Department of Economics and Finance

Chair of Gambling: Probability and Decision

The DeGroot Opinion Model in a Dynamic Random Graph: A Markov Chain and Computational Approach

Supervisor:

Prof.

Hlafo Alfie Mimun

Candidate:

Maxim Shlomo Voro

273501

Academic Year 2023/2024

A man is known by the company he keeps.

Contents

Introduction **i**

1 Markov Chains **1**

1.1 Introductory Definitions and Properties 1

1.2 Transience and Recurrence 4

1.3 Simple Random Walk on $\mathbb{Z}$ 9

1.4 Stationary Distribution 11

1.5 Period of a State and Aperiodic Markov Chains 12

1.6 Time Reversal and Reversible Markov Chains 15

1.7 Ergodic Theorem 19

2 Opinion Dynamics **21**

2.1 Introductory Definitions 21

2.2 Discrete Opinions Majority Model 21

2.2.1 Introductory Definitions 21

2.2.2 Stationary configuration and Consensus 25

2.2.3 Exit probability 26

2.2.4 Average opinion 26

2.3 Coevolution of Opinions and the graph 27

2.3.1 Network assortativity 27

2.4 Continuous opinions: De Groot Model 30
2.4.1 Intuition 30
2.4.2 Introduction 33
2.4.3 Convergence to a Consensus 35
2.4.4 Conditions for Convergence 36
2.4.5 Calculation of the Consensus 36

3 Networks & Python 39
3.1 Network Elements 39
3.2 Handling Network in Python 41
3.2.1 Undirected Graph 41
3.2.2 Graph Methods 42
3.2.3 Node and Edge Existence 43
3.2.4 Predecessor,Successors and neighbors 44
3.2.5 Networks Structures 44
3.3 Density and Sparsity 45
3.4 Subnetwork 46
3.5 Degree 47
3.6 Wieght 48
3.7 The Erdos-Renyi mode 48
3.8 Analysis of the DeGroot Model Behavior Netweok Approach 49

4 Analysis of the dynamic DeGroot Model Behavior in Erdos-Renyi Graphs 59
4.1 Introduction 59
4.2 Environment Creation 60
4.3 DeGroot Model in Costume Random Graph Markov chains approach 74
4.3.1 Experiment 1-Nodes per link 74

4.4 DeGroot Model in Costume Erdos-Renyi Graph Markov chains approach . . . 75
4.4.1 Experiment 2 -Number of Nodes . . . 75
4.4.2 Experiment 3 -Weight distribution . . . 75
4.4.3 Experiment 4 -Edge probability . . . 76
4.4.4 Experiment 5- Edge Rearrangement Probability . . . 76
4.5 Experiment Results . . . 77
4.6 Results Analysis . . . 77
4.6.1 Experiment 1 . . . 77
4.6.2 Experiment 2 . . . 78
4.6.3 Experiment 3 . . . 79
4.6.4 Experiment 4 . . . 79
4.6.5 Experiment 5 . . . 80

Conclusion **83**

Bibliography **84**

Introduction

We live in a world where game theory plays a decisive role in many aspects of life, from business to politics. In corporate America, shareholders choose their board of directors based on mathematical principles, while geopolitical conflicts are navigated through game theory strategies. Ultimately, game theory is a branch of applied mathematics.

The modern political climateis highly polarized; for example, people in the United States sometimes refuse to marry individuals who voted for a different political party. This polarization highlights the urgent need for common ground to stabilize society.

The goal of this thesis is to analyze the consensus-reaching process through mathematical models. To suggest possible solutions for the ongoing crisis.

To achieve this, I will employ a game-theoretical approach to decision-making, which uses mathematical methods to solve real-life dilemmas, from playing poker to determining investment strategies in real estate [5, 9]. As described in these sources, we approach every relevant outcome as an event with associated probabilities, such as reaching or not reaching a consensus.

I aim to develop a general model by utilizing the results of numerous experiments. Therefore, I will conduct over 100,000 simulations. Given that we are examining the behavior of a group of people essentially a network of people the best way to simulate and experiment with networks is through computational

power, specifically using Python and the NetworkX library [8].

To ensure normal and high-quality results, I chose the Erdos-Renyi graph among random graphs [4], as it generates a portion of all possible connections between agents, thus making the experiments more random.

To decide if a network of people can reach consensus, I selected a model from the field of opinion dynamics. This field seeks to provide explanations and algorithms for how ideas spread among people. I will utilize the DeGroot opinion model to define how opinions are spread. This model captures my interest because it considers that opinions are formed based on our own views and the influence of others, with each element affecting us to varying degrees [3].

The DeGroot consensus model is based on Markov chains [1]. Markov chains are a mathematical tool that examines stochastic processes, a series of events where the probability of the next event depends on the current situation. For example, moving around a house can be considered a stochastic process. When I am in my room, I have a 100 percent probability of going to the living room, but when I am in the living room, I can go to the kitchen, bathroom, or bedroom with some probability distribution. Each probability distribution should sum up to 100 percent.

DeGroot treats each agent as a node in a network, and each link in the network represents the influence of others' opinions with varying strengths. There can be self-loops, meaning an agent takes their own opinion into account while forming a new one. This resembles the phrase, "You are an average of the five people closest to you". Since the sum of the strength of the opinions equals 100 percent, we can treat this whole system as a stochastic process by applying Markov chains.

Even though Markov chains have been with us for more than a century, thanks to enormous computational power, they are more relevant than ever and should be treated as such.

1. Chapter One: A deep dive into Markov chains.

2. Chapter Two: An introduction to opinion models, with a focus on the DeGroot model.

3. Chapter Three: An exploration of network science and its applications in Python.

4. Chapter Four: The creation of an experimental lab, conducting experiments, and discussing the results.

Chapter 1

Markov Chains

1.1 Introductory Definitions and Properties

Notation 1.1.1. *In the entire chapter, we will write $P^t_{i,j}$ instead of $(P^t)_{i,j}$. So in general, $P^t_{i,j} \neq (P_{i,j})^t$.*

Definition 1.1.1. *V is a **state space**, which is a countable or finite set. $V = \{v_1, v_2, v_3, \ldots, v_n\}$.*

*n is the **number of elements** in V. If V is countable and infinite, then $n = \infty$.*

*The $n \times n$ matrix P is a **stochastic matrix** if the following two properties hold:*

***P1**: For $i, j = 1, \ldots, n$, $p_{ij} \in [0, 1]$.*

***P2**: For $i = 1, \ldots, n$, $\sum_{j=1}^{n} P_{i,j} = 1$.*

*$\pi_i(t) := P(X_t = v_i)$ and $\pi(t) := (\pi_1(t), \ldots, \pi_n(t))$ is the probability distribution of X_t. The vector $\pi(0)$ of length n is a **probability distribution** on V if the following two properties hold:*

***D1**: For any $j = 1, \ldots, n$, $\pi_j(0) \in [0, 1]$.*

D2: *$\sum_j \pi_j(0) = 1$. $P_{i,j}$ is the probability of moving from a state indexed by i to a state indexed by j.*

Definition 1.1.2. *The process $\{X_t\}_{t\in\mathbb{N}}$ is a **Markov chain** on the state space V with **transition matrix** P and **initial probability distribution** $\pi(0)$ if:*
M1: *The **Markov property** holds. That is, for any $t \geq 1$ and $i = 1, \ldots, n$, we have:*

$$P(X_t = v_i \mid X_1, \ldots, X_{t-1}) = P(X_t = v_i \mid X_{t-1})$$

The Markov property tells us that the future is independent of the past and depends only on the present.

M2: *For any $i = 1, \ldots, n$, $P(X_0 = v_i) = \pi_i(0)$.*

M3: *For any $i, j = 1, \ldots, n$ and $t \geq 1$,*

$$P(X_t = v_j \mid X_{t-1} = v_i) = P_{i,j}$$

*The Markov chain is **time-homogeneous** since the matrix P does not depend on the time t.*

Proposition 1.1.2.

$$\pi(t) = \pi(t-1) \cdot P \quad \text{for any } t \geq 1.$$

By iterating this formula, we get:

$$\pi(t) = \pi(t-1) \cdot P = \pi(t-2) \cdot P \cdot P = \pi(t-2) \cdot P^2 = \ldots = \pi(0) \cdot P^t.$$

So,

$$\pi(t) = \pi(0) \cdot P^t.$$

Example 1.1.1. *Consider the graph $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)\}$.*
G is the graph, V are the vertices, and E are the edges of the "square graph".

- *We start on vertex v_1.*
- *At each round, we have the probability p of moving clockwise and the probability q of moving counterclockwise.*

Let X_t be the position at time t. Since we start from vertex v_1, we have $P(X_0 = v_1) = 1$ and $P(X_0 = v_i) = 0$ for $\forall i = 2, 3, 4$. The probability distribution at round t is

$$\pi(t) = (\pi_1(t), \pi_2(t), \pi_3(t), \pi_4(t)),$$

where, $\pi_i(t) := P(X_t = v_i)$.

In our example, the initial probability distribution is

$$\pi(0) = (1, 0, 0, 0).$$

Note that, $\sum_{i=1}^{4} \pi_i(0) = \sum_{i=1}^{4} P(X_0 = v_i) = 1$.

The set V is called the state space, which contains 4 elements. Define P as the 4×4 matrix such that for $i, j = 1, 2, 3, 4$,

$$P_{i,j} = P(X_1 = v_j \mid X_0 = v_i).$$

Hence

$$P = \begin{pmatrix} 0 & p & 0 & q \\ q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \end{pmatrix}$$

P is the transition matrix, and it allows us to obtain the probability distribution at time t from the probability distribution at time $t - 1$.

In order to find the probability of being in state j at time 1, we multiply the probability of being in state i by the probability of moving from i to j for all i, and then sum all of these results over all i's, that is

$$\pi_j(1) = P(X_1 = v_j) = \sum_{i=1}^{4} P(X_1 = v_j \mid X_0 = v_i)P(X_0 = v_i)$$

$$= \sum_{i=1}^{4} P_{i,j}\pi_i(0) = \sum_{i=1}^{4} \pi_i(0)P_{i,j} = (\pi(0) \cdot P)_j.$$

The universal formula, that is for any $t > 0$, for $j = 1, 2, 3, 4$ is

$$\pi_j(t) = P(X_t = v_j) = \sum_{i=1}^{4} P(X_t = v_j \mid X_{t-1} = v_i)P(X_{t-1} = v_i)$$

$$= \sum_{i=1}^{4} P_{i,j}\pi_i(t-1) = (\pi(t-1) \cdot P)_j.$$

Hence,

$$\pi(t) = \pi(t-1) \cdot P.$$

1.2 Transience and Recurrence

Definition 1.2.1. *T_j is the the **Hitting time** of the state $v_j \in V$ as the first time $t \geq 1$ in which the chain X_t visits the state v_j (excluding time 0).*

$$T_j := \min\{t \geq 1 \mid X_t = v_j\},$$

with the convention that $\min \emptyset = \infty$.

Definition 1.2.2. *N_j is the the **number of visits of the state** $v_j \in V$ as the number of times that the chain visits v_j (excluding time 0).*

$$N_j := \sum_{t=1}^{\infty} \mathbf{1}_{\{X_t = v_j\}}.$$

Definition 1.2.3. *$f_{i,j}$ is the **probability** that, starting from state v_i, the Markov chain will eventually reach state v_j at least once.*

$$f_{i,j} = \mathbb{P}(T_j < \infty \mid X_0 = v_i) = \mathbb{P}(N_j \geq 1 \mid X_0 = v_i).$$

Definition 1.2.4. *We say that state v_j is*

- ***transient*** *if $f_{j,j} < 1$;*
- ***recurrent*** *if $f_{j,j} = 1$.*

Let $\{X_t\}_{t\in\mathbb{N}}$ be a Markov chain defined on the state space $V = \{v_1, \ldots, v_n\}$, where n could be finite or infinite.

If $N_j \geq m$ for some $m \geq 1$, it means the chain visits state v_j at least m times.

Let $t_0 = 0$ denote the starting time of our chain. There exist times $t_1, t_2, \ldots, t_m$ such that $1 \leq t_1 < t_2 < \ldots < t_m$. Each t_i signifies the moment when the chain reaches state v_j for the i-th time.

By definition at these times, we have

$$X_{t_1} = X_{t_2} = \ldots = X_{t_m} = v_j\,.$$

Moreover, between any consecutive occurrences t_{i-1} and t_i, the chain does not visit v_j, that is

$$X_t \neq v_j \quad \text{for all } t \in (t_{i-1}, t_i)\,.$$

This property ensures that the Markov chain $\{X_t\}$ precisely hits state v_j at time t_i, and during the interval (t_{i-1}, t_i), it visits in other states.

For $i = 1, 2, \ldots, m$, let us define A_i an event that describes the chain from the time it left state v_j until it reaches v_j again.

$$A_i = \{X_t \neq v_j \text{ for all } t \in (t_{i-1}, t_i),\ X_{t_i} = v_j\}\,.$$

Alternatively,

$$A_i = \{X_{t_{i-1}+1} \neq v_j,\ X_{t_{i-1}+2} \neq v_j,\ \ldots,\ X_{t_i-1} \neq v_j,\ X_{t_i} = v_j\}$$

Note that the event $\{N_j \geq m\}$ occurs if there exists a sequence of times $t_1, \ldots, t_m$ such that $A_1, \ldots, A_m$ occur, that is $\{N_j \geq m\} = \bigcap_{i=1}^{m} A_i$. Hence

$$P(N_j \geq m \mid X_0 = v_i) = P\left(\bigcap_{i=1}^{m} A_i \mid X_0 = v_i\right)$$

is the probability of hitting v_j at least m times given that $X_0 = v_i$. Note that

$$P\left(\bigcap_{i=1}^{m} A_i \mid X_0 = v_i\right) = \prod_{k=1}^{m} P\left(A_k \mid \bigcap_{i=1}^{k-1} A_i \cap \{X_0 = v_i\}\right).$$

This expression uses the multiplication rule, which states that the joint probability of a sequence of events can be computed by multiplying the conditional probabilities of each event given the previous ones. Moreover we have

$$\prod_{k=1}^{m} P\left(A_k \mid \bigcap_{i=1}^{k-1} A_i \cap \{X_0 = v_i\}\right) = \prod_{k=2}^{m} P\left(A_k \mid X_{t_{k-1}} = v_j\right) \cdot P(A_1 \mid X_0 = v_i).$$

The above formula is due to the Markov property. Indeed the Markov property ensures that the probability of transitioning to the next state depends solely on the current state. Since each event A_k concludes when the chain reaches v_j, we only need to consider the current state v_j. Moreover due the time-homogeneous property

$$\prod_{k=2}^{m} P(A_k \mid X_{t_{k-1}} = v_j) \cdot P(A_1 \mid X_0 = v_i) = P(A_1 \mid X_0 = v_j)^{m-1} \cdot P(A_1 \mid X_0 = v_i) = \\ = (f_{j,j})^{m-1} \cdot f_{i,j}.$$

Indeed the time-homogeneous property states that the probability of transitioning between states is independent of the specific time step. The f terms were derived using their definitions.
Summarising all the passages we have

$$P(N_j \geq m \mid X_0 = v_i) = (f_{j,j})^{m-1} \cdot f_{i,j}\,.$$

Let us now look at the expected number of visits to state v_j given that the initial state is v_i, that is $\mathbb{E}[N_j \mid X_0 = v_i]$. This expectation can be expressed as

$$\mathbb{E}[N_j \mid X_0 = v_i] = \sum_{m=1}^{\infty} \mathbb{P}(N_j \geq m \mid X_0 = v_i) = \sum_{m=1}^{\infty} (f_{jj})^{m-1} \cdot f_{ij}\,.$$

Let $k = m - 1$, then the above summation can be rewritten as $\sum_{k=0}^{\infty}(f_{jj})^k \cdot f_{ij}$. Using the geometric series identity

$$\sum_{k=0}^{\infty} a^k = \begin{cases} \frac{1}{1-a}, & \text{if } |a| < 1, \\ \infty, & \text{if } |a| \geq 1\,, \end{cases}$$

we derive

$$\mathbb{E}[N_j \mid X_0 = v_i] = \sum_{k=0}^{\infty}(f_{jj})^k \cdot f_{ij} = \begin{cases} \frac{f_{ij}}{1-f_{jj}}, & \text{if } v_j \text{ is transient,} \\ \infty, & \text{if } v_j \text{ is recurrent.} \end{cases}$$

This expression gives the expected number of visits to state v_j starting from state v_i, considering whether v_j is transient or recurrent under the Markov chain dynamics.

