Diffusion Dynamics Under the T-LISA Models

Marko S. Suchy
draft - 4/5/25

A thesis presented in partial fulfillment of
the requirements for the degree of
Bachelor of Science

Department of Physics and Engineering
Washington and Lee University
Lexington, VA, USA
September 10, 2023

Abstract

We study innovation dynamics through social networks, including the possibility for rejection of innovation. We make a slight improvement to the LISA model [17] mean-field, such that is results in a steady state with exclusively Adopter and Luddites. We then implement this improved model on Erdos-Renyi random graphs using Monte-Carlo techniques. We study the match between mean field and Monte-Carlo based approaches. We then use regression analysis to understand how both local and global network structural features effect the extent of innovation adoption in the final state. We find that under T-LISA dynamics, adoption is most likley to spread through small dense network clusters.

Acknowledgments

First, I would like to thank my thesis advisor, Professor Irina Mazilu, for her continued support throughout my undergraduate education. From my first year of college, she has nurtured my academic curiosity in physics and complex systems, and encouraged me to learn and grow. Over the past year, she has offered incredible support in the content, orginization, and context to my thesis, as well as mentorship in other facets of my life. Her and I are the 'we' referred to as the principal researchers throughout this thesis.

Second, I'd like to thank Professor Jon Eastwood, who inspires me to approach data science in an interdisciplinary way. In 2022, he introduced me to the fascinating world of social networks, which ultimately has resulted in this project. I have borrowed his Mark Newman 'Netowrks' textbook since early 2023.

Last, I'd like to thank the entire Washington and Lee Physics Department. The staff and faculty of this department make up a community which has forever changed me for the better. I am excited to carry the lessons I have learned from this community into the world at large, thinking critically about the world around me and approaching problems from first principals.

Contents

Abstract iii

Acknowledgments v

1 Background and Mean-Field Theory 1
- 1.1 Diffusion Models . . . 1
- 1.2 The LISA Model . . . 2
  - 1.2.1 LISA Model Mechanics . . . 3
  - 1.2.2 Mean-Field Theory Approach . . . 3
  - 1.2.3 A Note on Agent Based Simulation . . . 4
  - 1.2.4 Limitations of the LISA Model . . . 5
- 1.3 The T-LISA Model . . . 5
  - 1.3.1 T-LISA Model Mechanics . . . 6
  - 1.3.2 Mean Field Theory Results . . . 6
  - 1.3.3 Limitations of the Mean Field . . . 9

2 Graph Based Simulation Methods 11
- 2.1 Graph (Network) Theory . . . 11
  - 2.1.1 Node and Network Level Metrics . . . 12
  - 2.1.2 Erdos-Renyi Graphs . . . 12
- 2.2 Monte-Carlo Methods for Network-Based Simulation . . . 13
  - 2.2.1 Agent Based Rules . . . 13
  - 2.2.2 Computational Implementation of Simulations . . . 14
  - 2.2.3 A Normalization Issue . . . 14
- 2.3 Rescaling The Mean Field Theory to fit ER Graphs . . . 15

3 Analysis of T-LISA model on ER-Graphs 17
- 3.1 Results of re-scaled MFT vs. ER Simulation . . . 17
  - 3.1.1 Quantifying Error . . . 18
  - 3.1.2 Error Across Variables . . . 19
  - 3.1.3 What Does "Error" Even Mean? . . . 21
- 3.2 Effects of Local Environment on Innovation Adoption Under the T-LISA Model . . . 22
  - 3.2.1 Non-Noramlized Regression Results . . . 22

3.2.2 Normalized Regression Results 24
3.2.3 Local Structure Predictive Power 25
3.3 Effects of Global Environment on Innovation Adoption Under the T-LISA Model . . 25

4 Conclusion, Discussion, and Future Plans **29**

Bibliography **31**

A Python Tools **33**
A.1 `odeint` 33

B Python Code for Running Simulations **35**
B.1 Erdos-Renyi Random Graph Simulation Class 35
B.2 Simulation Class for Rescaled MFT 41
B.3 Parameter Sweep 43

Chapter 1

Background and Mean-Field Theory

1.1 Diffusion Models

Diffusion is occurring constantly, all around us. Think: changing room temperatures, dropping a tea-bag into a cup of warm water, Brownian motion, or the Democratic "Blue Wave" of the 2018 US election cycle. In each case, something spread continuously throughout a system. Across biology [24], physics [16], economics [11], sociology [20], and beyond, diffusion processes drive the change in dynamical systems. In this thesis, we concern ourselves specifically with diffusion of novel goods, ideas, and behaviors, through social systems. As E.M Rogers put it 'When new ideas are invented, diffused, and are adopted or rejected, leading to certain consequences, social change occurs' [21].

For a long time, researchers have tried to model diffusive systems through simplified mathematical rules. The mathematical framework typically applied to innovation diffusion is largely similar to that of the classic epidemiological SIR model: a set of coupled differential equations which track the proportions of the system in certain states. Often, in social diffusion models, these states are 'adopted' and 'not adopted.' Theoretically, innovation, new products, technologies, services, ideas, or behaviors, spread throughout society in the same way as disease does: through interaction with neighbors[1].

Among the early diffusion models was the Bass Model (BM), proposed by Frank Bass in 1969, which set out to predict the sales peak of novel durable consumer goods [1]. Ultimately, the model predicted sales as a function of time based on the proportion of the population who have already purchased the novel good. The Model works from the fundamental assumption that timing of consumers' initial purchases is influenced by the number of previous buyers, and utilizes a mathematical framework similar to that of contagion models. [1]. In his model, Bass accounted for *innovators*, who decide to adopt a new product regardless of the behavior of others, and *imitators*, who are pressured to adopt at a rate proportional to the number of previous adopters. Bass was able to use regression to tune the model parameters, and match his model to historical durable goods data quite well. Bass also had limited success using his model to predict long term behavior of a system given limited data.

In this thesis, we will first explore a generalization of Bass's model called the LISA model,

[1] 'Neighbors' in this context refers to agents with a direct connection physically, socially, economically, or otherwise, with the agent in question.

in the mean-field theory. We will then introduce the novel T-LISA model, a slight modification to the LISA model which offers more realistic steady states. We compare the T-LISA model with the original LISA model using mean-field theory and simulations. Finally, we simulate the T-LISA model on fabricated well-mixed social networks, Erdos Renyi graphs. Such simulations may help us understand how an agent's position in a network may effect how they react to innovation.

1.2 The LISA Model

In 2015, Mellor et al. [17] introduced a generalization of the Bass model, known as the LISA model which included a new class of agents: *Luddites*. Agents in the Luddite state permanently reject innovation. Thus the LISA model results in stable states with a proportion of the population having adopted innovation, and a proportion of the population having rejected it. The model does not allow for the possibility of Luddites reversing their rejection of innovation, or for *Adopters* to change their state. In other words, the final (stable) state produced by the model includes a proportion of the population which has adopted a novel idea, behavior, or good, and a proportion of the population which has rejected it. The flow of states which an agent may undergo can be seen in Figure 1.1.

