

The issue here is we don't really know the behavior of the user(will the user choose to click off the website or choose to click a link). But, we can assume that both are possible. Thus, we can abstract this behavior by naming the chance the user will maintain the first possibility with probability $\alpha \leq 1$. Then the chance of the user click off is the complement or the chance of $1 - \alpha$. If we sum both possibilities up, we get the following matrix:

$$\begin{aligned} \delta &= \alpha \left(\delta_{\text{Possibility 1}}\right) + (1 - \alpha) \delta_{\text{Possibility 2}} \\ &= \alpha \left(\delta_{\text{Possibility 1}}\right) + \frac{1 - \alpha}{n} J_{n \times n} \end{aligned}$$

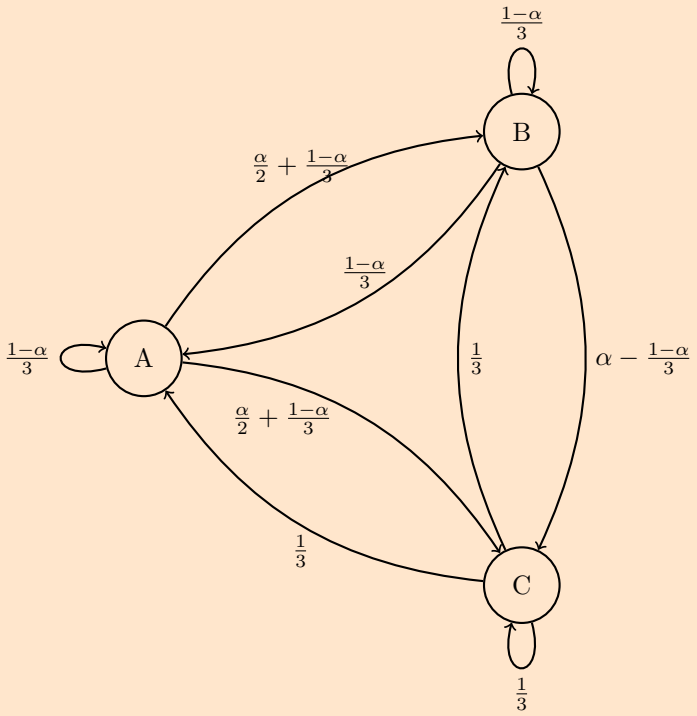
Note: If we compute the example from above we can an interesting fact

$$\delta = \alpha \begin{bmatrix} 0 & 0 & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{3} \\ \frac{1}{2} & 1 & \frac{1}{3} \end{bmatrix} + \frac{1-\alpha}{n} J_{n\times n} = \begin{bmatrix} \frac{1-\alpha}{3} & \frac{1-\alpha}{3} & \frac{\alpha}{3} + \frac{1-\alpha}{3} \\ \frac{\alpha}{2} + \frac{1-\alpha}{3} & \frac{1-\alpha}{3} & \frac{\alpha}{3} + \frac{1-\alpha}{3} \\ \frac{\alpha}{2} + \frac{1-\alpha}{3} & \alpha + \frac{1-\alpha}{3} & \frac{\alpha}{3} + \frac{1-\alpha}{3} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1-\alpha}{3} & \frac{1-\alpha}{3} & \frac{1}{3} \\ \frac{\alpha}{2} + \frac{1-\alpha}{3} & \frac{1-\alpha}{3} & \frac{1}{3} \\ \frac{\alpha}{2} + \frac{1-\alpha}{3} & \alpha + \frac{1-\alpha}{3} & \frac{1}{3} \end{bmatrix}$$

If the website has no outgoing edges, then the probability is uniform. This is probably why the probability in the first possibility is defined the way it is for zero degree websites.

The following is the equivalent markov chain with respect to a given α level.

Equivalent page rank given α



5.3 Ranking the pages

To rank the pages, we compute the stable states. Because the vector is n-dimensional, we just associate each value with the corresponding website. The website with the greatest value is ranked higher than the ones with smaller values. We have to give an α probability. This is better done with computer. But for the given example above lets select **the chance the user stays on the website 80% of the time or** $\alpha = 0.8$. The probability matrix for the given is with decomposition

$$\delta = \begin{bmatrix} \frac{0.2}{3} & \frac{0.2}{3} & \frac{1}{3} \\ \frac{0.8}{2} + \frac{0.2}{3} & \frac{0.2}{3} & \frac{1}{3} \\ \frac{0.8}{2} + \frac{0.2}{3} & 0.8 + \frac{0.2}{3} & \frac{1}{3} \end{bmatrix} = PDP^{-1} = \begin{bmatrix} 0.396825 & \dots & \dots \\ 0.555556 & \dots & \dots \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \dots \end{bmatrix} P^{-1}$$