Definition 1.2.5. *$\{X_t\}_{t\in\mathbb{N}}$ is an irreducible Markov chain if $f_{i,j} > 0$ for all $v_i, v_j \in S$. This means that, for any two states in the state space S, there is a positive probability of transitioning from one state to the other. In terms of the associated graph G, this implies that there is a path connecting any pair of vertices. Therefore, saying that G is an irreducible graph is equivalent to saying that P is an irreducible matrix.*

Proposition 1.2.1. *The state v_j is*

- *transient if $\sum_{m=1}^{\infty} P_{j,j}^m < \infty$,*
- *recurrent if $\sum_{m=1}^{\infty} P_{j,j}^m = \infty$.*

Proof. We have

$$\mathbb{E}[N_j \mid X_0 = v_j] = \mathbb{E}\left[\sum_{m=1}^{\infty} 1\{X_m = v_j\} \mid X_0 = v_j\right] = \\ = \sum_{m=1}^{\infty} \mathbb{P}(X_m = v_j \mid X_0 = v_j) = \sum_{m=1}^{\infty} P_{j,j}^m.$$

Since

$$\mathbb{E}[N_j \mid X_0 = v_j] = \begin{cases} \infty, \text{ if } v_j \text{ is transient} \\ \infty, \text{ if } v_j \text{ is recurrent,} \end{cases}$$

we have the thesis. □

Proposition 1.2.2. *If the state v_i is recurrent and $f_{ij} > 0$, then the state v_j is also recurrent and $f_{ji} = 1$.*

Proposition 1.2.3. *If $\{X_t\}_{t\in\mathbb{N}}$ is an irreducible Markov chain, then all states are either transient or recurrent.*

Proposition 1.2.4. *An irreducible recurrent Markov chain visits every state infinitely often. This means that for each state v_j in the state space V, the chain will return to v_j an infinite number of times. Formally, this can be expressed as:*

$$\mathbb{P}\left(\bigcap_{m=1}^{\infty}\bigcup_{k\geq m}\{X_k = v_j\}\right) = 1 \quad \forall v_j \in V$$

This expression indicates that the probability of visiting state v_j infinitely often is 1 for every state v_j.

Proposition 1.2.5. *An irreducible transient Markov chain visits no state infinitely often. This means that for each state v_j in the state space V, the chain will only visit v_j a finite number of times. Formally, this can be expressed as:*

$$\mathbb{P}\left(\bigcap_{m=1}^{\infty}\bigcup_{k\geq m}\{X_k = v_j\}\right) = 0 \quad \forall v_j \in V$$

This expression indicates that the probability of visiting state v_j infinitely often is 0 for every state v_j.

1.3 Simple Random Walk on $\mathbb{Z}$

Suppose there is a trader in the stock market where each trade can result in either winning or losing \$1. The probabilities are given by

$$P(\text{Gaining a dollar}) = p, \quad P(\text{Losing a dollar}) = q\,.$$

Assume that

- the trader starts with an initial balance $X_0 = 0$.
- each trade, the trader either wins or loses.
- X_t is the trader's balance after the t-th trade.
- $\{X_t\}_{t\in\mathbb{N}}$ is a Markov chain on state space $\mathbb{Z}$ since it satisfies the Markov property. Knowing X_{t-1}, we can derive X_t:

$$X_t = \begin{cases} X_{t-1} + 1, & \text{with probability } p, \\ X_{t-1} - 1, & \text{with probability } q. \end{cases}$$

  for any $t \in \mathbb{N}$ and $j \in \mathbb{Z}$

$$\mathbb{P}(X_t = j \mid X_{t-1}, \ldots, X_1) = \mathbb{P}(X_t = j \mid X_{t-1})\,.$$

The state space is countable, the transition matrix P and the initial probability distribution have infinite dimensions. The transition matrix associated with $\{X_t\}_{t\in\mathbb{N}}$ is the matrix P such that for $i, j \in \mathbb{Z}$

$$P_{i,j} := \mathbb{P}(X_t = j \mid X_{t-1} = i) = \begin{cases} p, & \text{if } j = i + 1, \\ q, & \text{if } j = i - 1, \\ 0, & \text{if } j \neq \{i - 1, i + 1\}. \end{cases}$$

Hence, for example we have $P(X_1 = 1) = p$, $P(X_1 = -1) = q$ and

$$P(X_2 = j) = P_{1,j} \cdot P(X_1 = 1) + P_{-1,j} \cdot P(X_1 = -1) = P(X_2 = j) =$$
$$= \begin{cases} p \cdot p + 0 \cdot q = p^2, & \text{if } j = 2, \\ q \cdot p + p \cdot q = 2pq, & \text{if } j = 0, \\ 0 \cdot p + q \cdot q = q^2, & \text{if } j = -2, \\ 0 \cdot p + 0 \cdot q = 0, & \text{if } j = \pm 1. \end{cases}$$

The graph G associated with the Markov chain $\{X_t\}_{t \in \mathbb{N}}$ has vertices representing all possible states, which are elements of $\mathbb{Z}$, the set of all integers. The edges of G are defined by pairs $(i, i+1)$ for each integer $i \in \mathbb{Z}$. This structure reflects the transitions where each state i can transition to $i + 1$.

The irreducibility of G implies that there is always a path between any two states in $\mathbb{Z}$.

Note that $P_{0,0}^m$ is the probability that, starting at 0, the chain visits 0 at time m. This probability can be different from 0 if and only if m is an even number. So,

$$P_{0,0}^m = \begin{cases} > 0, & \text{if } m \text{ is even;} \\ = 0, & \text{if } m \text{ is odd.} \end{cases}$$

For this reason, let us write $m = 2k$ for $k \in \mathbb{N}$. In order to return to $X_{2k} = 0$, the chain must take k steps to the right and k steps to the left. Combinatorially, this involves finding all possible rearrangements of k wins (W) and k losses (L), where each arrangement corresponds to a feasible chain

$$P_{0,0}^{(2k)} = \binom{2k}{k} p^k q^k$$

Here, $\binom{2k}{k}$ represents the binomial coefficient, which counts the number of ways to choose k steps (either W or L) from $2k$ total steps, and $p^k q^k$ gives the probability

associated with each specific arrangement (where p and q are the probabilities of winning and losing, respectively). When k is large, we can use Stirling's formula to asymptotically approximate $k!$ getting

$$k! \sim \sqrt{2\pi k}\left(\frac{k}{e}\right)^k .$$

So, by Stirling's formula for large k, we have

$$\binom{2k}{k} = \frac{(2k)!}{(k!)^2} \sim \frac{\sqrt{4\pi k}\left(\frac{2k}{e}\right)^{2k}}{2\pi k\left(\frac{k}{e}\right)^{2k}} = \frac{2^{2k}}{\sqrt{\pi k}} \sim 2^{2k} = 4^k$$

Recalling the geometric series

$$\sum_{k=1}^{\infty} a_k = \begin{cases} \frac{1}{1-a}, & \text{if } a \in (0,1), \\ \infty, & \text{if } a \geq 1\,, \end{cases}$$

we have

$$\sum_{m=1}^{\infty} P_{0,0}^m = \sum_{k=1}^{\infty} P_{0,0}^{2k} = \sum_{k=1}^{\infty} \binom{2k}{k}(pq)^k \approx \sum_{k=1}^{\infty} 4^k (pq)^k =$$

$$= \sum_{k=1}^{\infty} (4pq)^k = \begin{cases} \frac{1}{1-4pq}, & \text{if } 4pq < 1, \\ \infty, & \text{if } 4pq \geq 1. \end{cases}$$

let us define $g(p) = pq = p(1-p) = p - p^2$ and note that $g(p) < \frac{1}{4}$ if $p \in [0,1] \setminus \left\{\frac{1}{2}\right\}$, and $g\left(\frac{1}{2}\right) = \frac{1}{4}$. Thus, we conclude that the Markov Chain $\{X_t\}_{t\in\mathbb{N}}$ is recurrent if $p = \frac{1}{2}$; otherwise, it is transient.

In probability theory, the process $\{X_t\}_{t\in\mathbb{N}}$ described here is known as a simple random walk on $\mathbb{Z}$. When $p = q$, it is specifically referred to as a symmetric simple random walk on $\mathbb{Z}$.

1.4 Stationary Distribution

V is the state space of a Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with transition matrix P.

Definition 1.4.1. *A probability distribution ρ on the state space V is called a stationary distribution (or invariant distribution) for the matrix P (or equivalently, for the Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with transition matrix P) if $\rho = \rho \cdot P$.*

Example 1.4.1. *Let us find the stationary distribution ρ for the Markov chain from Example 1.1.1*

$$\begin{pmatrix}\rho_1 & \rho_2 & \rho_3 & \rho_4\end{pmatrix} \cdot \begin{pmatrix} 0 & p & 0 & q \\ q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \end{pmatrix} = \begin{pmatrix}\rho_1 & \rho_2 & \rho_3 & \rho_4\end{pmatrix}$$

The solutions for the system are proportional to the vector $(1,1,1,1)$, so the final solution will be $\rho = (c,c,c,c)$ with $c \in \mathbb{R}$. Being a probability distribution ρ has to satisfy these two conditions:

- *$\rho_i \in [0,1]$ for all i;*
- *$\sum_i \rho_i = 1$.*

So we need $c = \frac{1}{4}$ and hence $\rho = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$.

1.5 Period of a State and Aperiodic Markov Chains

Given a set of numbers $A = \{x_1, \ldots, x_r\}$ we denote by $\gcd(A)$ the greatest common divisor of the set A. For example $\gcd(7, 42, 700) = 7$.

Definition 1.5.1. *Given a state $v_i \in S$, we define the **period of the state** v_i as*

$$d(i) := \gcd(D(i)),$$

where $D(i) := \{t \in \mathbb{N} \setminus \{0\} \mid P_{ii}^t > 0\}$. $D(i)$ is the set of indices corresponding to the rounds in which the chain returns to v_i, and $d(i)$ is the greatest common divisor of these indices.

Example 1.5.1. *Consider the problem in Section 1.3. For all v_i,*

$$D(i) = \{2, 4, 6, 8, \ldots\} = \{2k \mid k \in \mathbb{N}^+\} \Rightarrow d(i) = 2$$

.

Proposition 1.5.1. *If $v_i, v_j \in S$ are such that $f_{i,j} > 0$ and $f_{j,i} > 0$, then $d(i) = d(j)$. In particular, in an irreducible Markov chain all the states have the same period, denoted as $d(i)$, and hence we can speak about the period of the Markov chain.*

Definition 1.5.2. *If an irreducible Markov chain has period of state 1, we say that the Markov chain is aperiodic.*

Proposition 1.5.2. *If an irreducible aperiodic Markov chain in S with transition matrix P has a stationary distribution ρ, then it is recurrent and for any $v_i, v_j \in S$ we have*

$$\lim_{t\to\infty} P_{i,j}^t = \rho_j.$$

Furthermore, $\rho_j > 0$ for all $v_j \in S$.

Proposition 1.5.3. *If the Markov chain does not possess a stationary distribution, then for any $v_i, v_j \in S$, it holds that $\lim_{t\to\infty} P_{i,j}^t = 0$.*

Proposition 1.5.4. *An irreducible Markov chain in a finite state space S has a stationary distribution.*

Proof. Let P be the transition matrix of the Markov chain, and suppose that a stationary distribution does not exist. Note that, being P a transition matrix, we have $\sum_{v_j\in S} P_{i,j}^t = 1$. According to Proposition 1.5.3, we have $\lim_{t\to\infty} P_{i,j}^t = 0$. Hence $\sum_{v_j\in S} \lim_{t\to\infty} P_{i,j}^t = 0$. Being S finite, the summation over S is finite and hence we can exchange the symbols of limit and summation. So we have

$$0 = \sum_{v_j\in S} \lim_{t\to\infty} P_{i,j}^t = \lim_{t\to\infty} \sum_{v_j\in S} P_{i,j}^t = \lim_{t\to\infty} 1 = 1\,,$$

that contradicts the hypothesis. So we conclude that the stationary distribution exists. □

Proposition 1.5.5. *Let $\{X_t\}_{t\in\mathbb{N}}$ be an irreducible, aperiodic, recurrent Markov chain in S with transition matrix P. Then one of the following conclusions holds:*

(a) $\mathbb{E}[T_i \mid X_0 = v_i] < \infty$ *for all* $v_i \in S$*, and P has a unique stationary distribution ρ given by*

$$\rho_i = \frac{1}{\mathbb{E}[T_i \mid X_0 = v_i]}$$

*for $v_i \in S$. In this case, the chain is said to be **positive recurrent**, and Proposition 1.5.2 holds.*

(b) $\mathbb{E}[T_i \mid X_0 = v_i] = \infty$ *for all* $v_i \in S$*, and P has no stationary distribution. In this case, the chain is said to be **null recurrent**, and Proposition 1.5.3 holds.*

Example 1.5.2. *Consider the Markov chain on the state space $S = \{v_1, v_2\}$ with fixed $\alpha, \beta \in (0, 1)$.*

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}.$$

- *The associated graph is **irreducible** because we can get to any index from any index.*
- *Recurrence of Irreducible Markov Chains: If the transition matrix P of a Markov chain is irreducible and satisfies $P_{i,j} > 0$ for all i, j, then the chain is **recurrent**.*
- *For the state v_1 of a Markov chain, the set $D(1)$ comprises all $t \in \mathbb{N}\backslash\{0\}$ such that $P^t_{1,1} > 0$. Given that $1 \in D(1)$, it follows that $d(1) = \gcd(D(1)) = 1$. Since the Markov chain is irreducible, Proposition 1.5.1 implies that $d(2) = d(1) = 1$, hence confirming that the Markov chain is **aperiodic**.*

Let us look for the stationary distribution of such a Markov chain. We have

$$\begin{pmatrix} \rho_1 & \rho_2 \end{pmatrix} \cdot \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 \end{pmatrix}$$

So we have to solve the system of equations

$$\begin{cases} (1-\alpha)\rho_1 + \beta\rho_2 = \rho_1 \\ \alpha\rho_1 + (1-\beta)\rho_2 = \rho_2 \\ \rho_1 + \rho_2 = 1 \end{cases}$$

To solve this system, we have added the conditions $\rho_1 + \rho_2 = 1$ and $\rho_1, \rho_2 \in [0,1]$ (since we want ρ to be a probability distribution). Hence,

$$\begin{cases} (1-\alpha)\rho_1 + \beta\rho_2 = \rho_1 \\ \alpha\rho_1 + (1-\beta)\rho_2 = \rho_2 \\ \rho_1 + \rho_2 = 1 \end{cases} \Rightarrow \begin{cases} \rho_1 = \frac{\beta}{\alpha+\beta} \\ \rho_2 = \frac{\alpha}{\alpha+\beta} \end{cases}$$

Since the Markov chain is irreducible, aperiodic, and recurrent, then by Proposition 1.5.5, the Markov chain is positive recurrent. According to Proposition 1.5.5, we have

$$E[T_1 \mid X_0 = v_1] = \frac{1}{\rho_1} = \frac{\alpha+\beta}{\beta}, \quad E[T_2 \mid X_0 = v_2] = \frac{1}{\rho_2} = \frac{\alpha+\beta}{\alpha}.$$

1.6 Time Reversal and Reversible Markov Chains

Definition 1.6.1. *A stochastic matrix P and a probability distribution λ are said to be in **detailed balance** if*

$$\lambda_i P_{i,j} = \lambda_j P_{j,i} \text{ for all } i, j.$$

Proposition 1.6.1. *If the stochastic matrix P and the probability distribution λ are in detailed balance, then λ is a stationary distribution for P.*

Proof. To be proved $\lambda P = \lambda$. Since λ and P are in detailed balance, we have

$$(\lambda P)_i = \sum_j \lambda_j P_{j,i} = \sum_j \lambda_i P_{i,j} = \lambda_i \sum_j P_{i,j} = \lambda_i.$$

Therefore, because $(\lambda P)_i = \lambda_i$ there is a full probability distribution λ

$$\lambda P = \lambda.$$

□

To determine a probability distribution λ that is in detailed balance with a transition matrix P, the following steps can be followed:

1. Check for the existence of a stationary distribution for P by solving the system $\rho \cdot P = \rho$.

2. If the system has no solutions, then no probability distribution λ exists that is in detailed balance with P.

3. If the system has a solution ρ, verify whether ρ satisfies the detailed balance condition with P.

Specifically, if P is symmetric (i.e., $P_{i,j} = P_{j,i}$), the unique probability distribution ρ that is in detailed balance with P is the uniform distribution over the state space (i.e., $\rho_i = \rho_j$ for all i, j). This is demonstrated as follows. For any i, j,

$$\lambda_i P_{i,j} = \lambda_j P_{j,i} \Rightarrow \lambda_i P_{i,j} = \lambda_j P_{i,j} \Rightarrow \lambda_i = \lambda_j.$$

Thus, the uniform distribution is the unique distribution in detailed balance with a symmetric transition matrix P.