The dichotomy of adoption and rejection may represent the diffusion of many types of innovation throughout society, especially those innovations which spur a counter-movement. Twitter is one location in whch the Adopter-Luddite dichotomy can be readily observed.

Consider, for example, the 'Anti-Vax' movement, which has ballooned in response to the recent COVID-19 pandemic. Durmaz and Hengirmen [7] explored the dynamics of the Anti-Vax movement in Turkey following the onset of Covid-19. The team collected Twitter data from hashtags that were for vaccines, against vaccines, and neutral, and then analyzed the data via social network analysis (SNA.) They found that after the onset of the pandemic there was a drastic increase in the number of accounts tweeting with exclusively pro or anti vaccine sentiment (like Adopters and Luddites might.)

Benoit and Mauldin [2] explore the use of media platforms as a method for anti-vaxers to disseminate ideas, without supervision, to parents. In the *Journal of Sociology*, Francis Russel suggests that performative nature anti-politics in the anti-vax movement has given rise a increase in visible protest [22].[2] Clearly, to perform, one must be motivated by their peers. Performative anti-politics is an incredibly interesting phenomena that could be modeled well by an innovation-diffusion model which incorporates rejection.

Other examples of the Adopter-Luddite dichotomy exist throughout society, historically and at present. People have/had strong views on political topics like Donald Trump, abortion, or Lyndon B. Johnson's war in Vietnam; behaviors such as veganism/vegetarianism, entering the Apple Ecosystem (iPhone, mac, etc.,) or polyamory; and products like electric vehicles (plus other green-energy products), rock and roll music, cryptocurrencies, and alcohol. All of these innovations were at one time novel, or came into the spotlight in society in a way which led to a shift in the Adopter-Luddite dichotomy. The LISA model offers a start in understanding the general dynamics which cause this dichotomy.

[2] Increasing and rapid adoption of masking, social distancing, and vaccination has led to a large and visible proportion of the population who actively reject such behaviors and products. This is exactly what the LISA model predicts!

1.2.1 LISA Model Mechanics

In the LISA model, a small proportion of the population is initialized as *Susceptible*, while the rest of the population is initially *Ignorant*. The Susceptible class is akin to Frank Bass's *innovators*, who adopt a new product regardless of the behavior of others at a constant rate. Susceptibles become Adopters at a constant rate γ Ignorants are like Bass's *Imitators*, who may become Succeptile based on their interaction with other Susceptibles in the system. Ignorants also may become Luddites, which do not have an analog in the Bass Model. The possible flow of states in the model can be seen in Figure 1.1.

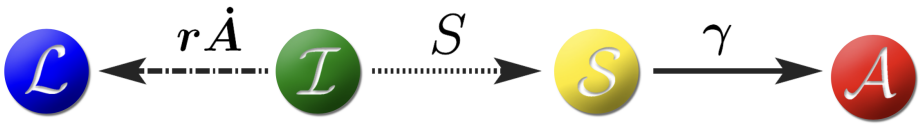

Figure 1.1: Possible changes in an an agent's state, from Mellor et. al's LISA model [17]. Note that an Ignorant's propensity to become Luddite is directly proportional to the increase in agents in the Adopter state.

Like the Bass Model, the LISA model works from the theoretical basis that an increasing number of Susceptibles across the system (which must eventually turn into Adopters) encourages those ignorant of novel innovation to adopt. the inclusion of the intermediary step between Ignorant and Adopter allows for a similar dependence on system changes in Luddite proportion. The change in the systems Luddite proportion is directly proportional to the rate of adoption in the system. The following set of coupled differential equations governs the behavior of the system:

$$L' = rA'I = r\gamma SI \tag{1.1}$$
$$I' = -(1 + \gamma r)SI \tag{1.2}$$
$$S' = S(I - \gamma) \tag{1.3}$$
$$A' = \gamma S \tag{1.4}$$

where L, I, S, and A are the proportions of the population in their respective state, and r is the luddism parameter (essentially, how prevalent luddism is in this system).

Equation 1.1 shows that when the rate of increase in adopters is high, the rate at which luddism spreads increases. In other words, there is an interaction between Ignorants and the rate at which the Adopter proportion of the system increases, which leads to an increase in the number of Luddites. Equation 1.3 shows an interaction between Suceptibles and Ignorants, which increases the number of Susceptibles. In other words, Susceptibles have become aware of some innovation, and they are talking about it amongst their Ignorant friends, who in turn either reject that innovation or become Susceptible to it themselves.

1.2.2 Mean-Field Theory Approach

In the Mean-Field Theory (MFT) case, described above by coupled differential equations 1.1 through 1.4, we assume a well mixed system in which every agent has some effect on every other

agent. In real social networks, this isn't always the case. Agent's often lie in the middle of groups, putting them in a unique position to effect innovation diffusion. While the MFT-model does not account for these intricacies, it offers a good approximation of a decentralized social system. For instance: one community where all members are roughly equally close, are all connected with one another, and are not influenced by the outside.

Beginning with the MFT case allows for an easily implementable model, where we can explore the average dynamics arising from the rules of the model. Differential equations 1.1 through 1.4 can easily be implemented in Python, and result in graphs that look like those in Figure 1.2.