Note: There are other values in the matrix decomposition. I chose not to include them because they are very long and overflowed the page. However, they do not play a part in the stable state. If you want to check it yourself, input the following into wolfram:

1. **Naive method** `{{0.2/3, 0.2/3, 1/3}, {0.8/2 + 0.2/3, 0.2/3, 1/3}, {0.8/2 + 0.2/3, 0.8 + 0.2/3, 1/3}}`

Then we have to apply L_1 normalization to the vector or

$$\text{stable state} = \frac{v_{\lambda=1}}{|v_{\lambda=1}|} \frac{1}{\left\| \begin{bmatrix} 0.396825 \\ 0.555556 \\ 1 \end{bmatrix} \right\|} \begin{bmatrix} 0.396825 \\ 0.555556 \\ 1 \end{bmatrix} = \frac{1}{1.952381} \begin{bmatrix} 0.396825 \\ 0.555556 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 0.203... \\ 0.284... \\ 0.512... \end{bmatrix}$$

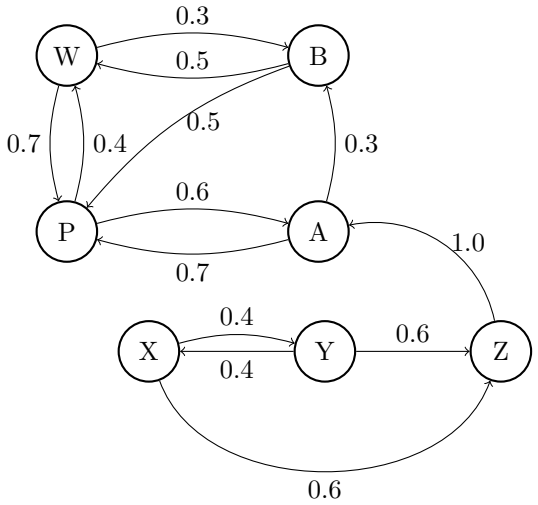
We can double check our computation is right by doing the following computation naively

$$\delta^{100} = \begin{bmatrix} 0.203252 & 0.203252 & 0.203252 \\ 0.284553 & 0.284553 & 0.284553 \\ 0.512195 & 0.512195 & 0.512195 \end{bmatrix}$$

So in our case, we will rank the pages as following $C(.512) > B(0.284) > A(0.20)$.

5.4 More complicated example

Given the following webspace, we will rank these pages from 5.1: Wikipedia page on Markov Chains(**W**), Blogpost(**B**), University PDF(**P**), Amazon Textbook(**A**), Useless pages(**X,Y,Z**).



Note: Very annoying to make in tikz. No idea what the websites are doing.

The corresponding probability matrix is

$$\delta = \begin{array}{c|ccccccc} & W & B & P & A & X & Y & Z \\ \hline W & 0 & 0.5 & 0.4 & 0 & 0 & 0 & 0 \\ B & 0.3 & 0 & 0 & 0.3 & 0 & 0 & 0 \\ P & 0.7 & 0 & 0 & 0.7 & 0 & 0 & 0 \\ A & 0 & 0.5 & 0.6 & 0 & 0 & 0 & 1 \\ X & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 \\ Y & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 \\ Z & 0 & 0 & 0 & 0 & 0.6 & 0.6 & 0 \end{array}$$

Goal is to write some code to calculate based on the α chances. To be done.

$$\delta = \begin{bmatrix} 0 & 0.5 & 0.4 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0.3 & 0 & 0 & 0 \\ 0.7 & 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0.5 & 0.6 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6 & 0.6 & 0 \end{bmatrix}$$

6 Random Walk algorithm

Adapted from lecture 6 of some cmu pdf. Link here

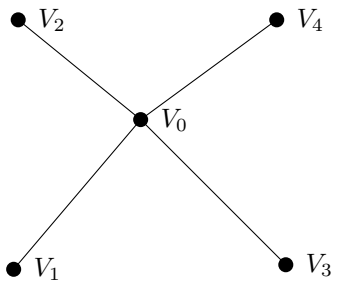
6.1 Random walk algorithm

Given a bidirectional graph, we can define a random vertex selector as follows:

(2) Randomly walk:

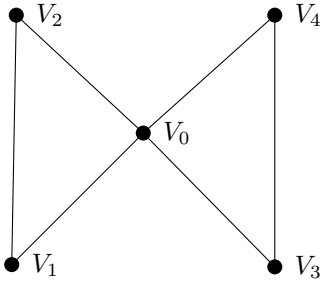
1. Pick a random edge (u, v)
2. Pick one side of the vertex v