Proposition 1.6.2. *Let P be irreducible and have an stationary distribution ρ. Fix $T \geq 1$. Suppose that $\{X_n\}_{0 \leq n \leq T}$ is a Markov chain with initial probability*

distribution ρ and transition matrix P. Define $Y_0 = X_T$ and $Y_n = X_{T-n}$ for $n \geq 1$. Then the process $\{Y_n\}_{0 \leq n \leq T}$ is a Markov chain with initial distribution ρ and transition matrix $\hat{P}$, where the entries of $\hat{P}$ satisfy the equations

$$\rho_j \hat{P}_{j,i} = \rho_i P_{i,j} \quad \text{for all } i, j.$$

for all i, j. Moreover, $\hat{P}$ is also irreducible with stationary distribution ρ. The chain $\{Y_n\}_{0 \leq n \leq T}$ is called the ***time-reversal*** *of $\{X_n\}_{0 \leq n \leq T}$.*

Definition 1.6.2. *Let $\{X_n\}_{n \geq 0}$ be a Markov chain with initial distribution ρ and transition matrix P. If the Markov chain is irreducible, we say it is reversible if, for every $T \geq 1$, the sequence $\{X_{T-n}\}_{0 \leq n \leq T}$ also forms a Markov chain with initial distribution ρ and transition matrix P.*

Proposition 1.6.3. *Let P be an irreducible stochastic matrix and let ρ be a probability distribution. Suppose $\{X_n\}_{n \geq 0}$ is a Markov chain with initial distribution ρ and transition matrix P. The following statements are equivalent:*

- *$\{X_n\}_{n \geq 0}$ is reversible.*
- *P and ρ satisfy the detailed balance condition.*

Example 1.6.1. *Consider the Markov chain on the state space $S = \{v1, v2, v3\}$ with transition matrix*

$$P = \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

We want to determine if the Markov chain is reversible. Since the associated graph is irreducible, it is sufficient to demonstrate the existence of a probability distribution $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ that satisfies the detailed balance condition with P. To find such a probability distribution we need to first find the stationary distributions of P and then identify which of them are in detailed balance with P. So let us

determine if P has a stationary distribution, that is, let us find the solution $\rho = (\rho_1, \rho_2, \rho_3)$ of

$$\begin{pmatrix} \rho_1 & \rho_2 & \rho_3 \end{pmatrix} \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 & \rho_3 \end{pmatrix}$$

$$\begin{cases} \rho_1 \cdot 0 + \rho_2 \cdot \frac{1}{3} + \rho_3 \cdot \frac{2}{3} = \rho_1 \\ \rho_1 \cdot \frac{2}{3} + \rho_2 \cdot 0 + \rho_3 \cdot \frac{1}{3} = \rho_2 \\ \rho_1 \cdot \frac{1}{3} + \rho_2 \cdot \frac{2}{3} + \rho_3 \cdot 0 = \rho_3 \\ \rho_1 + \rho_2 + \rho_3 = 1 \end{cases}$$

This system has solution $\rho = (c, c, c)$ for all $c \in \mathbb{R}$. We have to choose $c \in \mathbb{R}$ such that $\sum_{i=1}^{3} \rho_i = 1$ and $\rho_i \in [0, 1]$ for $i = 1, 2, 3$. Hence,

$$1 = \sum_{i=1}^{3} \rho_i = \sum_{i=1}^{3} c = 3c \Rightarrow c = \frac{1}{3}.$$

So $\rho = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ is a stationary distribution for P and it is unique, since it is the unique solution of the system above that is also a probability distribution. To verify if $\rho = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ is in detailed balance with P, we check if $\lambda = \rho$ satisfies the system of equations:

$$\begin{cases} \lambda_1 P_{1,2} = P_{2,1} \lambda_2 \\ \lambda_1 P_{1,3} = P_{3,1} \lambda_3 \\ \lambda_2 P_{2,3} = P_{3,2} \lambda_3 \end{cases}$$

Let us examine each equations. For $\lambda_1 P_{1,2} = P_{2,1} \lambda_2$:

$$\frac{1}{3} \cdot \frac{2}{3} = \frac{1}{3} \cdot \frac{1}{3}$$

Simplifying gives $\frac{2}{9} = \frac{1}{9}$, which is false. Hence we conclude that ρ is not in detailed balance with P. Therefore, the Markov chain defined by the transition matrix P and the stationary distribution ρ is not reversible.

1.7 Ergodic Theorem

Definition 1.7.1. *Denote by $Y_i(n)$ the number of visits to the state v_i before time n, that is*

$$Y_i(n) = \sum_{k=0}^{n-1} 1_{\{X_k = v_i\}}.$$

Then $\frac{Y_i(n)}{n}$ is the proportion of time before time n spent in state v_i.

Theorem 1.7.1 (Ergodic Theorem)**.** *Let P be irreducible and let λ be any distribution. Let $\{X_n\}_{n\in\mathbb{N}}$ be a Markov chain on the state space S with initial distribution λ and transition matrix P. Then*

$$\mathbb{P}\left(\frac{Y_i(n)}{n} \underset{n\to\infty}{\longrightarrow} \frac{1}{\mathbb{E}[T_i \mid X_0 = v_i]}\right) = 1.$$

Moreover, if the Markov chain is positive recurrent, for any bounded function $f : S \to \mathbb{R}$ we have

$$\mathbb{P}\left(\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \underset{n\to\infty}{\longrightarrow} \sum_{v_i\in S} \rho_i f(v_i)\right) = 1,$$

where ρ is the unique stationary distribution of P.

In an irreducible and positive recurrent Markov chain, the average value of a function f over all states converges over time to a weighted sum. Each state's contribution to this average is weighted by how often the chain visits that state, as given by the stationary distribution ρ. This property highlights the long-term behavior and stability of the chain's dynamics.

Example 1.7.1. *Consider Example 1.5.2*

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}.$$

We have established that this Markov chain is irreducible and positive recurrent. Furthermore, its stationary distribution is given by

$$\rho = \left(\frac{\beta}{\alpha+\beta}, \frac{\alpha}{\alpha+\beta}\right).$$

Let f be defined such that $f(v_1) = 1$ and $f(v_2) = -1$. According to the Ergodic Theorem (Theorem 1.7.1), we have

$$\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \underset{n\to\infty}{\longrightarrow} \rho_2 \cdot f(v_2) + \rho_1 \cdot f(v_1) = \frac{\beta}{\alpha+\beta}\cdot(-1) + \frac{\alpha}{\alpha+\beta}\cdot 1 = \frac{\alpha-\beta}{\alpha+\beta}\,,$$

with probability 1.

Chapter 2

Opinion Dynamics

2.1 Introductory Definitions

Opinions can be of two types:

- **Discrete** (Binary or more): for instance, I like apple or not.
- **Continuous**: I have an appreciation rating for this restaurant that is a real value (for example a real valued between 0 and 5).

2.2 Discrete Opinions Majority Model

2.2.1 Introductory Definitions

Definition 2.2.1. *In the Majority Model, nodes (representing agents) in an undirected graph update their opinions based on the majority opinion of their neighbors.*

Definition 2.2.2. *There are three options for the agent's own opinion*

- ***Yes*** *- The agent considers their own opinion with the same importance as the opinions of their neighbors. In other words, their own opinion is treated*

just like any other neighbor's opinion.

- ***No*** *- The agent completely ignores their own opinion and only considers the opinions of their neighbors.*

- ***Yes, but with special weight*** *- The agent considers their own opinion, but it carries a different weight compared to their neighbors' opinions. This means their own opinion could be either more or less influential than a neighbor's opinion.*

Definition 2.2.3. ***The timing of how updates are made*** *in the model can significantly influence the final outcome. There are several options for updating the nodes:*

- ***Synchronous Updating****:*
  - *All nodes are updated simultaneously based on the opinions in the previous configuration.*

- ***Asynchronous Updating in Fixed Order****:*
  - *Nodes are updated one by one in a predetermined, fixed order. As each node is updated, the graph is immediately changed to reflect this new opinion before moving on to the next node in the sequence.*

- ***Asynchronous Updating in Random Order****:*
  - *Nodes are updated one by one, but in a random order that is chosen anew for each iteration.*

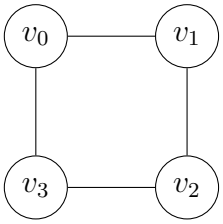
These different methods of updating can lead to different behaviors and results in the model, making the choice of timing crucial for the dynamics of opinion formation.

Definition 2.2.4. *When updating opinions in a social network model, ties can occur when the number of opinions among a node's neighbors is evenly split. In such cases, it's necessary to decide how the node should proceed. There are several approaches to break ties:*

- ***Keep**: The agent keeps their own opinion.*
- ***Random**: The agent randomly chooses a new opinion from the set of possible opinions. This approach ensures that the tie is broken unpredictably.*
- ***Switch**: The agent switches their opinion to the opposite of their current opinion.*

Each method has implications for how the model evolves over time and can influence the dynamics of consensus formation or polarization in the network.

Notation 2.2.1. *For the future graphs we will use the notation in following picture.*

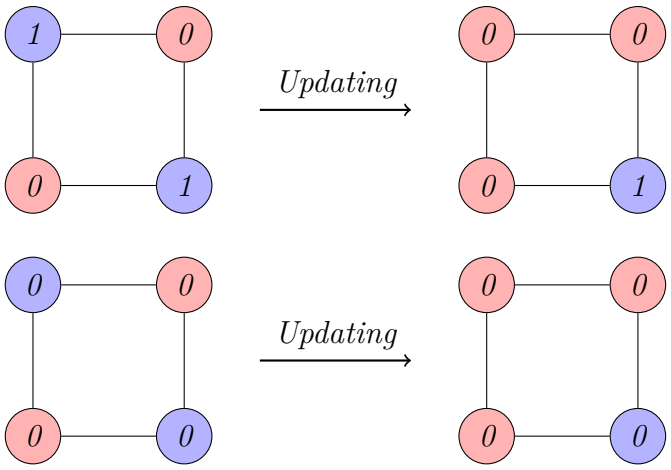


Example 2.2.1. *Consider the following network and the Majority dynamics with following rules:*

- *timing of the updating: Synchronously*
- *ties: keep*

Example 2.2.2. *Consider the following network and the Majority dynamics with following rules:*

- *Timing of the updating: asynchronous-fixed order 0, 1, 2, 3*
- *Ties: keep*



Example 2.2.3. *Consider the following network and the Majority dynamics with following rules:*

- *Timing of the updating: asynchroneous-random order 0, 1, 2, 3*
- *Ties: keep*
- *Starting with index 0 (with probability 0.25) or index 2(with probability 0.25)*

- *Starting with index 1 (with probability 0.25) or index 3 (with probability 0.25)*

2.2.2 Stationary configuration and Consensus

Definition 2.2.5. *There are two possible outcomes for any discrete dynamic.*

1. ***Stationary Configuration:***

   - ***Consensus:*** *All agents converge to the same opinion, resulting in a stationary state where no one changes their opinion anymore.*
   - ***Polarization:*** *At least two different opinions persist in the society despite the dynamic process.*

2. ***Non-Stationary Behavior:*** *Agents continue to change their opinions indefinitely without reaching a consensus.*

Proposition 2.2.2.

- *Consensus configurations are stationary configurations*
- *There may exist stationary configurations that are not consensus.*

Proposition 2.2.3.

- *In general, the dynamic does not converge to consensus.*
- *In one-dimensional and two-dimensional grids, the dynamic in the asynchronous models converges and there is emergence of the consensus.*

2.2.3 Exit probability

Definition 2.2.6. ***The exit probability*** *is a function defined as follows*

- ***Argument:*** *Initial fraction of agents with opinion 1.*
- ***Output:*** *The likelihood that the sequence of opinion configurations converges to consensus on opinion 1.*

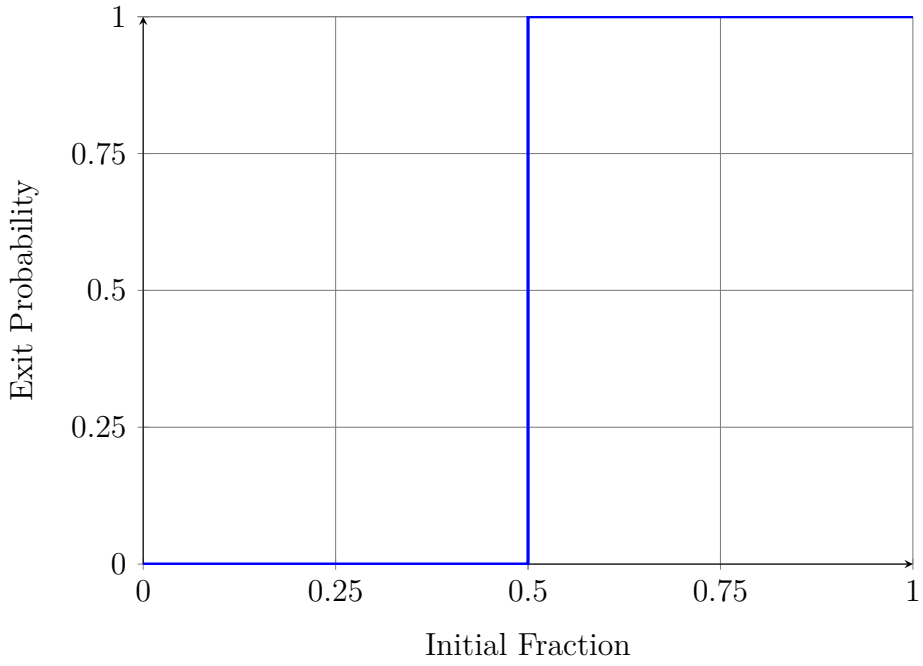


Fig. 2.1: Exit Probability as a function of the initial fraction of opinion 1

2.2.4 Average opinion

Definition 2.2.7. ***Average Opinion:*** *The average opinion is defined as the arithmetic mean of the opinions across all nodes in a network.*

2.3 Coevolution of Opinions and the graph

2.3.1 Network assortativity

Network assortativity refers to the tendency of nodes in a network to connect with other nodes that are similar to themselves in some way. Nodes in a network become similar to their neighbors through two main mechanisms: **social influence** and **selection**.

- **Social influence** refers to the process by which individuals adapt their behaviors, attitudes, or attributes to be more like those of their connected peers.
- **Selection** refers to the tendency of individuals to form connections with others who are already similar to them. This mechanism operates on the principle that people prefer to associate with those who share similar attributes, interests, or social statuses.

    1. **Initial similarity and connection formation (selection)**: Initially, individuals may seek out and form connections with those who are already similar to them. This is the selection process in action.
    2. **Increase in similarity through interaction (social influence)**: Once connections are established, ongoing interactions lead to social influence, where connected individuals become even more similar over time.