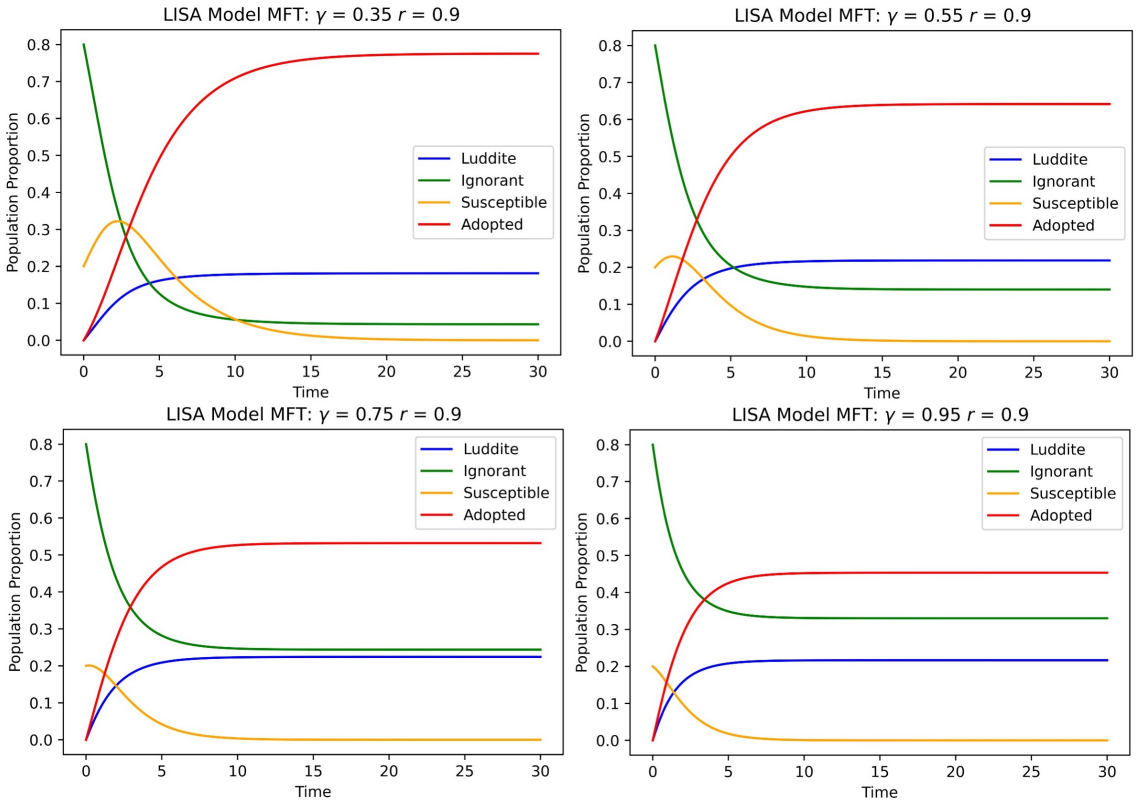

Figure 1.2: Mean Field Theory Plots of LISA model for various model parameters. Note, in all runs of the model seen here $I_0 = 0.8$. Thus, the only run seen here in the 'rapid but sparse' adoption scheme is the bottom right, however the bottom left is quite close.

As presented in Mellor et al., the LISA model gives rise to two major schemes of adoption: slow but wide spread, when γ (the rate of adoption) is low; and rapid but sparse adoption when γ is high [17]. The specific threshold for rapid but sparse adoption, defined by Melor et. al in the mean field theory section if their paper, is $\gamma > I_0$ (initial Ignorant proportion.)

1.2.3 A Note on Agent Based Simulation

Discussion of the LISA model would be incomplete without mentioning the model's agent-based simulated results. In re-creation of the LISA model, we performed stochastic simulation on networks, as did Melor et al.. A full discussion of network simulation will be saved for chapters 2

and 3. For now, it is important to know that one cannot directly apply differential equations 1.1 through 1.4 to an agent based network. Rather, a new set of equations must be developed which describes the propensities[3] of state change based on a an agent's current state, and the states of the agent's neighbors. Those propensities, directly as Melor et al., describe them, are as follows, where N represents the total number of nodes in the system, s_i represents the Susceptible neighbors of node i, and k_i is the number of neighbors of node i.

$$I \to S : \frac{s_i}{N} \tag{1.5}$$

$$I \to L : r\gamma\frac{s_i}{k_i} \tag{1.6}$$

$$S \to A : \gamma \tag{1.7}$$

The resulting curves from stochastic implementation of these propensities are qualitatively similar to those from equations 1.1 through 1.4. In fact, the model parameter γ, along with the threshold between the two adoption schemes, are systematically rescaled as shown by Melor et. Al [17]. In chapter 2, we will go through a similar rescaling for our alteration to the LISA model (which we call the T-LISA model.) For now, it enough to know that we have succesfully re-created both the MFT and agent-based results of Melor et al., such that we may begin to start building off of them.

1.2.4 Limitations of the LISA Model

As seen in Fig. 1.2, the original LISA model leaves several Ignorants in the steady state. This is because the change in Ignorants (I') is dependent only on the interaction between Susceptibles and Ignorants, as seen in equation 1.2. When the growth of Adopters outpaces the growth of Susceptibles for some time, no more Susceptibles will exist in the system. Therefore, there can be no interaction between Susceptibles and Ignorants, stopping any growth in Susceptibles while leaving a sometimes significant population proportion of Ignorants.

This sort of stalling of innovation diffusion is one characteristic of the LISA model which is difficult to reconcile with intuition for how real systems work. Especially in a well mixed system, as our mean-field representation assumes, it does not make sense that Ignorants neighboring Adopters would remain ignorant as time goes on. Intuitively, especially in a well mixed system, there must also be some interaction between Adopters and Ignorants, similar to that between Susceptibles and Ignorants. This is the problem with the LISA model which we attempt to address in our introduction of the novel T-LISA model.

1.3 The T-LISA Model

The Terminal-LISA, or T-LISA, model is set up in such a way that the system always reaches a steady state with the entire population having adopted or rejected innovation. Thus the model depicts a stringent Adopter-Luddite dichotomy. In many situations such a stringent dichotomy is implicit. For instance, parents either vaccinate their children, or they don't, and voters participating

[3]Note, these are not probabilities, as they do not necessarily sum to one. When implemented they are normalized using the random.choices() function in Python.

in a 2 party system either vote one way or another. Sure, some agents may not have strong feelings but in these examples they are still forced to make a choice (even if they choose to remain at default) and they are influenced by the dynamics of the system.

1.3.1 T-LISA Model Mechanics

The T-LISA model achieves a stringent Adopter-Luddite dichotomy in the steady state by the inclusion of new interaction terms between Adopters and Ignorants, such that even when no Susceptibles are present in the system, Ignorants may still change states. The following set of coupled differential equations governs the MFT of the T-LISA model, with new interaction terms between Adopters and Ignorants in all equations except for 1.11:

$$L' = \gamma r SI + \omega r AI \tag{1.8}$$

$$I' = -(S + A)I - \gamma r SI - \omega r AI \tag{1.9}$$

$$S' = (S + A)I - \gamma S \tag{1.10}$$

$$A' = \gamma S \tag{1.11}$$

Just like the original model, γ is the rate at which Susceptibles turn to Adopters (implicitly effecting the the growth rate of Luddites) and r is a parameter that represents how prevalent luddism is in this particular instance of innovation diffusion. The $\gamma r SI$ term in Equation 1.8 is equivalent to rIA', meaning the interaction between Ignorants and the increase in Adopters leads to greater luddism throughout the system.

We introduce the new parameter ω which captures the interaction between Adopters and Ignorants resulting in luddism. Its interpretation is similar to γ, however not a direct analog as the second term in equation 1.8, $\omega r AI$ lacks A' term. It is interpreted more like $(rI)(\omega A)$ where ωA represents the influence of current Adopters on Ignorants. A positive ω represents an anti-establishment tendency of Ignorants, who tend to reject what a large proportion of neighbors have adopted. The effect of individual 'hipsters' has been studied previously by Juul and Porter [15], however in our model 'hipster' effects, brought on by ω, are distributed throughout all nodes. A negative ω represents a bandwagon tendency in Ignorants, or tendency to follow the proportion of Adopters. The inclusion of bandwagon tendency (negative ω) allows for new type of behavior in the mean field: a negative change in the population proportion of Luddites. Unfortunately, when negative ω is included in the MFT, some proportions of the population become negative, and others greater than one. Therefore we will examine only positive values for ω in the MFT case.