There is $\pi(v)$ distribution based on a random vertex v on a given bidirectional graph. A vertex that participates in more edges than others will have a higher rate of being selected. For example given this graph G_1, the distribution $\pi(v)$ is



$$\pi(v \in V(G_1)) = \begin{cases} \frac{1}{2} & v = V_0 \\ \frac{1}{8} & v \neq V_0 \end{cases}$$

Modifying the graph, given $G_1^{'}$ the distribution $\pi(v)$ is



$$\pi(v \in V(G_1^{'})) = \begin{cases} \frac{1}{3} & v = V_0 \\ \frac{1}{6} & v \neq V_0 \end{cases}$$

In both cases we can see there is a relationship between the number of degrees of v_i and the total edges of the graph. We can calculate $\pi(v)$ on a bidirectional graph as

$$\pi(v) = \frac{\deg(v)}{2|E|}$$

We can reason this is true by the fact that our chances of picking a random edge that includes the said vertex v is $\frac{\deg(v)}{|E|}$. But assume we picked said edge (v, v'), we still have a chance to pick v' with probability $\frac{1}{2}$. Thus,

$$P(\text{picking an edge with v}) * P(\text{picking v} \in (u, v)) = \frac{\deg(v)}{|E|}\frac{1}{2} = \frac{\deg(v)}{2|E|}$$

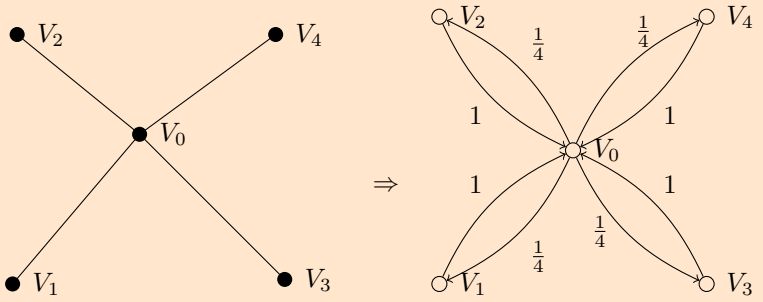
6.1.1 Reduction to Markov chains

We can reduce the problem of the random walks to the language of markov chains. We simply need to remap the vertex list to the state space and the adjacency matrix into a probability matrix.

1. **States** We can simply assign each vertice as a state in our markov chain
2. **Probability Matrix** For a markov chain, the probability matrix's columns need to sum up to 1. This is not the case, but we can just apply L_1 normalization here. *For our simple graphs, we will assume there exists no self loops as well as the degree of all vertices > 0. In other special cases, we may just have to add weights to self loops.*

So for example, the two graphs above are translated to

Translation of G_1



$$\text{Adj}(G_1) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \delta = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & 0 \end{bmatrix}$$

Running the decomposition we will have the following

$$\delta = PDP^{-1} = \begin{bmatrix} -4 & 0 & 0 & 0 & 4 \\ 1 & -1 & -1 & -1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} P^{-1}$$

We can see the stable state is the L_1 normalization of the 1 eigenvector

$$\text{stable state} = \frac{1}{|\underline{v}_{\lambda=1}| = 8} \begin{bmatrix} 4 & 1 & 1 & 1 & 1 \end{bmatrix}^T = \begin{bmatrix} \frac{1}{2} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{bmatrix}^T$$

On the other graph $G_1^{'}$, we have the following probability matrix:

$$\text{Adj}(G_1^{'}) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow \delta = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

So the PDP^{-1} decomposition of the probability matrix is as follows:

$$\delta = PDP^{-1} = \begin{bmatrix} -2 & -2 & 0 & 0 & 2 \\ 1 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus the resulting stable state is again

$$\text{stable state} = \frac{1}{|\underline{v}_{\lambda=1}| = 6} \begin{bmatrix} 2 & 1 & 1 & 1 & 1 \end{bmatrix}^T = \begin{bmatrix} \frac{1}{3} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}^T$$

In both graphs, G_1, G_1', their stable states matches up to the π distribution formula. In essence, the random walk is a more specific case of markov chains.

(1) To Prove: On a directed graph, the random walk is $\frac{\deg(v)}{|E|}$. One can imagine a bidirectional graph encoded as a unidirectional graph.

Some interesting facts about this:

Note: Regular graphs will have all the π distribution be all uniform. In other words $\pi(v_i) = \pi(v_j)$ for all $v_i, v_j \in K_n$. The distribution of any vertex with the algorithm described by 1 is equivalent to randomly picking an vertex.