Example 2.3.1. *Consider a coevolution models where both opinions and networks evolve over time. Every iteration requires two rules:*

- *Rule for updating the opinion.*

- *Rule updating the graph (connections between nodes)*

Asynchronous Dynamic

- *Initialization: Select random opinions for agents*
- 1. *We go over all nodes in a random order.*
  2. *For each node i, pick a random neighbor j with a different opinion than i if it exists.*
  3. *With probability p, the link between i and j is rewired to a randomly chosen non-neighbor of i with the same opinion as i. If no such non-neighbor exists, keep the link intact (Selection).*
  4. *Otherwise, with probability $1-p$, i takes the opinion of j (Influence).*
  5. *If no such neighbor exists, go to the next node.*

General behavior of this dynamic

- *The number of links is constant.*
- *Impact of the selection step:*
  - *Decreases the number of pairs of nodes that are linked and have different opinions (mismatches).*
  - *Strictly decreases the number of mismatches if there exists a non-neighbor with the same opinion as node i.*
- *Impact of the influence step: Can increase or decrease the number of missmatches*

Proposition 2.3.1. *From any graph, the previous dynamic converges to a graph such that:*

1. *There may be several connected components. Regardless of the initial configuration of the graph (which nodes are connected to each other), the dynamic process described will eventually lead to a state where the graph is divided into several connected components.*

2. *All nodes in the same connected component have the same opinion. In simpler terms, all nodes within a connected component are directly or indirectly connected to each other.*

The parameter p plays a crucial role in the dynamics of opinion formation and network structure evolution. Here's an elaboration on its impact.

Low p:

- Opinions homogenize: when p is low, the likelihood of rewiring links between nodes (selection step) is minimal. Nodes tend to maintain their existing connections over time.

- Network stability: due to the low probability of rewiring, the overall network structure remains largely unchanged. Nodes retain their initial connectivity patterns.

High p:

- Opinions almost do not change: with a high p, nodes frequently rewire their connections based on their opinions (selection step). This leads to a network where nodes are strongly connected to others sharing the same opinion.

- Group formation: nodes tend to cluster into groups or communities where all members hold similar opinions. This clustering effect is intensified as nodes selectively rewire to reinforce existing opinion alignments.

2.4 Continuous opinions: De Groot Model

2.4.1 Intuition

Example 2.4.1. *Consider the following synchronous model. The opinion of the society is the average of the opinions of the agents. The* ***Restaurant Quality Evaluation*** *model can be described as follows:*

- *The quality of a restaurant is a real number in the interval* $[0, 5]$.
- *There are* N *agents.*
- *Each agent* i *goes to the restaurant and evaluates it.*
- *Each agent obtains an opinion* x_i*, which is randomized around the true quality of the restaurant.*
- *The opinions are aggregated by taking their average:*

$$\bar{x}_N = \frac{1}{N} \sum_{i=1}^{N} x_i$$

- *All agents communicate their opinions to their neighbors. Subsequently, each agent updates their own opinion based on the aggregated average opinion of their neighbors.*
- *There are two criticisms of this simplified model that we will address:*
  1. *Not all agents interact with the same neighbors.*
  2. *Some individuals are more influential than others.*

DeGroot's model provides a framework for understanding how individuals update their opinions through iterative interactions with others in a social network.

- **Naive opinion aggregation**: The model uses a simple rule where each agent updates their opinion based on the opinions of their neighbors.

- **Local information exchange**: Agents consider the opinions of their immediate neighbors, reflecting the idea of observing and aggregating information from their local social environment.

- **Adaptation to changes**: If an agent's neighbors suddenly change their opinions, the agent may adjust their own opinion to reflect new information or insights learned indirectly from their neighbors' interactions.

- **Weighted average update**: At each iteration, each agent updates their opinion by taking a weighted average of the opinions of their neighbors. This process continues iteratively until opinions converge or reach a stable state.

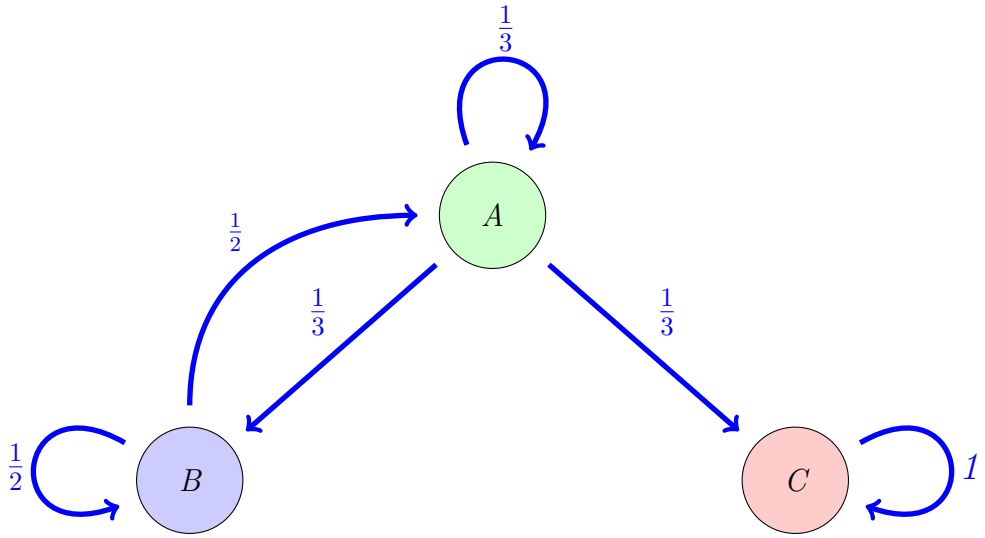
Example 2.4.2. *Consider a society with three agents updating their opinions as follows:*

- *Agent A calculates the average opinion, including their own opinion.*

- *Agent B considers the average opinion by including their own opinion and Agent A's opinion, but excluding Agent C's opinion.*

- *Agent C updates their opinion based solely on their own opinion.*

Let us denote by $\mathbf{P}(t)$ *the vector whose* j*-th entry is the opinion of agent* j *at*



time t.

Starting review

$$\mathbf{P}(1) = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

Review after second iteration

$$\mathbf{P}(2) = \begin{pmatrix} 1/3 \cdot 0 + 1/3 \cdot 0 + 1/3 \cdot 1 \\ 1/2 \cdot 0 + 1/2 \cdot 0 \\ 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 0 \\ 1 \end{pmatrix}$$

Review after third iteration

$$\mathbf{P}(3) = \begin{pmatrix} 1/3 \cdot 1/3 + 1/3 \cdot 0 + 1/3 \cdot 1 \\ 1/2 \cdot 1/3 + 1/2 \cdot 0 \\ 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 4/9 \\ 1/6 \\ 1 \end{pmatrix}$$

Review after infinite iterations

$$\lim_{t \to \infty} \mathbf{P}(t) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

2.4.2 Introduction

Definition 2.4.1 (Subjective probability distribution)**.** *For $i = 1, \ldots, K$, F_i refers to a person's personal belief or assessment of the likelihood of different outcomes or values of a variable. Unlike objective or frequentist probabilities.*

Consider a group of K individuals, who must act together as a team. Suppose that each of the K individuals can specify their own subjective probability distribution for the unknown value for some parameter value θ.

Proposition 2.4.1. *θ may be regarded as any arbitrary variable whose value is not completely known to the K individuals. The value of θ is assumed to lie on an abstract parameter space Ω that is endowed with a σ-field of measurable subsets for which probabilities can be specified. Thus, $F_i, \ldots, F_K$ are subjective probabilities over Ω which represent the prior beliefs about θ of the K individuals. In other words, for any measurable set A in the parameter space Ω, $F_i(A)$ is the prior probability of individual i that the value of θ lies in A.*

Definition 2.4.2. *The set $\{F_1, \ldots, F_K\}$ represents the beliefs about the assignment of θ by the agents. The set $\{F_1(A), \ldots, F_K(A)\}$ represents the beliefs of the assignment of the probability of event A by the agents.*

Definition 2.4.3. *$p_1, \ldots, p_k$ represent the weights that agent j assigns to the opinions of all other agents in the group.*

Proposition 2.4.2. *If $\sum_{i=1}^{k} p_i = 1$, then $\sum_{i=1}^{k} P_i \cdot F_i$ denotes a probability distribution over Ω, for which the probability of any measurable set A is $\sum_{i=1}^{k} P_i \cdot F_i(A)$. In simple terms, if we have a stochastic weight distribution, we can calculate the probability of event A by weight-averaging the values of the subjective probability distribution. This collective assessment is often referred to as the opinion pool.*

The precision of this model faces a significant obstacle: its accuracy relies heavily on the correct distribution of weights. the Degroot model should fix this problem

Weighing the Opinion of others

To address this challenge in the DeGroot model, each agent will have their own subjective distribution, where each agent assigns weights to the opinions of others based on factors such as expertise, closeness, and personal preferences.

Definition 2.4.4. *$P_{i,j}$ denotes the weight that agent i assigns to the opinion distribution of agent j.*

Definition 2.4.5. *If for every i and j, $P_{i,j} \geq 0$ and $\sum_{i=1}^{k} P_{i,j} = 1$, and the agent knows the subjective distributions $F_1, \ldots, F_k$, then the agent will revise their own distribution from $\{F_i\}$ to*

$$F_{i1} = \sum_{j=1}^{k} P_{i,j} \cdot F_j$$

Definition 2.4.6. *Let P denote a $K \times K$ stocastic matrix comprising the elements $P_{i,j}$ for $i, j = 1, \ldots, k$.*

Let us define

- Initial Distributions:

$$\mathbf{F}' = (F_1', F_2', \ldots, F_k')$$

- Updated Distributions after the first iteration:

$$\mathbf{F'}^1 = (F_1'^1, F_2'^1, \ldots, F_k'^1)$$

Hence

$$\mathbf{F'}^1 = P \cdot \mathbf{F}'$$

Iteration: after the first iteration, we have moved from the set $\{F_1, \ldots, F_K\}$ to the set $\{F_1^1, \ldots, F_K^1\}$. If we iterate the procedure again, we have

$$F_i^2 = \sum_{j=1}^{k} P_{i,j} \cdot F_j^1$$

and hence

$$\mathbf{F'}^2 = P \cdot \mathbf{F'}^1 .$$

The general formula will be:

$$\mathbf{F'}^n = P \cdot \mathbf{F'}^{n-1} . \tag{2.1}$$

It is assumed that the members of the group will continue to make these revisions until the revisions stop changing any member's subjective distribution. In other words,

$$\mathbf{F}^{(n)} = \mathbf{F}^{(n-1)} = \ldots = \mathbf{F}^*$$

2.4.3 Convergence to a Consensus

Definition 2.4.7. *The opinion configuration converges to consensus if*

$$\lim_{n\to\infty} F_{in} = F^* \quad \textit{for all} \quad i = 1, \ldots, k$$

Proposition 2.4.3. *Following* (2.1) *and Proposition 1.5.2, a consensus can be reached if and only if there exists a vector* $\pi = (\pi_1, \ldots, \pi_k)$ *such that*

$$\lim_{n\to\infty} P_{ij}^n = \pi_i$$

Proposition 2.4.4. *If Proposition 2.4.3 is satisfied for every* i *and* j*, then* $\pi = (\pi_1, \ldots, \pi_k)$ *are necessary, non-negative, and* $\sum_{i=1}^{k} \pi_i = 1$*. Thus, when consensus is reached, the common subjective probability distribution of each of the* k *agents will be*

$$\sum_{i=1}^{k} \pi_i \cdot F_i.$$

2.4.4 Conditions for Convergence

Theorem 2.4.5. *If there exists at least one positive integer n such that the n-th column of the matrix P^n consists of positive elements, then a consensus will be reached. In simple words, if there is an agent that affects the opinion of everybody in the group, consensus will be reached.*

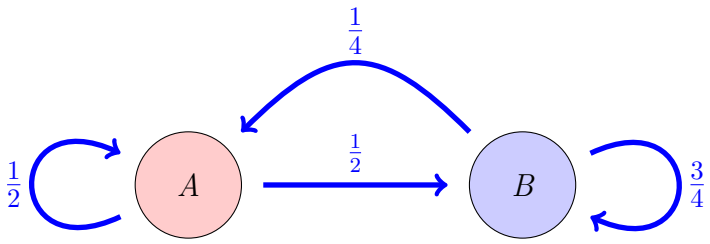
Theorem 2.4.6. *If all the recurrent states (agents) of the Markov chain communicate with each other and are aperiodic, then a consensus will be reached. In simple words, if there exists a direct or indirect connection between agents, and the cycle of connection is not periodic, then consensus will be reached.*

2.4.5 Calculation of the Consensus

Theorem 2.4.7. *Suppose consensus is reached, and let $\sum_{i=1}^{k} \pi_i \cdot F_i$ denote the common subjective distribution. Then, $\pi = (\pi_1, \ldots, \pi_k)$ is the unique stationary vector.*

Example 2.4.3. *Consider the transition matrix*

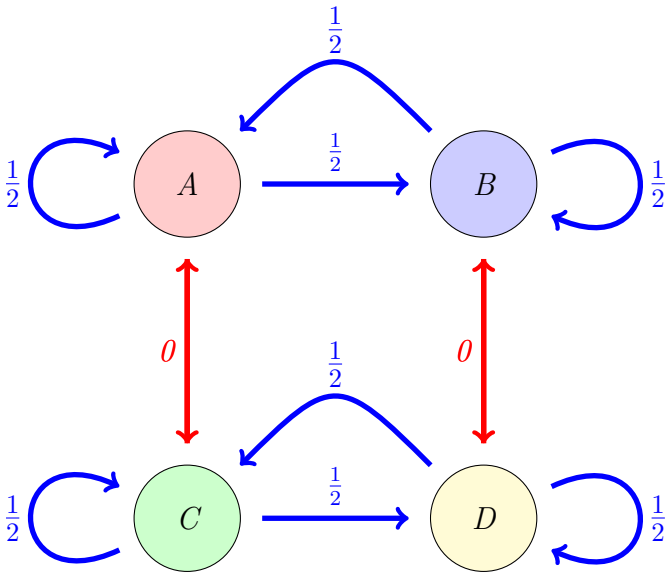
$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \end{pmatrix}$$



There is a unique stationary distribution $\pi = \left(\frac{1}{3}, \frac{2}{3}\right)$. Therefore, by Theorem 2.4.7, the consensus distribution of both agents will converge to $\frac{1}{3} \cdot F_1 + \frac{2}{3} \cdot F_2$.