The T-LISA model also includes a new interaction terms between Ignorants and Adopters resulting is an increase in Susceptibles. These $\pm AI$ terms are included in equations 1.9 and 1.10, and like the interaction terms between Ignorants ans Susceptibles, are not governed by any particular parameters. Thus, the Adopter proportion of the population has the same pro-innovation effect on system as the Susceptible proportion. These parameters ensure diffusion through the system never stalls, and represent a more accurate social scenario where Adopters influence Ignorants.

1.3.2 Mean Field Theory Results

One frustrating result of T-LISA MFT differential equations is that they cannot be analytically solved due to the added degree of freedom, ω. Therefore, it is difficult to predict the time at

which certain shifts, like the peak in Susceptibles, will occur, as Melor et al. do in their exploration of the LISA model. Because it is impossible for the T-LISA model, discussion of the analytical solution to the LISA model has been left out of this thesis. This represents a movement away from the original methods of the Bass Model, as Bass was interested in the timing of adoption of durable consumer goods. We are begining to enter a new territory, where MFT results may be less tractable than simulated results. That being said, we will still examine the MFT results of the T-LISA model to develop an intuition for how it behaves, and how it is different than the LISA model.

We begin by comparing the shapes of graphs generated by the LISA and T-LISA models. Figure 1.3 shows 8 runs of the T-LISA model at different values for γ and ω, with LISA model results for the same γ overlaid. Generally, we find that the shape of our graph remains similar to that of before, but with steeper gradients. This is expected considering the fact that the T-LISA model adds positive terms to differential equations corresponding with population proportions which increase, and negative terms to those which decrease. Over all, there is greater influence exerted on Ignorants, who are more quickly filtered into Susceptibles or Luddites.

As a rule, when $\omega = 0$ the T-LISA model produces a stable state with a lower proportion of Luddites in the final state (L_{tMax}) than the LISA model. This is because Ignorants feel greater pressure to become Susceptible from Adopters, but there is no extra to become Luddite from Adopters. Alternatively, when $\omega = \gamma$ the Luddite proportion produced by the T-LISA model is always greater than or equal to that of the LISA model. It appears that the two modes of adoption still hold: a high value for γ leads to a fast adoption with widespread luddism, and a low value for γ leads to slower adoption with less Luddites in the final state. However variance in ω can clearly perturb this.

To examine the resulting Luddite proportion over all values of γ and ω between zero and one, we introduce a phase plot, as seen in Figure 1.4. The phase plot upholds our intuitions about case when $\omega = 0$ and when $\omega = \gamma$. The plot also shows us that the possible significance of ω is dependent on the adoption rate γ. When γ is high, the number of Adopters in the system prior to the steady state (while Ignorants remain) becomes relatively higher, meaning ω can have a greater impact on L_{tMax}. Furthermore, Figure 1.4 shows us that, while holding γ constant, the T-LISA model can result in an L_{tMax} both above and below what would be expected for the LISA model. There also always must exist some combination of γ and ω for the T-LISA model such that L_{tMax} for the two models match.

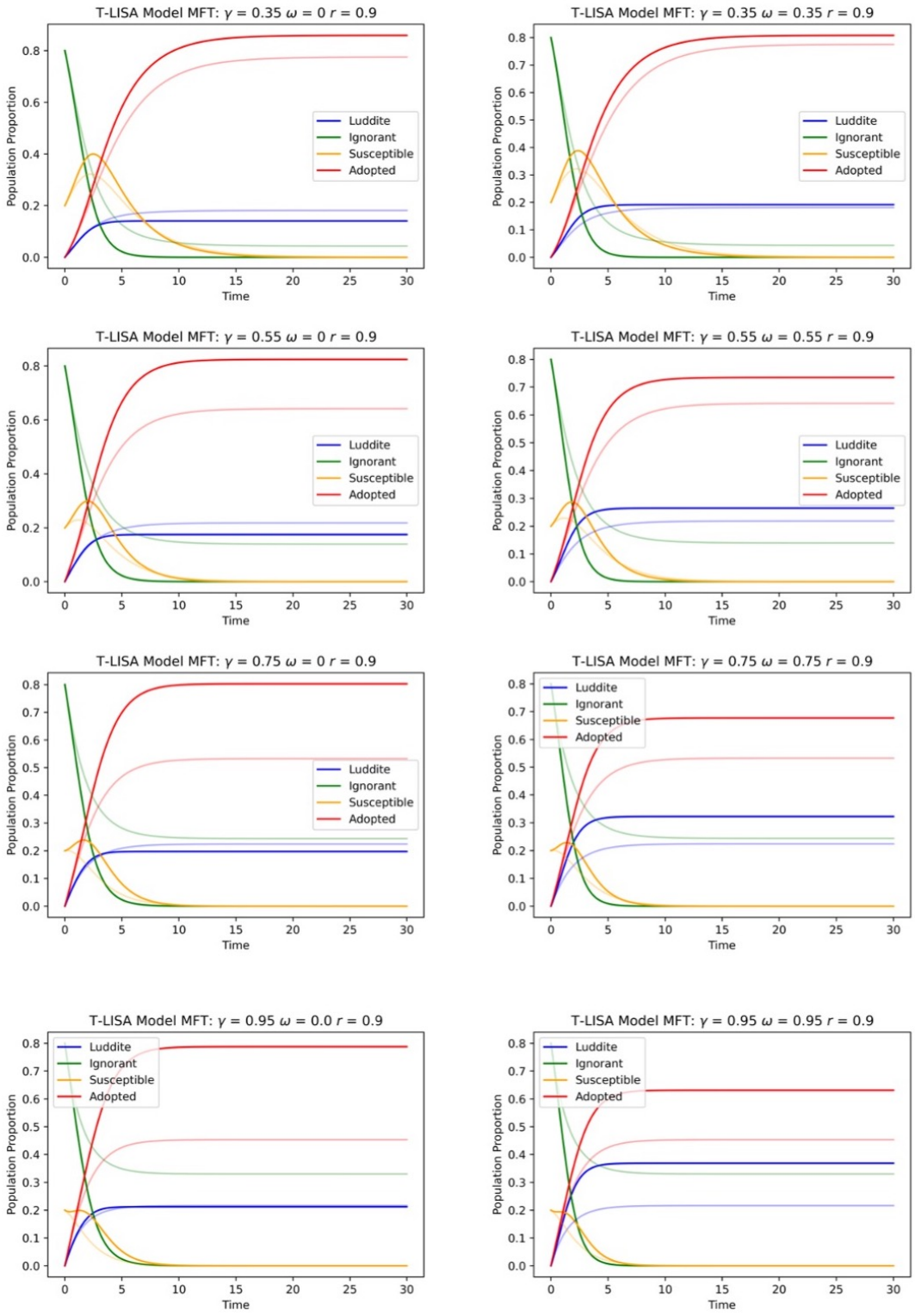

Figure 1.3: MFT Plots for the T-LISA model with $\omega = 0$ (left) and $\omega = \gamma$ (right.) All plots are using $r = 0.9$ and $S_0 = 0.2$ LISA model results based on the listed γ and r values are overlayed on each graph with less opacity.