Example 2.4.4. *Consider the transition matrix*

$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$



In this example, there is no global consensus distribution. However, the pairs of agents have reached local consensus with the following local consensus distributions

$$\frac{1}{2} \cdot F_1 + \frac{1}{2} \cdot F_2 \quad \textit{and} \quad \frac{1}{2} \cdot F_3 + \frac{1}{2} \cdot F_4$$

Chapter 3

Networks & Python

3.1 Network Elements

Definition 3.1.1. *A network G consists of two main components:*

- *A set of N elements, called **nodes** or **vertices**.*
- *A set of pairs of nodes, called **links** or **edges**. The link (i, j) joins the nodes i and j.*

Definition 3.1.2. *A network can be **directed** or **undirected**:*

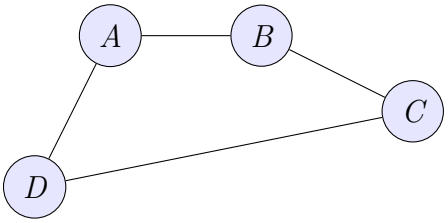
- *A **directed network**, also called a **digraph**, has directed links where the order of the nodes in a link reflects the direction.*
- *An **undirected network** has bi-directional links, and the order of the nodes in a link does not matter.*

Definition 3.1.3. *A network can be **weighted** or **unweighted**:*

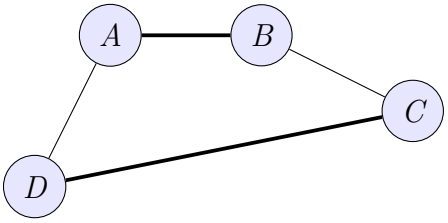
- *In a **weighted network**, links have associated weights. The weighted link (i, j, w) represents a connection between nodes i and j with a weight w.*

- *In an **unweighted network**, all links have equal weight, typically* 1 *or* 0 *for the presence or absence of a link.*

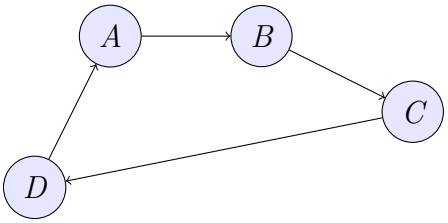
Example 3.1.1. *The following is an example of an undirected and unweighted network:*



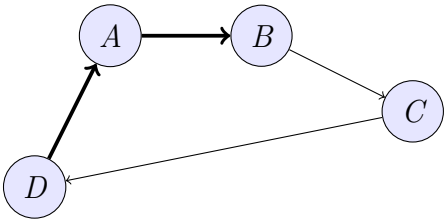
The following is an example of an undirected and weighted network where thicker lines have weight twice as much as the thinner ones:



The following is an example of an directed and unweighted network: :



The following is an example of an directed and weighted network where thicker lines have weight twice as much as the thinner ones:



3.2 Handling Network in Python

3.2.1 Undirected Graph

Importing the relevant librarries

```
import matplotlib.pyplot as plt
import networkx as nx
```

This line tells Jupyter notebook to draw graphics inline

```
%matplotlib inline
```

Creating the graph

```
Graph =  nx.Graph()
```

Adding nodes

```
Graph = nx.Graph()
Graph.add_node('A')
nodes_to_add =['B', 'C', 'D']
Graph.add_nodes_from(nodes_to_add)
```

Adding Edges

```
Graph.add_edge('A','B')
edges_to_add = [('a', 'C'), ('B', 'C'), ('C', 'D')]
Graph.add_edges_from(edges_to_add)
```

Represent graphically

```
nx.draw(Graph, with_labels=True)
```

Save to PDF

```
plt.savefig('output_graph.pdf')
```

Additional ways to customize the graph (excluding the execution of this code section).

```
nx.draw(Graph,
        with_labels=True,
        node_color='blue',
        node_size=1600,
        font_color='white',
        font_size=16,)
```

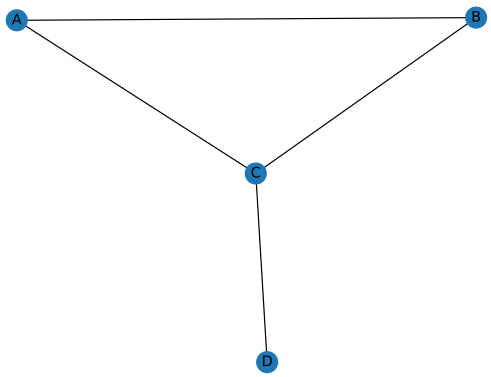


Fig. 3.1: Undirected Graph resulted by the code above

3.2.2 Graph Methods

Listing nodes

```
Graph.nodes()
```

Listing edges

```
Graph.edges()
```

Iteration over edges

```
for edge in G.edges
```

iteration over nodes

```
for node in Graph.nodes:
```

Number of nodes

```
G.number_of_nodes()
```

Number of edges

```
G.number_of_edges()
```

Creating list of neighbors

```
neighbors_of_A = list(Graph.neighbors('A'))
```

3.2.3 Node and Edge Existence

To check if a node is present in a graph

```
Graph.has_node('A')
or
'D' in G.nodes
```

To check if two nodes are connected by an edge:

```
Graph.has_edge('A', 'B')
or
('C', 'D') in Graph.edges
```

3.2.4 Predecessor,Successors and neighbors

Definition 3.2.1. *We define*

- ***Predecessor****: If there is a directed edge (i, j), then i is a predecessor of j.*
- ***Successors****: If there is a directed edge (i, j), then j is a successor of i.*
- ***Neighbors****: If there is an edge (i, j), then j is a neighbor of i and i is a neighbor of j.*

To check number of predecessor of a node

```
Graph.predecessors
```

To check number of successors of a node

```
Graph.successors
```

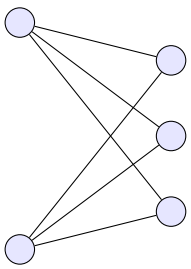
To check number of neighbors of a node

```
Graph.neighbors
```

3.2.5 Networks Structures

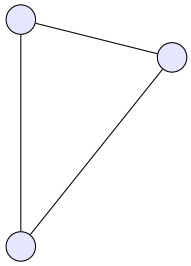
To draw a bipartite graph

```
graph = nx.complete_bipartite_graph(2,3)
```



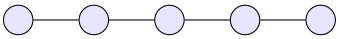
To draw a cycle graph

```
graph = nx.cycle_graph(3)
```



To draw a path graph

```
graph = nx.cycle_graph(5)
```



3.3 Density and Sparsity

Definition 3.3.1. *The maximum number of links in an undirected network with N nodes is the number of distinct pairs of nodes, which can be expressed as*

$$L_{max} = \binom{N}{2} = \frac{N(N-1)}{2}\,.$$

Every node can be connected to $N-1$ other nodes, but we have to divide by two because a link from node A to node B is considered the same as a link from node B to node A.

Definition 3.3.2. *We define*

- *the density of a network with N nodes and L links is given by:*

$$d = \frac{L}{L_{max}}\,;$$

- *density in an undirected network*

$$d = \frac{2L}{N(N-1)};$$

- *density in a directed network*

$$d = \frac{L}{N(N-1)}.$$

Definition 3.3.3. *Let us define the following structure of networks:*

- *a complete network is a network that has* $d = 1$.
- *a sparse network is a network that has* d *smaller than 1.*
- *in the case of a large network with an increasing number of nodes:*
  - *If the number of links is growing proportionally to the number of added nodes, the network is considered sparse.*
  - *If the number of links is growing at a rate faster than linear with respect to the number of added nodes, the network is considered dense.*

To check density of the network

```
nx.density(Graph)
```

3.4 Subnetwork

Creating a complete graph with 10 Nodes.

```
k10 = nx.complete_graph(10)
```

Creating A Subnetwrok that includes Nodes 0,2,4,8 from the 10 graph.

```
sub_network_of_k10 = K10.subgraph([0, 2, 4, 8])
```

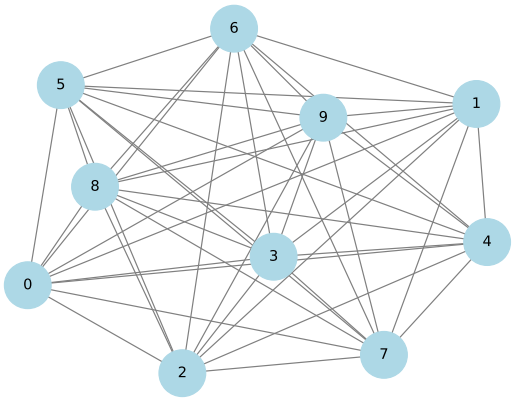


Fig. 3.2: k10 graph resulted by the code above.

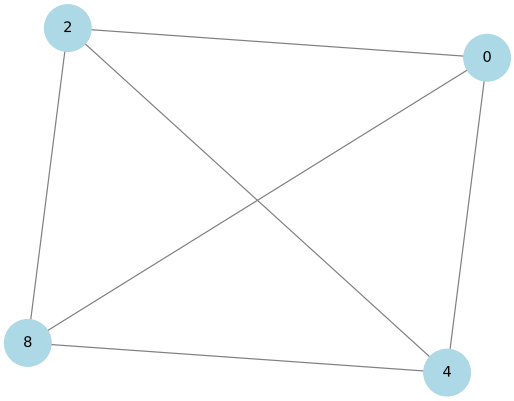


Fig. 3.3: The subgraph of k10 resulted by the code above.

3.5 Degree

To check the Degree of a node (Number of neighbors).

```
len(list(Graph.neighbors('A')))
or
Graph.degree('A')
```

To check the number of successors.

```
Graph.out_degree
```

To check the number of predecessors.

```
Graph.in.degree
```

3.6 Wieght

Definition 3.6.1. *We define*

- *the weight w_{ij} of an edge from node i to node j represents the strength of the connection between them.*
- *the weighted degree of a node i in an undirected graph is given by:*

$$s_i = \sum_j w_{ij}$$

- *the weighted in-degree of a node i in a directed graph is given by:*

$$s_i^{in} = \sum_j w_{ji}$$

- *The weighted out-degree of a node i in a directed graph is given by:*

$$s_i^{out} = \sum_j w_{ij}$$

3.7 The Erdos-Renyi mode

Definition 3.7.1. *The Erdos-Renyi model, denoted as $G(n, p)$, is a classic model for generating random graphs. In this model, a graph is constructed by connecting n labeled nodes randomly. Each possible edge between two nodes is included in the graph with probability p, independently of every other edge. Formally, the probability of generating a specific graph with n nodes and M edges is given by:*

$$p^M (1-p)^{\binom{n}{2}-M}$$

In simpler terms, the algorithm for generating a graph using the $G(n, p)$ model iterates over every possible edge between nodes ($\binom{n}{2}$ possible edges). For each edge, it is created with probability p. As p varies from 0 to 1, the model transitions from graphs with very few edges to graphs with many edges. When $p = \frac{1}{2}$, every graph on n vertices is equally likely, and the total number of possible graphs is $2^{\binom{n}{2}}$.

Here is an example of how to use this function in Python:

```
G = nx.erdos_renyi_graph(n, p, seed=None, directed=False)
```

- `n` - Number of nodes in the graph.
- `p` - Probability of edge creation.
- `seed` - Seed for the random number generator to ensure reproducibility.
- `directed` - A boolean parameter that specifies whether the graph is directed (True) or undirected (False).

3.8 Analysis of the DeGroot Model Behavior Netweok Approach

The goal of this experiment is to investigate how the number of random predecessors affects the likelihood of reaching consensus in a directed graph.

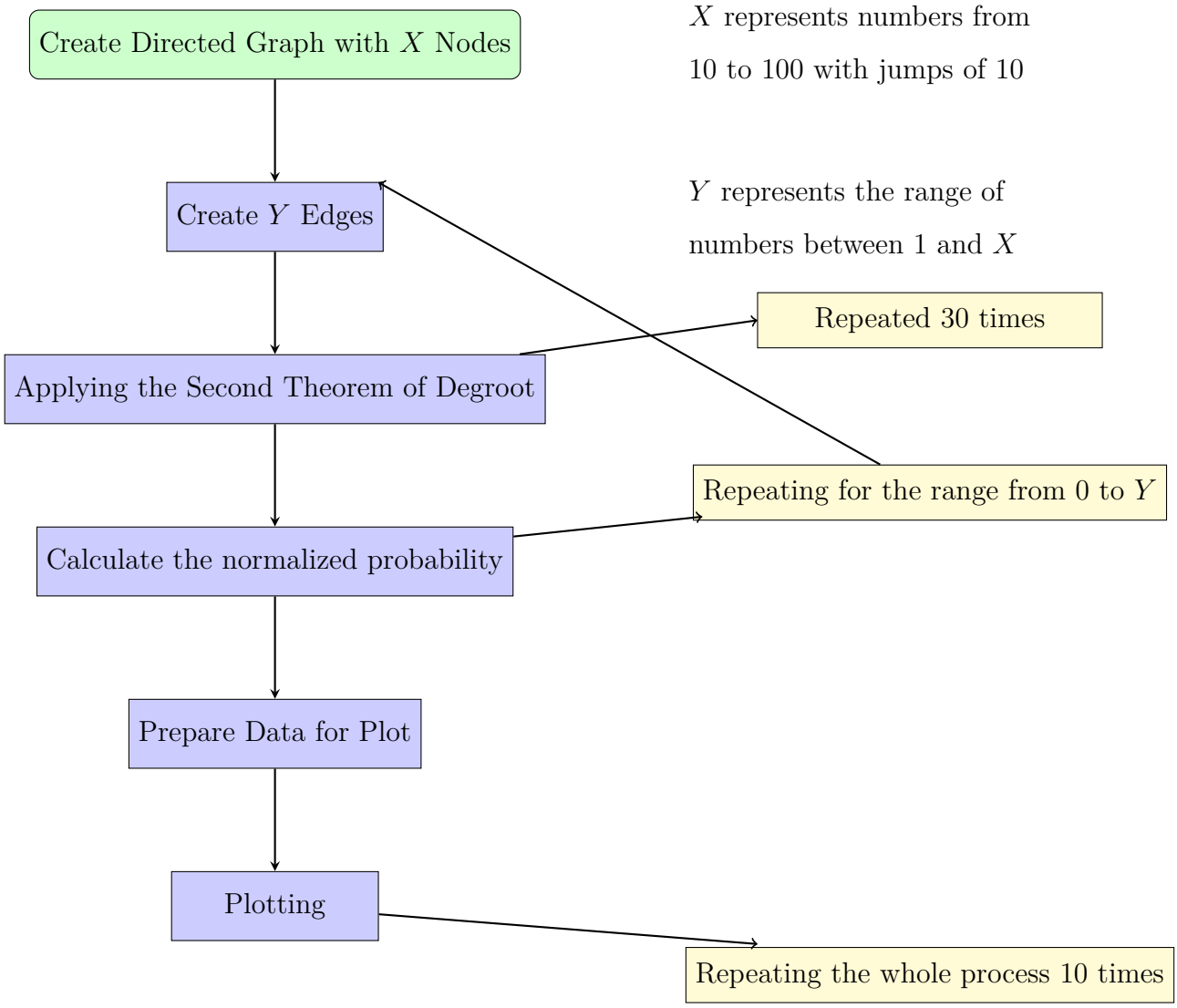
1. We begin by constructing a directed graph with X nodes, where X is an integer ranging from 10 to 100, with increments of 10.
2. For each graph, we perform Y iterations, where Y is an integer ranging from 1 to $X - 1$, with increments of 1. Here, Y represents the number of random predecessors allocated to each node.

3. The consensus of the network is evaluated using the second DeGroot's theorem (i.e. Theorem 2.4.6). The theorem assesses whether the network can reach consensus under the given configuration of random predecessors.

4. Each allocation of Y predecessors is repeated 30 times to ensure the results are statistically significant and to normalize the data. Instead of a binary outcome, we calculate the probability of reaching consensus.

5. Once the number of links per node reaches the total number of nodes minus one, the aggregated data is used to create a bar plot graph. This visualization depicts the probability of consensus as a function of the number of random predecessors.

6. Because we have ten graphs, we will have ten bar plots. The results from the bar plots will help determine how varying the number of random predecessors influences the likelihood of the network achieving consensus.



We are starting with importing the relevant libraries

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import networkx as nx
import random
```

Initialize global counters

```
number_of_consensus = 0
```

```
number_of_recurrent_and_not_aperiodic = 0
number_of_non_recurrent_and_not_aperiodic = 0
number_of_non_recurrent_and_aperiodic = 0
```

This function takes an argument a number and returns a directed graph with the corresponding number of nodes.

```
def creating_directed_graph(number_of_nodes):
    G = nx.DiGraph()
    nodes_to_add = range(number_of_nodes)
    G.add_nodes_from(nodes_to_add)
    return G
```

This function takes a graph and a number as arguments and returns a directed graph with each node having the corresponding number of random predecessors.

```
def assigning_edges(G, number_of_edges_per_node):
    for node in G.nodes:
        possible_targets = [target for target in G.nodes if
           target != node]
        targets = random.sample(possible_targets,
           number_of_edges_per_node)
        for target in targets:
            G.add_edge(target, node)
```

This function checks if the graph can reach consensus, and if not, it determines the reason.

```
def second_Degroot_theorem(G):
    global number_of_consensus,
       number_of_recurrent_and_not_aperiodic,
       number_of_non_recurrent_and_not_aperiodic,
       number_of_non_recurrent_and_aperiodic
```

```
    if nx.is_strongly_connected(G) and nx.is_aperiodic(G):
        number_of_consensus += 1
    elif nx.is_strongly_connected(G) and not nx.is_aperiodic
       (G):
        number_of_recurrent_and_not_aperiodic += 1
    elif not nx.is_strongly_connected(G) and not nx.
       is_aperiodic(G):
        number_of_non_recurrent_and_not_aperiodic += 1
    elif not nx.is_strongly_connected(G) and nx.is_aperiodic
       (G):
        number_of_non_recurrent_and_aperiodic += 1
```

This function is the core of the code. For every possible number of random predecessors for each node, the function calculates the probability of reaching consensus and aggregates all the results into a list ready for plotting.