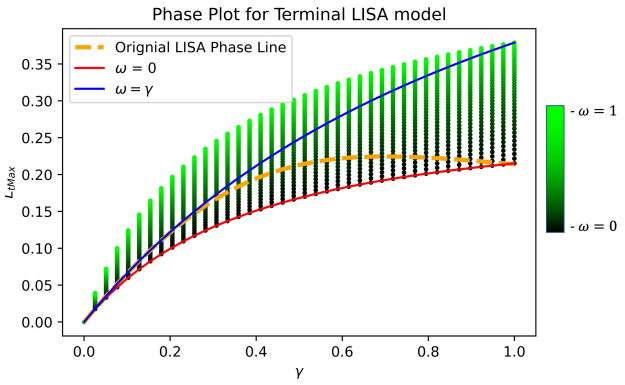

Figure 1.4: Phase diagram for the Terminal LISA model with $r = 0.9, S_0 = 0.2$.. The range of black and green dots coming off of the $\omega = 0$ line represent the possible results for L_{tMax} when introducing an anti-establishment pressure parameter ranging from 0 to 1.

1.3.3 Limitations of the Mean Field

Frank Bass analytically solved his model to estimate the time at which consumers would purchase novel consumer goods. As we have already discussed, such an analytical solution is not possible for the T-LISA model. So, what is the T-LISA model good for? It is hard to say what the MFT of the model is helpful for in commercial applications. With more paramters than both the Bass and LISA model, fitting the T-LISA model to real world data seems a Herculean task.

The MFT gives us an understanding of what happens in a well-mixed population, but our society is far more complex than that. How can we understand the Adopter-Luddite dichotomy in the context of real world social networks? To do so, we must adapt our model's rules to the context of a network, and then we can simulate innovation diffusion in a way that might get us closer to understanding the factors that lead to someone's accepting or rejecting of novel goods, ideas, behaviors, and so on.

Chapter 2

Graph Based Simulation Methods

2.1 Graph (Network) Theory

Networks, also known as graphs, offer a robust way to visualize, interpret, and analyze relational datasets. As put by Kleinberg and Easley, networks are 'a pattern of interconnections among a set of things' [8]. All networks contain a set of *nodes*, which can be thought of points in space, agents in a system, or generally 'things.' Networks also contain a set of *edges*, which represent relations between nodes. By introducing a number of node and edge level attributes, many datasets can be viewed through the lens of networks. The study of networks reaches far across disciplines. In fact, due to the nature of the network science, multiple attempts have been made to codify the study of networks into a common language, and mathematical framework [18], [4].

Typically, networks are represented mathematically through adjacency matrices. Suppose a graph G, with 5 nodes, as seen in Figure 2.1. The corresponding adjacency matrix A would be a 5x5 matrix, with columns and rows 1 through 5 representing each node. Thus, the matrix element A_{ij} represents the relationship between nodes i and j, where 1 represents the existence of an edge and 0 represents the absence of an edge. If we assume that nodes do not have a relationship with themselves, we know that the diagonal of this matrix will consist of 0s. If relationships are non-directional in our network, we also know the adjacency matrix is mirrored across the diagonal.

When simulating the T-LISA model, our nodes represent agents in our system and contain the characteristic 'state', which can be set to Luddite, Ignorant, Susceptible, or Adopted. The edges in our network represent general relationships between these agents, such a friendship, kinship,

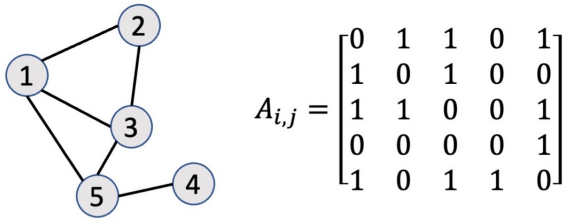

Figure 2.1: A simple example of a labeled, undirected network with 5 nodes, and it's adjacency matrix.

business relations, etc.. Our model is set up so that the probabilities of change in a node's state are dependent on the distribution of the states of that node's neighbors. (The set of a node's neighbors are all other nodes connected to that node through 1 edge.)

The T-LISA model is meant to simulate the diffusion of innovation throughout a pre-existing network. We will start with special classes of Networks known as Erdos-Renyi (ER) random graphs, which fit the assumptions of our mean-field well, as described in section 2.3. In future, we hope to use previously connected network data to examine how a novel idea or good might spread through networks such as academic publishing networks, or social media networks.

When conducting graph-based simulation in Python, the NetworkX package is used. This packages stores Adjacency matrices for graphs, along with node attributes, and offers several methods for network analysis. The use of NetwotkX can be seen in Appendix B.1.

2.1.1 Node and Network Level Metrics

The effects of network position are of great importance in our lives. A node's position in a network is closely related to power that node has to influence the entire system. For instance, in a hub and spoke model the hub controls the flow of all information from one spoke to the rest. Nodes also may exist that create connections between densely connected network clusters, which may otherwise be isolated. These nodes are in the unique position to diffuse information throughout a system, or to stop the flow of information. This sort of phenomena has long been studied in the context of sociology, and can be referred to as 'social capital' [10], [5].

Basic node-level metrics exist which help quantify social capital. For instance, the degree of a node (it's number of neighbors) many forms of centrality ('The extent to which the ties of a given network are concentrated on a single actor or group of actors' [12],) and local clustering coefficient (a measure of how interconnected a node's neighbors are.) There are also network level metrics, which offer a sort of aggregate picture of how nodes are connected. Such metrics include network size, degree distribution, average path length, and more. Later in this thesis, we will ask the question: *is there a relationship between node or network level metrics and the final distribution of Luddism and Adoption throughout a system?*

2.1.2 Erdos-Renyi Graphs

We begin with Erdos-Renyi random graphs. As described by Paul Erdos and Alfred Renyi, we define a graph $G(N, P)$, where N is the number of nodes, and P is the probability of an edge existing between two arbitrary nodes [9]. The graph is initialized such that every possible edge between nodes is generated (or not generated) with probability P. Thus, the ER framework for random graphs results in a well mixed graph (although not perfectly mixed) which decently approximates our MFT condition. Using ER graphs in our graph-based investigation allows for comparison to our MFT results.

Note that the parameter P can also be expressed in terms of the average degree on nodes in our ER network, which we will call k. $P = \frac{k}{N-1}$, where N is the total number of nodes in the network. Generally, when constructing our ER graphs we think in terms of k, and convert to P, as k gives a more tangible metric for the interconnectedness of our system. Parameter k will also become important in section 2.3, where we rescale MFT for better comparison with Monte-Carlo simlulated results.

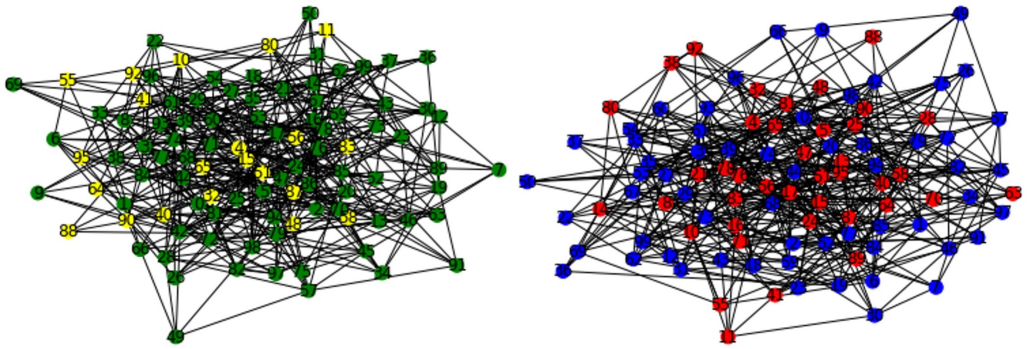

Figure 2.2: Two ER random graphs, both with $N = 100$, $k = 10$. The Left graph is in an initial state of the T-LISA model, where green nodes are Ignorant and yellow are Susceptible. The Right graph is a stable state of the T-LISA model, in which red nodes are Adopters and blue nodes are Luddites.

Computationally, Erdos-Renyi graphs are generated using the python *nx.erdos_renyi_graph()* function, as seen in Appendix section B.1.

2.2 Monte-Carlo Methods for Network-Based Simulation

When simulating the T-LISA model on networks we no longer can rely on the coupled differential equations 1.8 through 1.11, as we are no longer in a continuous space. In our graph-based simulation we maintain discrete states, meaning nodes cannot be partially in 2 states, as our MFT results imply. We assume that each node in the network is influenced primarily by it's neighbors. Based on who a node is in contact with, probabilities for state change vary at the individual level. In a single system one Ignorant node may be surrounded by several Adopters and Susceptibles which exert a strong pressure on that node to change state, while another Ignorant node may be surrounded by only Ignorants which exert no pressure on that node to change. Thus complex dynamics arise for which Monte-Carlo simulation methods are appropriate.

2.2.1 Agent Based Rules

The rules of the T-LISA model must be transformed to the level of the individual to support Monte-Carlo simulation. To do so, we follow Mellor et al., who use the Gillespie algorithm to make the transformations [17].[1] We replace interaction terms leading to luddism with the fraction of Susceptibles and Adopters out of a node's neighbors. Thus, luddism is based on the local rates of susceptibility and adoption in our Monte-Carlo simulation. We replace interactions leading to susceptibility with the fraction of susceptible and adopted neighbors per the total number of nodes in the system. Thus, increases in susceptibility are based on an agent's perception of global adoption rates. The logistic nature of growth behavior in susceptibility as predicted by the Bass Model is preserved. The propensities of state change are seen in 2.1, 2.2, and 2.3.

[1] Note, the Monte-Carlo methods used in our simulation are not consistent with the Gillespie algorithm. However, our Monte-Carlo methods do produce results which are fairly consistent with our Mean-Field results. More on this issue in section 2.2.3.

We let k_i be the degree of arbitrary node i in the system, and let s_i and a_i be the number of node i's Susceptible and Accepted neighbors respectively. N represents the total number of nodes within the system. The meanings of parameters r, γ, and ω, are preserved.

$$I \to L : r(\gamma \frac{s_i}{k_i} + \omega \frac{a_i}{k_i}) \tag{2.1}$$

$$I \to S : \frac{s_i + a_i}{N} \tag{2.2}$$

$$S \to A : \gamma \tag{2.3}$$

2.2.2 Computational Implementation of Simulations

We use a class-based approach to applying our model in Python. As seen in Appendix Section B.1, we create an *ER_Graph* class which takes model parameters r, γ, and ω when initialized, along with hyperparameters N, k, *simulations* (maximum simulation steps), and *initial_conditions*. When initialized, an *ER_Graph* object creates a Erdos Renyi graph with N nodes, and a randomly assigned node states as given (probabilistically) by the *initial_conditions* vector. For our investigation, we use only initial conditions which contain on average 20% Susceptibles, and 80% Ignorants.

We iterate over each node in our network in the *simulation_step* method of the class, and update nodes sequentially based on propensities 2.1, 2.2, and 2.3, using the *random.choices()* Python function. The *run_one_simulation* method calls the *simulation_step* method several times in a row[2] and saves the resulting population fractions into solutions lists. Finally, the *run_n_simulations* method runs n simulations with the same parameters on different ER graphs initialized with the same hyperparameters. The method also exports the solutions data from each simulation run, along with metadata for the parameters, hyperparameters, and simulation batch number (a unique id for each stochastic simulation such that every specific simulation run can be identified).

2.2.3 A Normalization Issue

A known normalization issues exists in our computational implementation of equations 2.1, 2.2, and 2.3. When using the *random.choices()* function to determine the state change for Ignorant nodes, we assume there is some probability that nothing happens: $I \to I$. We calculate this probability of no state change assuming normality: $I \to I = 1 - (r(\gamma \frac{s_i}{k_i} + \omega \frac{a_i}{k_i}) + \frac{s_i + a_i}{N})$, or $I \to I = 1 - (I \to L) - (I \to S)$. Such a normalization condition implies that $(I \to L) + (I \to S) < 1$ (so as not to yield a negative probability) however in some edge cases this condition is not satisfied.

Consider the case with node i in a maximally connected triad, as seen in Figure 2.3. By the propensities 2.1, 2.2 and 2.3, we find that in such a case the model breaks down if the product $r\gamma$ is greater than $\frac{2}{3}$, leading to a negative value for the implicit probability $I \to I$.

To generalize, solving for $(I \to L) + (I \to S) \leq 1$ with general s_i, a_i, and k_i, we find condition 2.4. For node i, propensity $I \to I$ will be non-negative only if the local environment meets this condition. Notably, when $I \to I$ is negative, it is treated by the *random.choices()* function as 0, and state change is selected based on the normalized probabilities 2.1 and 2.2 only. Thus when condition 2.4 is not met for node i, a state change must occur.

[2]Until a stable state is reached, or we have run the number of simulation steps as set out by the *simulations* hyperparameter.

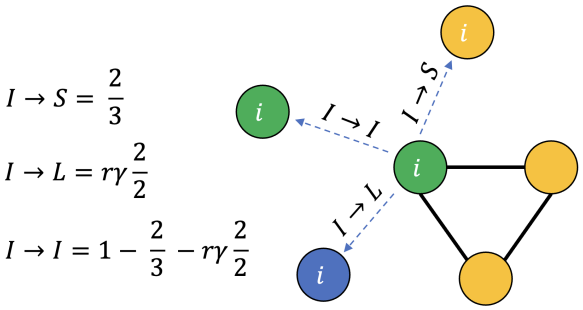

Figure 2.3: A maximally connected triad with 2 Susceptibles (yellow) and one Ignorant (green). Dotted blue lines represent the possible evolutions of node i at $t+1$, where blue represents Luddite, with probabilities given by propensities 2.1, 2.2 and 2.3. In this case the condition $r\gamma \leq \frac{1}{3}$ must be met for $I \to I$ to remain non-negative.

$$(\gamma s_i + \omega a_i)\frac{1}{k_i} + (s_i + a_i)\frac{1}{N} \leq 1 \tag{2.4}$$

Ultimately, this normalization issue arises because we do not use the Gillespie algorithm for simulation, which assumes that at every simulation step some change occurs (where simulation steps are stochastically time variant.) However, regardless of this normalization issue, we find solid evidence that our agent based model does a good job of recreating both qualitative and quantitative features of our MFT model, as discussed in section 3.1.