```
def experiment(number_of_nodes, trials=30):
    global number_of_consensus,
       number_of_recurrent_and_not_aperiodic,
       number_of_non_recurrent_and_not_aperiodic,
       number_of_non_recurrent_and_aperiodic
    list_of_probability_distribution = []
    for n in range(1, number_of_nodes):
        number_of_consensus = 0
        number_of_recurrent_and_not_aperiodic = 0
        number_of_non_recurrent_and_not_aperiodic = 0
        number_of_non_recurrent_and_aperiodic = 0
        for _ in range(trials):
            G = creating_directed_graph(number_of_nodes)
            assigning_edges(G, n)
```

```
12              second_Degroot_theorem(G)
13          probability_consensus = number_of_consensus / trials
14          probability_non_recurrent_and_not_aperiodic =
                number_of_non_recurrent_and_not_aperiodic /
                trials
15          probability_recurrent_and_not_aperiodic =
                number_of_recurrent_and_not_aperiodic / trials
16          probability_non_recurrent_and_aperiodic =
                number_of_non_recurrent_and_aperiodic / trials
17          list_of_probability_distribution.append((
                probability_consensus,
                probability_non_recurrent_and_not_aperiodic,
                probability_recurrent_and_not_aperiodic,
                probability_non_recurrent_and_aperiodic))
18      return list_of_probability_distribution
```

In this function, we plot the graph. Notice that we are using the `ax.set` method and not the regular `plt` method. `ax` represents the coordinates of the specific plot. We are planning to have 10 plots, so each plot needs to have its own index.

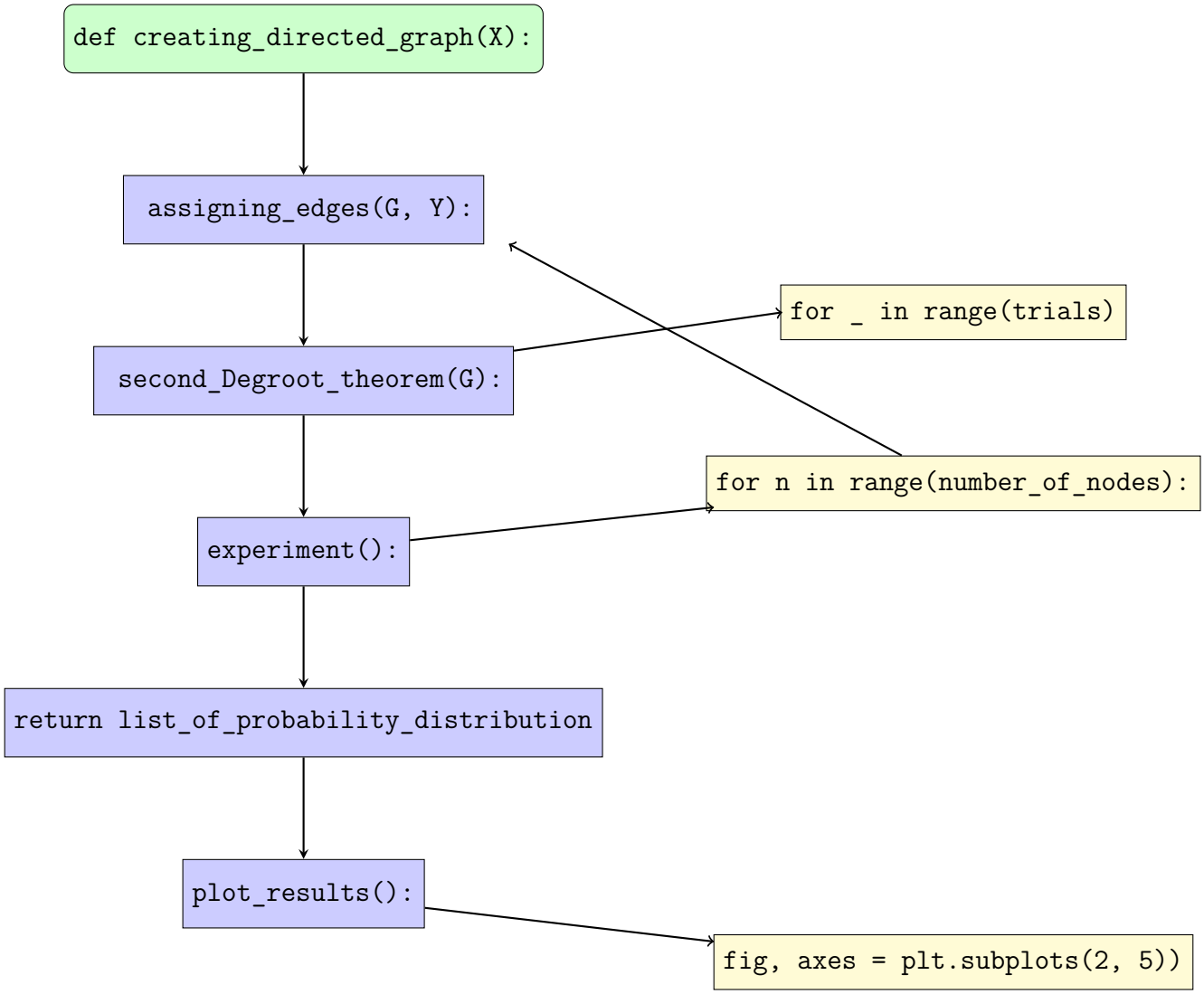
```
1 def plot_results(ax, list_of_probability_distribution,
      number_of_nodes):
2     n_values = range(len(list_of_probability_distribution))
3     probability_consensus_values = [result[0] for result in
          list_of_probability_distribution]
4     probability_non_recurrent_and_not_aperiodic_values = [
          result[1] for result in
          list_of_probability_distribution]
5     probability_recurrent_and_not_aperiodic_values = [result
          [2] for result in list_of_probability_distribution]
```

```
    probability_non_recurrent_and_aperiodic_values = [result[3] for result in list_of_probability_distribution]
    bar_width = 0.4
    ax.bar(n_values, probability_consensus_values, color='blue', width=bar_width, edgecolor='blue', label='Probability of Consensus')
    ax.bar(n_values, probability_non_recurrent_and_not_aperiodic_values, color='red', width=bar_width, edgecolor='red', label='Probability of Non-Consensus and Not Aperiodic', bottom=probability_consensus_values)
    ax.bar(n_values, probability_recurrent_and_not_aperiodic_values, color='green', width=bar_width, edgecolor='green', label='Probability of Recurrent and Not Aperiodic', bottom=[i+j for i,j in zip(probability_consensus_values, probability_non_recurrent_and_not_aperiodic_values)])
    ax.bar(n_values, probability_non_recurrent_and_aperiodic_values, color='orange', width=bar_width, edgecolor='orange', label='Probability of Non-Recurrent and Aperiodic', bottom=[i+j+k for i,j,k in zip(probability_consensus_values, probability_non_recurrent_and_not_aperiodic_values, probability_recurrent_and_not_aperiodic_values)])
    ax.set_xlabel('Number of Edges per Node (n)', fontsize=10)
    ax.set_ylabel('Probability', fontsize=10)
    ax.set_title(f'Nodes: {number_of_nodes}', fontsize=12)
    ax.legend(fontsize=8)
```

```
16      ax.tick_params(axis='both', which='major', labelsize=8)
17      ax.grid(False)
```

This function creates a 2x5 grid of plots. It runs everything described before 10 times, creating 10 directed graphs with X number of nodes, where X ranges from 10 to 100 in increments of 10. Thus, it creates 10 corresponding bar plots. Eventually, the program saves the grid as a PDF.

```
1  with PdfPages('consensus_plots2.pdf') as pdf:
2      fig, axes = plt.subplots(2, 5, figsize=(20, 10))
3
4      for idx, number_of_nodes in enumerate(range(10, 101, 10)
           ):
5          ax = axes[idx // 5, idx % 5]
6          probability_distribution = experiment(
               number_of_nodes)
7          plot_results(ax, probability_distribution,
               number_of_nodes)
8      plt.tight_layout()
9      pdf.savefig(fig)
10     plt.show()
11     plt.close(fig)
```



Code output:

- **Blue**: Probability of reaching consensus.
- **Yellow**: Probability of not reaching consensus because the system is not recurrent but aperiodic.
- **Red**: Probability of not reaching consensus because the network is neither recurrent nor aperiodic.

The plots are indexed as follows:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

The y-axis represents the probability, while the x-axis represents the number of random predecessors per node.

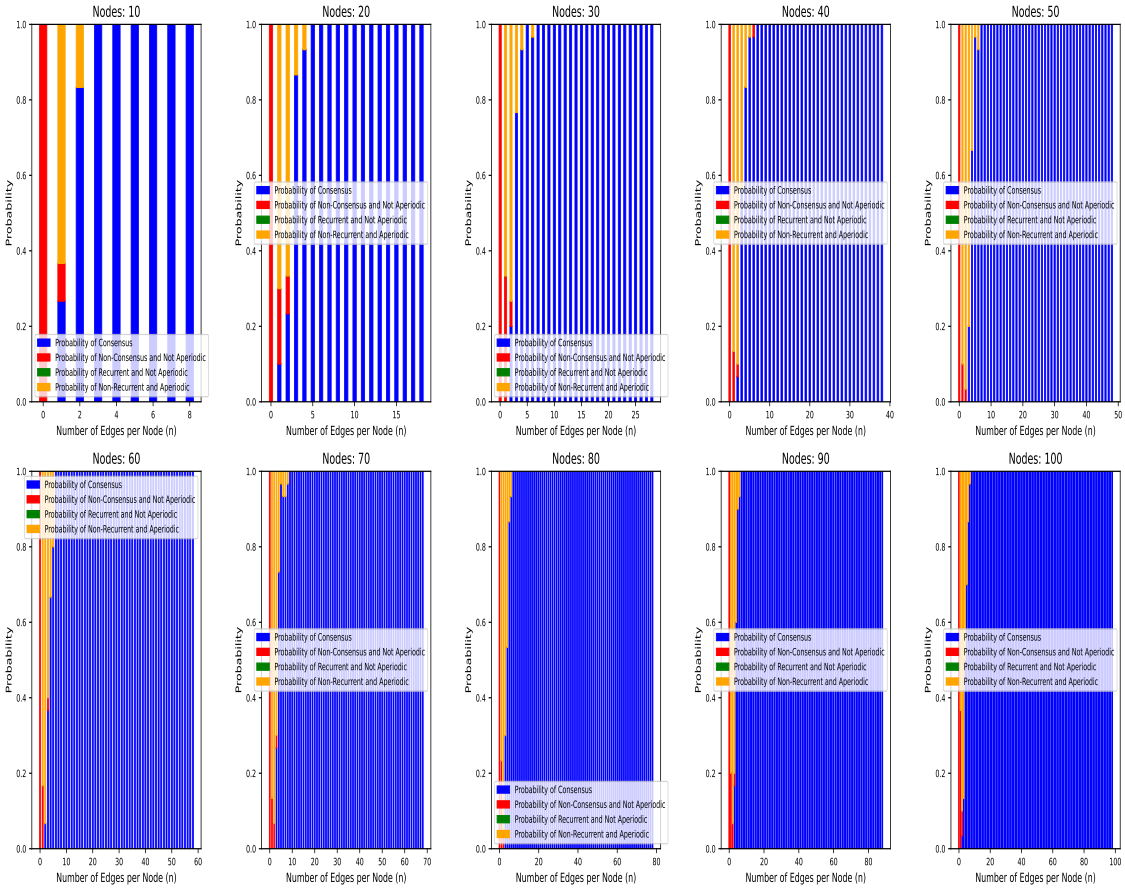


Fig. 3.4: A graph resulting from the code above.

Chapter 4

Analysis of the dynamic DeGroot Model Behavior in Erdos-Renyi Graphs

4.1 Introduction

This chapter aims to analyze the behavior of the module in a simulated environment and to verify if the resulting outcomes correspond to the expected values as determined by theoretical mathematics. The analysis will be organized into four sections:

1. **Environment Creation**: An explanation of the code used to create the simulation environment.

2. **Experiment Setups**: Detailed descriptions of five experimental setups using flexible code from section 1.

3. **Results**: Presentation of the outcomes of each experiment, plotted in four graphs.

4. **Discussion**: A brief discussion of the results obtained from the experiments.

4.2 Environment Creation

- `networkx` is a library for creating and manipulating complex networks.
- `quantecon` is a library that has specialized methods to deal with Markov chains and other economic models.
- `numpy` is a library for linear algebra operations and other numerical computations.
- `random` is a module that has methods related to random processes.
- `matplotlib` is a library to plot data.
- `matplotlib.backends.backend_pdf` helps save the plot of subplots as one file.

```
1 import networkx as nx
2 import quantecon as qe
3 import numpy as np
4 import random
5 import matplotlib.pyplot as plt
6 from matplotlib.backends.backend_pdf import PdfPages
```

This function takes as arguments:

- `n` - number of nodes in the graph
- `y` - number of links per node, including the self loop
- `w` - weight. Each node will have a weight of `w` for the self loop, and all the successors will have $(1 - \texttt{w})$/number of successors

The function returns `t` - the transition matrix of the graph by:

- creating an `n` x `n` matrix
- every row represents the weight distribution of each node, so for each row, the function will assign a self loop with weight `w`
- for each node, the function will select `y-1` random other nodes excluding itself and assign $(1 - \texttt{w})$/number of successors to each successor

```
def create_directed_graph_random_successors(n, y, w):
    if not isinstance(n, int) or n <= 0:
        raise ValueError("The size parameter must be a
           positive integer.")
    t = np.zeros((n, n))
    for i in range(n):
        t[i, i] = w
        if y - 1 > n - 1:
            raise ValueError("Number of successors must be
               less than number of nodes.")
        succ_weigh = (1 - w) / (y - 1)
        random_indices = [j for j in range(n) if j != i]
        selected_indices = np.random.choice(random_indices,
           size=y - 1, replace=False)
        for j in selected_indices:
            t[i, j] = succ_weigh
        if t.shape[0] != t.shape[1]:
        raise ValueError(f"Matrix is not square: shape = {t.
           shape}")
    return t
```

- `t` - number of nodes in the graph
- `p2` - edge rearrangement probability, the probability that every possible node has to eliminate a link, excluding the self-loop

The function returns `t`, the transition matrix by:

- excluding nodes that have only self-loop, The function will iterate over each node
- with probability p2 a node will eliminate a link and will create a new link with another node with the same weight to retain the stochastic properties of the matrix

The function takes the following arguments:

- `n` - Number of nodes in the graph.
- `p1` - Probability of edge creation in the Erdos-Renyi model.
- `w` - Weight for the self-loop. Each node will have a weight of `w` for the self-loop, and the remaining weight will be distributed equally among the outgoing links (excluding the self-loop).

The function returns `t` - the transition matrix of the graph by:

- Creating an `n` x `n` matrix.
- Ensuring that every node has a self-loop with weight `w`. If a self-loop does not exist initially, it will be added.
- Distributing the remaining weight $(1 - \texttt{w})$ equally among the outgoing links for each node (excluding the self-loop).

```
def creating_directed_weighted_erdos_renyi_graph(n, p1, w):
    G = nx.erdos_renyi_graph(n, p1, seed=None, directed=True
       )
    for node in G.nodes():
        if not G.has_edge(node, node):
            G.add_edge(node, node)
    t = nx.to_numpy_array(G)
    for i in range(len(t)):
        c1 =np.sum(t[i] == 1)
        if c1 == 1:
            continue
        t[i][i]= w
        for j in range(len(t)):
            if i !=j and t[i][j] != 0:
                t[i][j]= (1-w)/(c1-1)
    return t
```

This function takes as arguments:

- `t` - Transition matrix, which must be a 2D square matrix.
- `p2` - Edge rearrangement probability, the probability that every possible node has to eliminate a link, excluding the self-loop.

This function returns the updated transition matrix `t` by:

- Iterating over each node. With probability `p2`, it eliminates one of the successors and creates a link with a new successor, maintaining the same assigned weight to preserve the stochastic properties of the matrix.