2.3 Rescaling The Mean Field Theory to fit ER Graphs

Our ER graph is an excellent first system on which to explore the effects of the T-LISA model. Because of the nature of the ER graph (randomly and highly interconnected) T-LISA may be effectively compared to the mean field theory (MFT) predictions. Our MFT population proportion curves are calculated under the assumption of a perfectly mixed population in which every agent impact each other. To adapt our MFT to our well-mixed ER graph, in which only connected nodes impact each other, we must re-scale two-body contagion processes by $\frac{k}{N}$ [17]. This scalar factor, average connections per total nodes in system, captures the proportion mixing in our system. When properly scaled, we should see matching qualitative behavior in system dynamics for T-LISA models ran via ER simulations and MFT equations.

By similar methods from the Melor et al. [17], we can find the equation for the re-scaled change in susceptibles. This will help us quantify how well MFT predictions and simulation results match.

To find the re-scaled form for S' we assume adjecency matrix element $A_{ij} \approx k_i k_j / Nk \approx k/N$, where k_i is the degree of node i, and k is the average node degree. With, S_i, I_i, and A_i are the probabilities that node i is in the susceptible, ignorant, or adopted states. We generalize:

$$S' = S_i[\sum_j (\frac{A_{ij}}{N} I_j) - \gamma] + A_i \sum_j (\frac{A_{ik}}{N} I_j) \tag{2.5}$$

which can be re-written as:

$$S' = S(\frac{k}{N}I - \gamma) + A\frac{k}{N}I \tag{2.6}$$

Applying our new definition for two-body contagion processes, we come up with a re-scaled set of MFT equations, which takes into account the proportion of average connections to total nodes, $\frac{k}{N}$:

$$L' = \gamma r S I + \omega r A I \tag{2.7}$$

$$I' = -\frac{k}{N}(S + A)I - \gamma r S I - \omega r A I \tag{2.8}$$

$$S' = \frac{k}{N}(S + A)I - \gamma S \tag{2.9}$$

$$A' = \gamma S \tag{2.10}$$

Chapter 3

Analysis of T-LISA model on ER-Graphs

3.1 Results of re-scaled MFT vs. ER Simulation

We begin by looking in to the alignment between our re-scaled MFT, and our Monte-Carlo simulation with ER graphs results. To do do, we discretely sweep across parameters N, r, ω, γ, and k, running batches of stochastic simulations for several parameter sets. The computational implementation of our method uses the *np.linspace()* function to set up parameter space, along with several *for* loops to sweep across it, as seen in Appendix B.3. The results for a few batches can be seen in Figure 3.1.

The initial results as seen in Figure 3.1 are promising. Clearly, at least in these cases, there is a strong qualitative match between the shapes of stochastic and deterministic simulation. The question of quantitative match remains unclear. Analytically, we cannot prove there should be a quantitative match, however it is plausible that with more simulations average values generated by Monte-Carlo simulations may converge onto MFT.

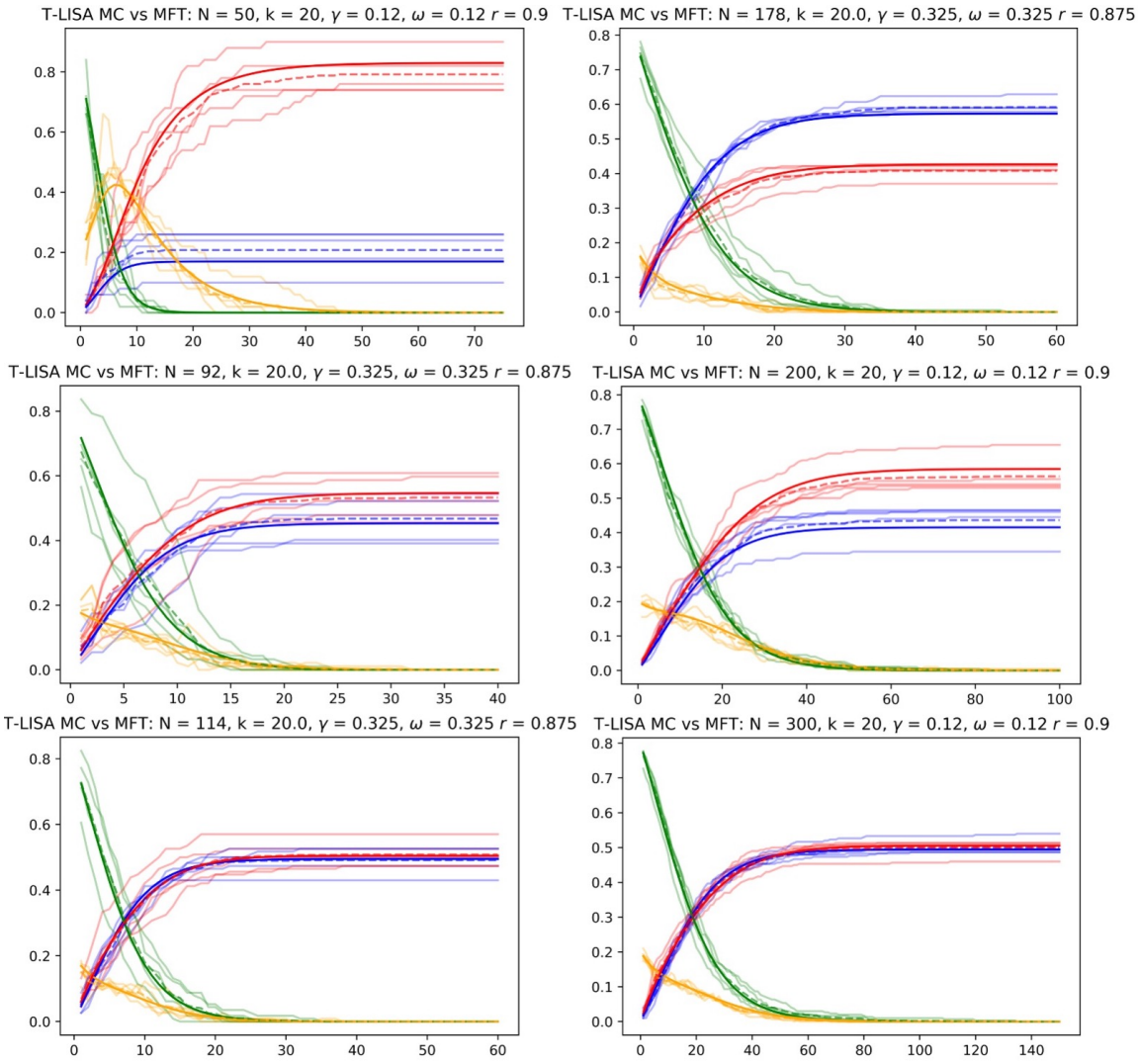

Figure 3.1: Monte-Carlo vs MFT results for various parameters. Monte-Carlo simulated results are overlayed with low opacity. The dotted lines represent the average value generated by Monte-Carlo simulation.