```
def rearrange_edges(t, p2):
    rows, cols = t.shape
    for i in range(rows):
        if t[i, i] == 1:
            continue
        if np.random.rand() >= p2:
            continue
        row = t[i, :]
        non_empty = np.nonzero(row)[0]
        empty = np.where(row == 0)[0]
        if len(non_empty) == 0 or len(empty) == 0 or  len(
           non_empty) == 1:
            continue
        random_non_empty = np.random.choice(non_empty)
        random_empty = np.random.choice(empty)
        if random_non_empty == i:
            continue
        t[i, random_empty] = t[i, random_non_empty]
        t[i, random_non_empty] = 0

    return t
```

This function takes as arguments:

- `n` - the number of agents (or nodes) in the graph.

The function returns o ,opinion vector by:

- Generates random values uniformly distributed between 0.01 and 0.99.
- Returns an array of size n, where each element is a random value within the

specified range.

```
def generate_random_vector(n):
    if not isinstance(n, int) or n <= 0:
        raise ValueError("The size parameter must be a
           positive integer.")
    random_values = np.random.uniform(0.01, 0.99, n)
    rounded_values = np.round(random_values, 3)
    return rounded_values
```

This function takes as arguments:

- `t` - Transition matrix, which must be a 2D square matrix.
- `o` - Opinion vector, which must be a 1D array of length equal to the number of rows (or columns) in `t`.

The function returns `result` - The converged consensus by:

- **Matrix Validation**: Ensure that `t` is a 2D square matrix.
- **Stationary Distribution Calculation**: Utilize a specialized Markov chain library to compute the stationary distribution of the transition matrix. This distribution is the eigenvector corresponding to the eigenvalue 1.
- **Converged Consensus Calculation**: Compute the converged consensus by performing a weighted sum. Multiply the transition matrix by the opinion vector. Ensure that the matrix and vector dimensions are compatible and that the order of operations is correctly maintained.

```
def converged_consensus(t, o):
    if not (isinstance(t, np.ndarray) and len(t.shape) == 2
       and t.shape[0] == t.shape[1]):
```

```
        raise ValueError(f"Transition matrix is not square:
            shape = {t.shape}")
    mc = qe.MarkovChain(t)
    st = mc.stationary_distributions
    first_st = st[0]
    if len(first_st) != len(o):
        raise ValueError("Dimension mismatch between
            stationary distribution and opinion vector.")
    result = np.dot(first_st, o)
    return result
```

This function checks if the transition matrix satisfies the second DeGroot theorem by verifying if the matrix is aperiodic and irreducible. In simple terms, it checks:

- Whether there exists a path of finite length p between every pair of states, ensuring that every state can be reached from every other state. This confirms the **irreducibility** of the matrix.
- Whether the Markov chain does not have a fixed cycle length for returning to the same state, which means it is **aperiodic**. This ensures that the number of steps required to return to a given state does not follow a predictable pattern.

By verifying these conditions, the function ensures that the transition matrix represents a Markov chain where the long-term behavior converges to a unique steady-state distribution, as guaranteed by the second DeGroot theorem.

```
def degroot_theorem(t):

    mc = qe.MarkovChain(t)
```

```
4     is_irreducible = mc.is_irreducible
5     is_aperiodic = mc.is_aperiodic
6     return is_irreducible and is_aperiodic
```

This function takes as arguments:

- `t` - Transition matrix, which must be a 2D square matrix.
- `o` - Opinion vector, which must be a 1D array of length equal to the number of rows (or columns) in `t`.
- `p2` - Edge rearrangement probability, the probability that every possible node has to eliminate a link, excluding the self-loop.

The function returns the following counted data:

- `c` - Number of iterations it takes to reach a consensus.
- `dc/c` - the ratio of iteration with states that satisfied the second Degroot theorem to the total number of iterations
- `dco` - The difference between the theoretical converged opinion and the average value in the opinion vector at the time of final iteration.
- `dmm` - The difference between the highest value and the lowest value in the opinion vector.
- Start a loop until the system converges to consensus. In this system, consensus is considered a state in which the absolute difference between the highest value and the lowest value in the opinion vector does not exceed 0.1. This value can be adjusted to obtain more precise results.

- The function returns `c`, `dc/c`, `dco`, and `dmm`. The function returns `dc/c` to provide a probability value rather than a count, making the analysis more feasible.

```
def counter(t,o,p2):
    c =0
    dc =0
    dco =0
    dmm = 0
    (c_s) = converged_consensus(t,o)
    while abs(np.max(o) - np.min(o)) >0.1:
        if c >= 100:
            mean_o = np.mean(o)
            dmm = abs(np.max(o) - np.min(o))
            print(np.max(o) ,np.min(o))
            dco =abs(mean_o-c_s)
            return np.nan,  dc/c, dco ,dmm
        rearrange_edges(t, p2)
        o = np.dot(t,o)
        if degroot_theorem(t):
            dc += 1
        c += 1
    dmm = abs(np.max(o) - np.min(o))
    mean_o = np.mean(o)
    dco =abs(mean_o-c_s)
    return c, dc/c, dco , dmm
```

This function takes the next arguments

- `n` - Number of nodes in the graph.

- `w` - Weight for the self-loop. Each node will have a weight of `w` for the self-loop.
- `p2` - Edge rearrangement probability. This is the probability that any possible node will eliminate a link, excluding the self-loop.
- `p1` - Probability of edge creation in the Erdos-Renyi model.
- `y` - Number of links per node, including the self-loop.
- `variable` - A variable that can be any of the aforementioned variables (`n`, `w`, `p2`, `p1`, or `y`). When one of these variables is chosen, it is transformed into a list of numbers that will be iterated over. The function conducts experiments using this changing variable while keeping all other variables static.

The function returns the following lists of data. The length of each list varies depending on the number of elements in the list produced by the changing variable:

- `l2` - Each element in this list represents the average number of iterations needed to reach consensus for each value of the changing variable.
- `ld2` - Each element in this list represents the average probability of being in the second DeGroot state during the iterations of the counter for each value of the changing variable.
- `ldco2` - Each element in this list represents the average difference between the theoretical converged opinion and the average value in the opinion vector at the time of the final iteration for each value of the changing variable.
- `ldmm2` - Each element in this list represents the average difference between the highest and lowest values in the opinion vector for each value of the changing variable.

- `l1` - Each element in this list represents the number of iterations needed to reach consensus for one value of the changing variable.
- `ld1` - Each element in this list represents the probabilities of being in the second DeGroot state during the iterations for one value of the changing variable.
- `ldco1` - Each element in this list represents the differences between the theoretical converged opinion and the average value in the opinion vector at the time of the final iteration for one value of the changing variable.
- `ldmm1` - Each element in this list represents the differences between the highest and lowest values in the opinion vector for one value of the changing variable.
- For each element in the changing variable list, the function runs the counter 30 times to normalize the data.
- Results from each run of the counter are stored in the lists ending with `1`.
- After the normalization process, the lists ending with `1` are averaged, and the resulting averages are stored in the lists ending with `2`. Each average corresponds to the index of the current changing variable.

This function is highly versatile and can be customized for different graph creation methods. Depending on the desired graph type, you can choose the appropriate graph creator function and delete the other. Below are two example graph creator functions, demonstrating how to select and use one of them.

```
1 def lists_Creator(n, w, p2 ,p1, y, variable):
2     l2 = []
3     ld2 = []
```

```
    lco2 = []
    ldmm2 = []
    if variable == "n":
        variable_list = list(range(10, 101, 1))  # Integers
    elif variable == "w":
        variable_list = np.arange(0.1, 0.91, 0.1).tolist()
           # Floating-point
    elif variable == "p1":
        variable_list = np.arange(0.01, 1.01, 0.01).tolist()
             # Floating-point
    elif variable == "p2":
        variable_list = np.arange(0.01, 1.01, 0.01).tolist()
             # Floating-point
    elif variable == "y":
        variable_list = list(range(2,n))  # Integers
    else:
        raise ValueError("Invalid variable name")
    print(f"Processing variable: {variable}")
    print(f"Variable list: {variable_list}")
    for idx, element in enumerate(variable_list):
        print("ok")
        if variable == "n":
            n = element
        elif variable == "w":
           w = element
        elif variable == "p2":
            p2 = element
        elif variable == "y":
            y = element
```

```
30          o = generate_random_vector(n)
31          l1 = []
32          ld1 = []
33          lco1 = []
34          ldmm1 = []
35          for _ in range(30):
36              #t = create_directed_graph_random_successors(n,
                    y, w)
37              #t = create_directed_graph_random_successors(n,
                    y, w)
38              if degroot_theorem(t):
39                  c, p, dco, dmm = counter(t, o, p2)
40                  l1.append(c)
41                  ld1.append(p)
42                  lco1.append(dco)
43                  ldmm1.append(dmm)
44          average_iteration = np.nanmean(l1)
45          average_probability_of_degroot_state = np.nanmean(
                ld1)
46          average_difference_average_opinion_and_converged =
                np.nanmean(lco1)
47          average_difference_average_max_min_opinion = np.
                nanmean(ldmm1)
48          l2.append(average_iteration)
49          ld2.append(average_probability_of_degroot_state)
50          lco2.append(
                average_difference_average_opinion_and_converged)
51          ldmm2.append(
                average_difference_average_max_min_opinion)
```

```
52      return l2, ld2, lco2, ldmm2
```

This function plots and saves the plots as pdf

```
1  def plot_results(l2, ld2, lco2, ldmm2, variable, filename='
       plots.pdf'):
2
3      titles = ['Average Iteration (l2)', 'Average Probability
           of Degroot State (ld2)',
4                'Average Difference Between Average Opinion
                   and Converged State (lco2)',
5                'Average Difference Between Max and Min
                   Opinion (ldmm2)']
6
7      data = [l2, ld2, lco2, ldmm2]
8
9      fig, axs = plt.subplots(2, 2, figsize=(14, 10))
10     axs = axs.flatten()
11
12     for ax, dat, title in zip(axs, data, titles):
13         x = np.arange(len(dat))
14         bar_width = 0.4
15         ax.bar(x - bar_width/2, dat, color='purple',
               edgecolor='none', width=bar_width, align='center'
               )
16
17         ax.set_title(title)
18         ax.set_xlabel(variable)
19         ax.set_ylabel('Value')
20
```

```
        interval = max(1, len(dat) // 10)

        ax.set_xticks(x[::interval])
        ax.set_xticklabels([str(i) for i in x[::interval]],
            rotation=45)
    plt.tight_layout()

    plt.savefig(filename, format='pdf')

    plt.show()
```

4.3 DeGroot Model in Costume Random Graph Markov chains approach

4.3.1 Experiment 1-Nodes per link

```
n = 100
w = 0.5
p1 = 1
p2 = 0.1
y = 0
variable = 'y'
l2, ld2, lco2, ldmm2 = lists_Creator(n, w, p2, p1, y,
   variable)
plot_results(l2, ld2, lco2, ldmm2, variable, 'plots1.pdf')
```

4.4 DeGroot Model in Costume Erdos-Renyi Graph Markov chains approach

4.4.1 Experiment 2 -Number of Nodes

```
n = 0
w = 0.9
p1 = 0.07
p2 = 0.11
y = 9
variable = 'n'
l2, ld2, lco2, ldmm2 = lists_Creator(n, w, p2, p1, y,
   variable)
plot_results(l2, ld2, lco2, ldmm2, variable, 'plots.pdf')
```

4.4.2 Experiment 3 -Weight distribution

```
n = 100
w = 0
p1 = 0.1
p2 = 0.11
y = 9
variable = 'w'
l2, ld2, lco2, ldmm2 = lists_Creator(n, w, p2, p1, y,
   variable)
plot_results(l2, ld2, lco2, ldmm2, variable, 'plots.pdf')
```

4.4.3 Experiment 4 -Edge probability

```
n = 100
w = 0.9
p1 = 0
p2 = 0.11
y = 9
variable = 'p1'
l2, ld2, lco2, ldmm2 = lists_Creator(n, w, p2, p1, y,
   variable)
plot_results(l2, ld2, lco2, ldmm2, variable, 'plots.pdf'))
```

4.4.4 Experiment 5- Edge Rearrangement Probability

```
n = 100
w = 0.9
p1 = 0.07
p2 = 0
y = 9
variable = 'p2'
l2, ld2, lco2, ldmm2 = lists_Creator(n, w, p2, p1, y,
   variable)
plot_results(l2, ld2, lco2, ldmm2, variable, 'plots.pdf')
```

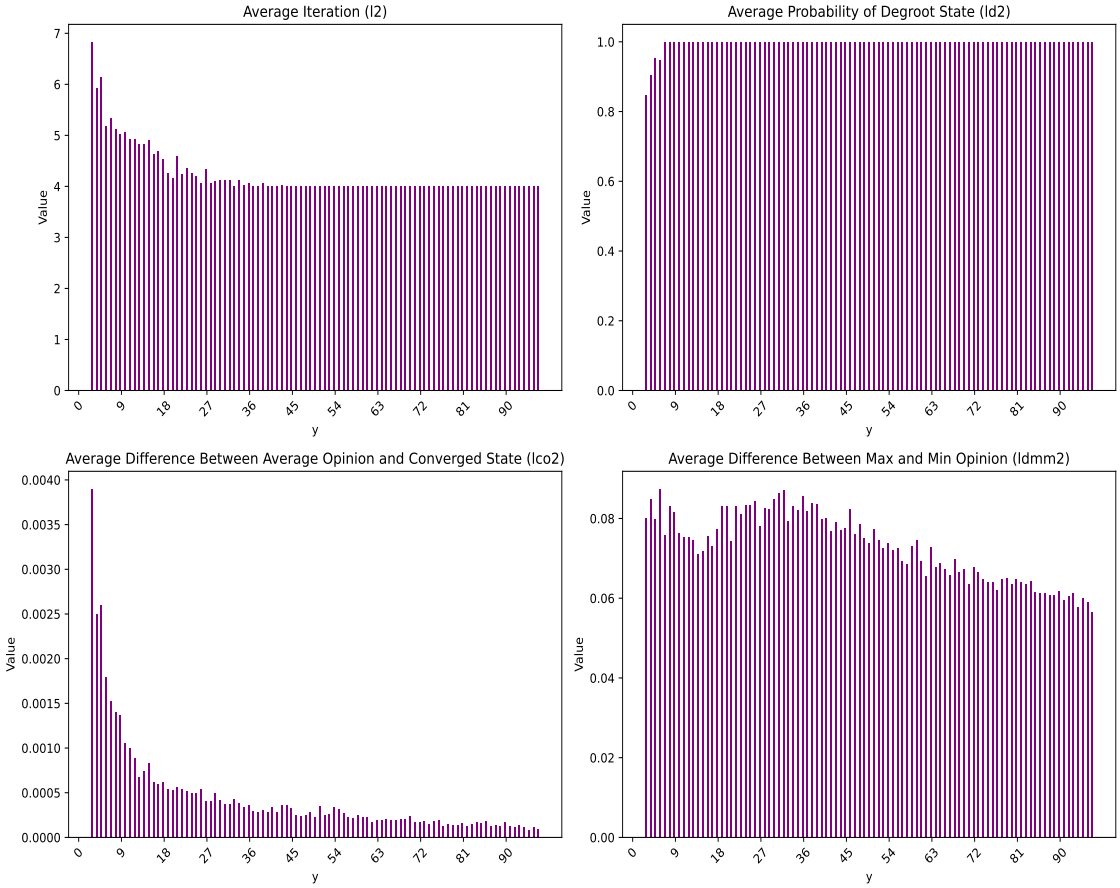


Fig. 4.1: A graph resulting from Experiment 1.

4.5 Experiment Results

4.6 Results Analysis

4.6.1 Experiment 1

Even though the experiment was constricted by the counter function, so the system didn't reach true consensus, we observe that even with a distance of around 0.1 between the extreme views, the average opinion is almost identical to the

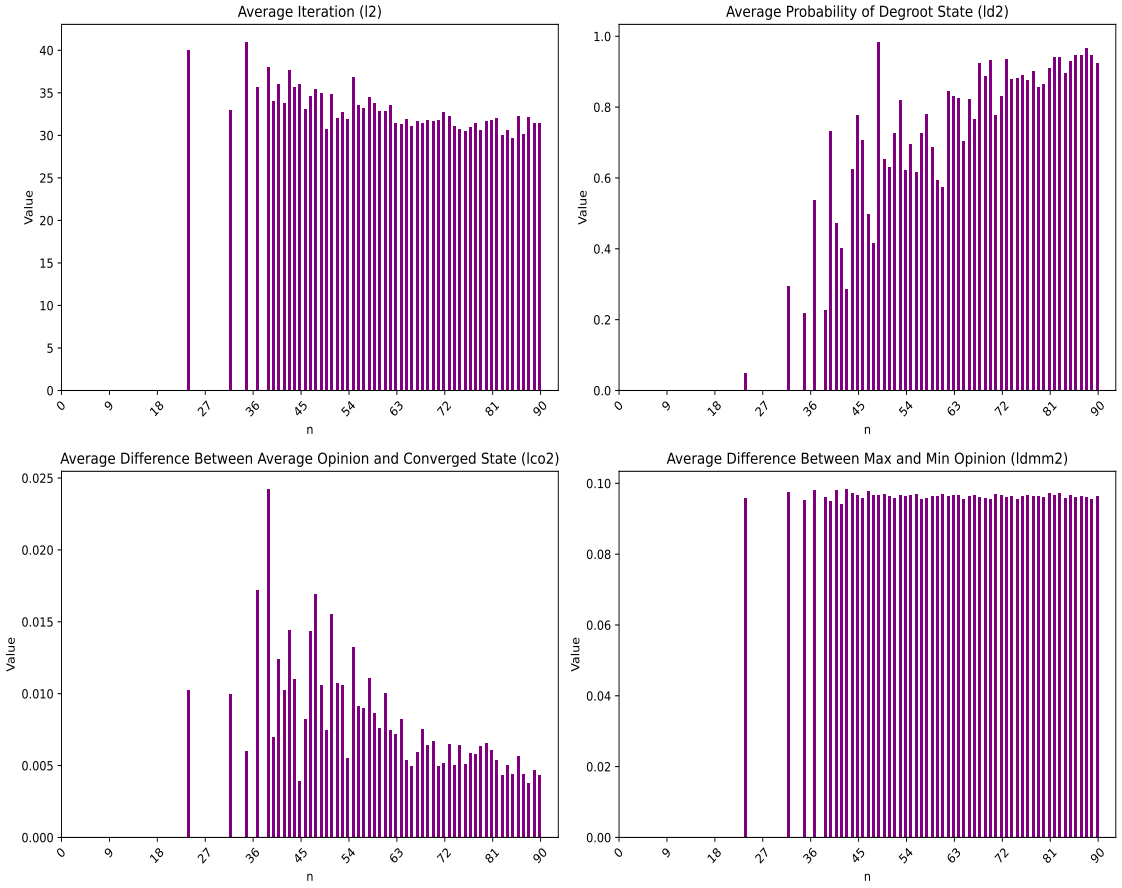


Fig. 4.2: A graph resulting from Experiment 2.

theoretical value.

Regarding the iteration, the system operates in a threshold dynamic. Up until a density of 20 percent, there is an improvement. However, beyond this threshold, additional links do not make the system reach consensus faster.

4.6.2 Experiment 2

In this experiment, we discovered that when the p_1 edge probability is fixed, there is a threshold of minimum edges required for the system to reach consensus.

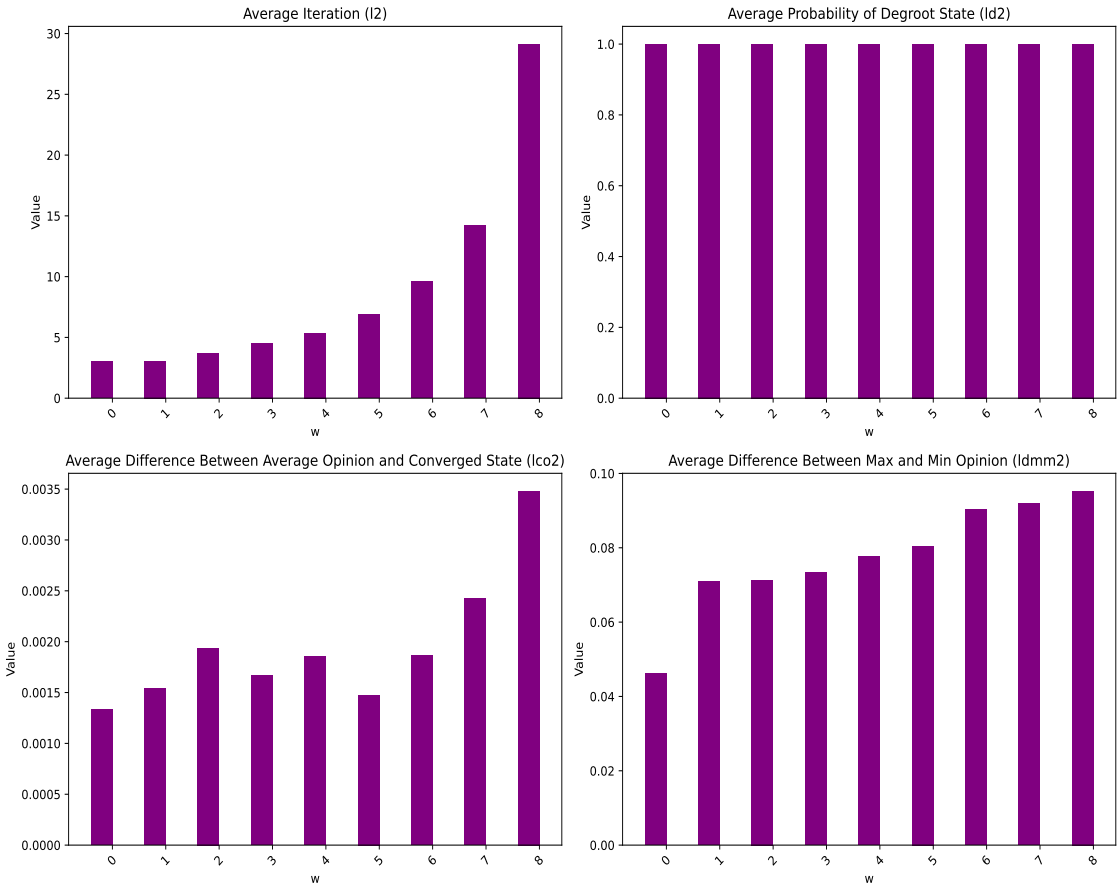


Fig. 4.3: A graph resulting from Experiment 3.

4.6.3 Experiment 3

In this experiment, we concluded that there is an exponential correlation between the weight of the self-loop and the number of iterations needed to reach consensus.

4.6.4 Experiment 4

In this experiment, we discovered that an edge probability of 3 percent is sufficient to reach a consensus. We also observed a decrease in the number of

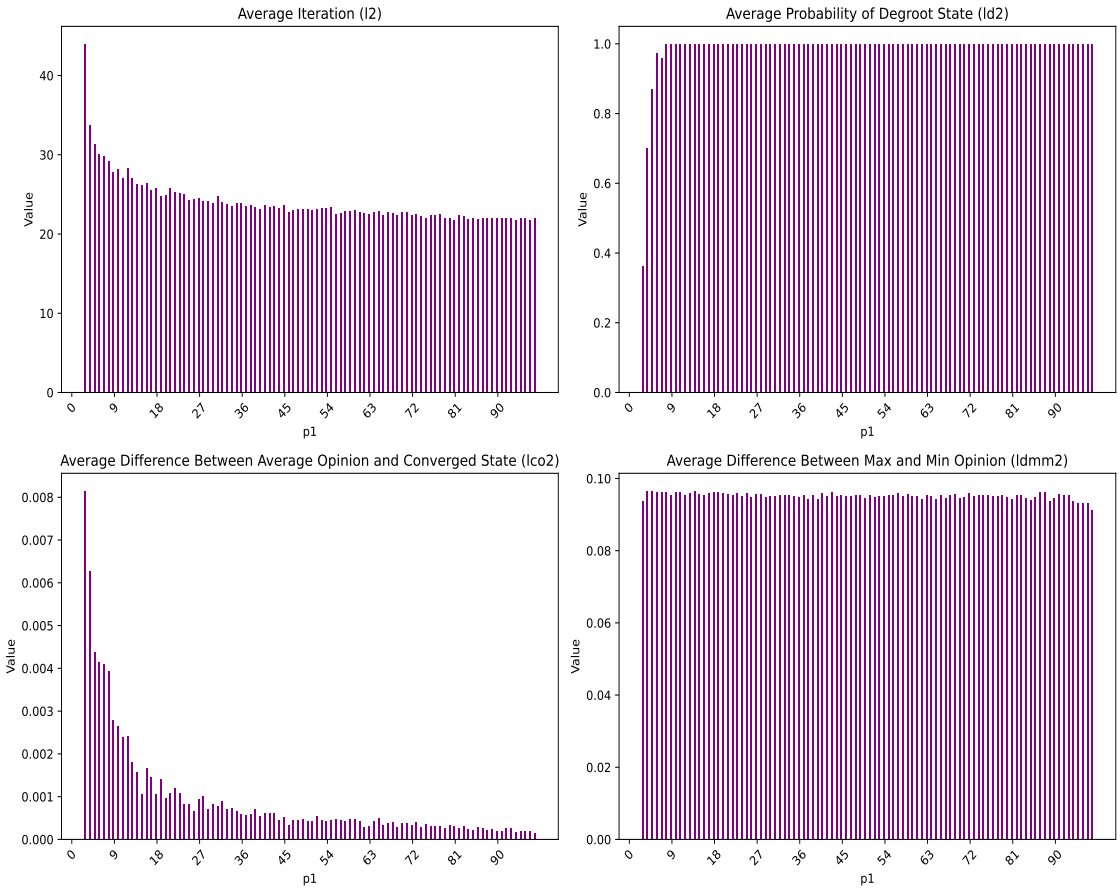


Fig. 4.4: A graph resulting from Experiment 4.

iterations needed until the edge probability reaches 18 percent, after which any additional edges do not further decrease the iterations needed to reach consensus.

4.6.5 Experiment 5

In this experiment, we concluded that the more nodes change neighbors, the less time is needed to reach consensus. In my opinion, this occurs because it eliminates the possibility of creating weakly connected sub-networks, which can develop significantly different views.

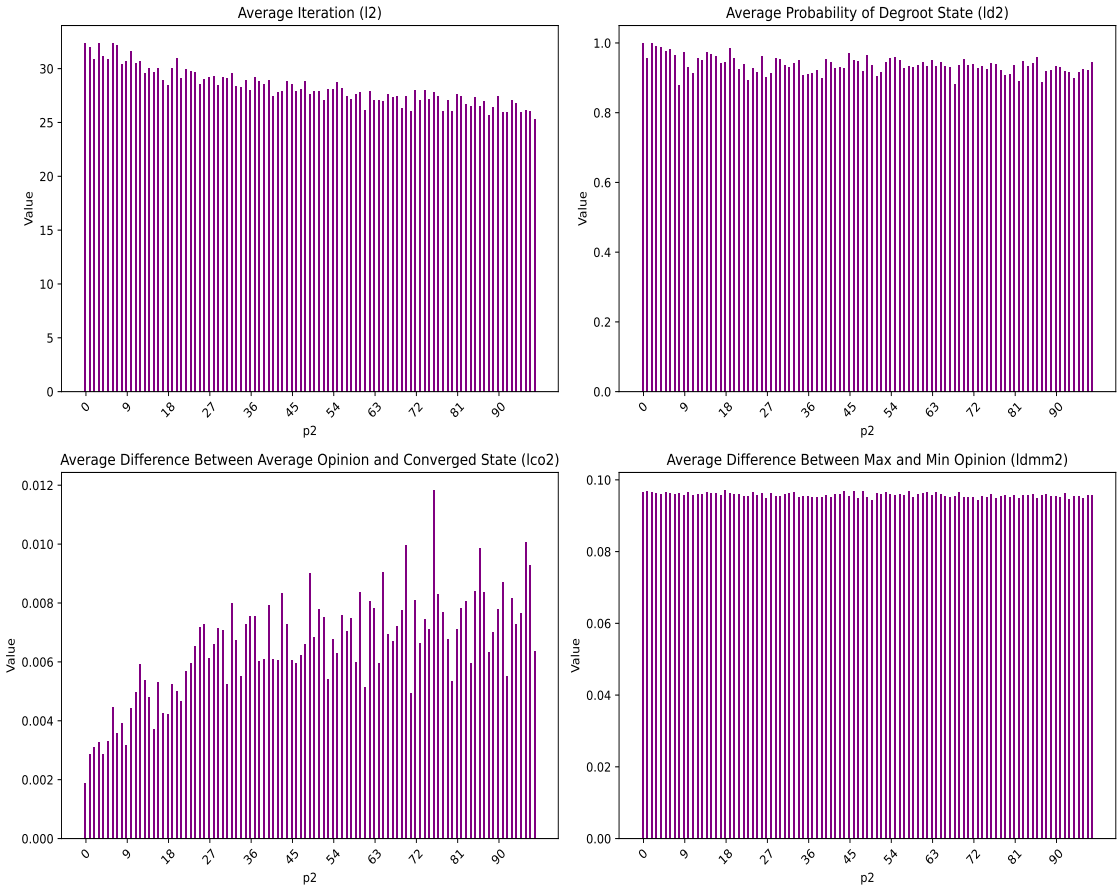


Fig. 4.5: A graph resulting from Experiment 5.

Conclusion

Throughout this thesis, I have analyzed the fundamental concepts and applications necessary to understand and implement the DeGroot opinion model within dynamic random graphs. The thesis began with an in-depth examination of Markov chains as the mathematical foundation for modeling stochastic processes. Then it explores opinion models, focusing on the DeGroot model to understand how opinions spread and reach consensus. Using Python and the NetworkX library, I simulated network structures and conducted over 100,000 computational experiments, which resulted with some interesting insights.

On a macro level, I was fascinated by the precision of theoretical mathematical methods compared to the results obtained from simulations. On a micro level, the exploration of how to move towards finding common ground has several implications for policy suggestions.

The results demonstrate that the more we listen to others, the easier it becomes to reach a common ground, even though consensus has not always been achieved. Additionally, simulations showed that exposure to individuals outside of one's echo chamber increases the likelihood of reaching common ground for the entire society.

Mathematically speaking, if each person establishes at least one meaningful connection outside their echo chamber, the society as a whole can stabilize its political climate.

Bibliography

[1] W. K. Ching and M. K. Ng, *Markov chains*, Models, algorithms and applications **650** (2006).

[2] K. L. Chung, *Markov chains*, Springer-Verlag, New York (1967).

[3] M. H. DeGroot, *Reaching a consensus*, vol. 69, Taylor & Francis, 1974.

[4] P. Erdos and A. Rényi, *On the evolution of random graphs*, Publ. math. inst. hung. acad. sci **5** (1960), no. 1, 17–60.

[5] S. N. Ethier, *The doctrine of chances: Probabilistic aspects of gambling*, Probability and Its Applications, Springer, Berlin and Heidelberg, 2010.

[6] M. O. Jackson, *Social and economic networks*, vol. 3, Princeton university press Princeton, 2008.

[7] D. A. Levin and Y. Peres, *Markov chains and mixing times*, vol. 107, American Mathematical Soc., 2017.

[8] F. Menczer, S. Fortunato, and C. A. Davis, *A first course in network science*, Cambridge University Press, 2020.

[9] H. A. Mimun, *Notes and slides of the course "gambling: probability and decision"*, 2023.

[10] J. R. Norris, *Markov chains*, no. 2, Cambridge university press, 1998.

[11] S. M. Ross, *Introductory statistics*, Academic Press, 2017.