3.1.1 Quantifying Error

To examine how the model compares to MFT throughout a broader slice of parameter space we calculate error from MFT, then analyze error across parameter space. For calculating error, we consider the MFT curve as our baseline. We also utilize absolute values when calculating total error, such that our errors don't cancel when adding. The following algorithm is used to calculate error in our simulations:

1. Generate a comparison dataframe by joining solutions list for MFT results at integer time steps[1] with Monte-Carlo simulation results. Also, include a column for T, simulation step.

2. For L, I, S, and A solutions lists, compute the difference in population proportion for Monte-Carlo simulation and MFT simulation, at every time step. Store the difference in columns *L_difference*, *I_difference*, *S_difference*, and *A_difference*.

3. Calculate the mean of the difference columns for L, I, S, and A, and store it in an *output_csv* dataframe, along with all model parameters for the particular simulation at hand.

4. Calculate total average error by adding absolute values of mean L error, mean I error, mean S error, and mean I error. Store this error in the *output_csv* dataframe.

We can boil down the above algorithm into implementation of the following equations, where MC_sol and MFT_sol are the matrices for the Monte-Carlo and MFT solutions respectively, with columns for L, I, S, and A, and D is the difference matrix between them.

$$D = MC\_sol - MFT\_sol$$

Assuming D is an $m \times n$ matrix with elements d_{ij}, we find element n of the error vector (**error** has dimension $1 \times n$) by calculating:

$$error_n = \frac{1}{m} \sum_{i=1}^{m} d_{in}$$

Finally, we calculate the scalar "total average error" by finding the mean of the absolute values of the elements within the **error** vector. Thus our total average error scalar captures an aggregate measure of the total difference between our MFT and Monte-Carlo Simulation. Note that this differences is in units of population proportion, not percentage.

We theorize that our Monte-Carlo based model effectively matching MFT on ER graphs would indicate robust simulation methods. We are especially weary of large and sparsely connected graphs (high N, low k) as these are likely to trigger condition 2.4, where normalization breakdown occurs. We analyze error across a small section of parameter space in the following section.

3.1.2 Error Across Variables

Throughout our analysis, we are computationally limited to a fairly low maximum for N. The time of computation for each step of our simulation via *NetworkX* scales positively with N, such that by the time we reach $N = 300$ one simulation step takes multiple seconds to run. Considering the large combinatorial complexity for sweeping through a discrete part of parameter space[2] it is only feasible to explore parameter space with small N. Because of this computation limitation, we must make some difficult choices in limiting the discrete parameter space to sweep over. In most of our error analysis, we limit $r \in [0.5, 1]$, $k \in [10, 30]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$, and $N \in [50, 200]$.

[1] Note, to guarantee we have MFT results at every integer time step, we make sure to call the *odeint()* function on a linspace object which has the appropriate integers in it. This can be seen in Appendix B.2 under the definition of the *solve_MFT* method

[2] Per one step in N, on the order of thousands of full simulations must be run to capture some useful discrete parameter space. Combinatorial complexity arises from batch size, r space, k space, γ space, and ω space.

We begin to visualize error distribution across N in Figure 3.2. Note that in this figure parameter values for r, k, γ, and ω are not specified. We see error increases with decreasing graph size. This is expected considering, as larger systems are typically are less impacted by stochastic variations.

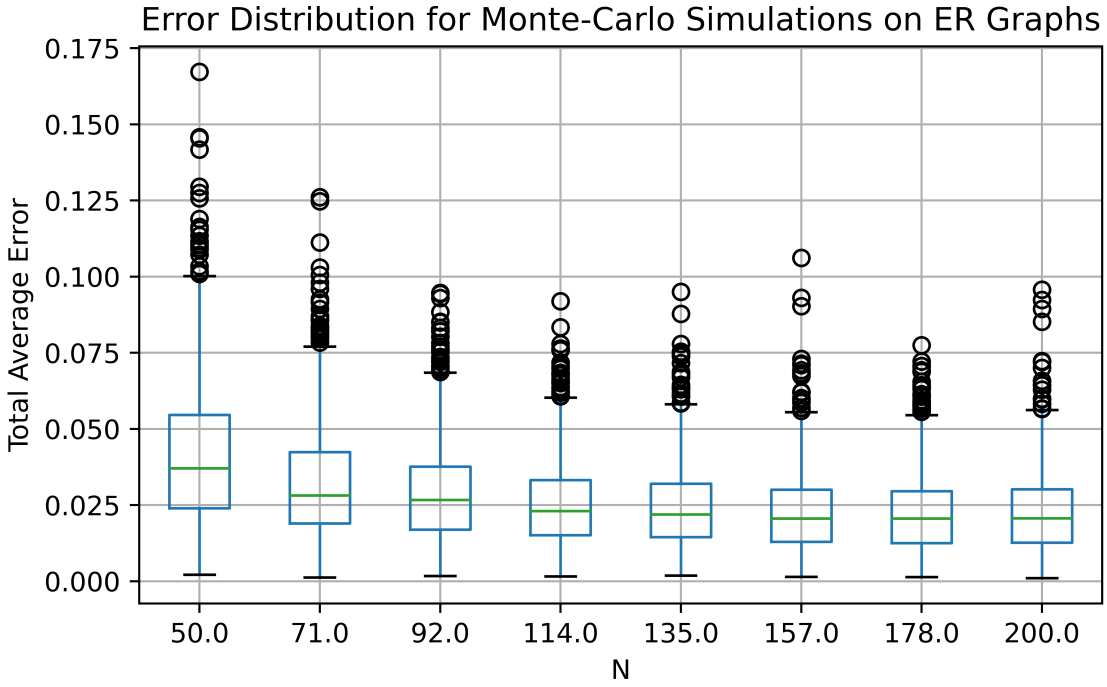

Figure 3.2: Distribution of total average error across N for 9000 stochastic runs. Parameters r, k, γ, and ω varied on the intervals $[0.5, 1]$, $[10, 30]$, $[0.1, 1]$ and $[0.1, 1]$ respectively.

To capture the effects of all parameters, we utilize standard OLS regression techniques for analysis. Figure 3.3 shows output tables for regressions run with total average error as the dependent variable, and other model parameters as independent variables. This method allows us to look at the effects of each parameter on total average error. Figure 3.3 is based on a parameter sweep run over 9000 simulations with $r \in [0.5, 1]$, $k \in [10, 30]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$. We run two regressions to examine the effects of model parameters on total average error, first including only N and k as explanatory variables, then including N and k while controlling for effects of r, γ, and ω. In both cases we find an low R^2 value (sum of residuals squared), roughly 0.1 for each regression, indicating that only 10% of the variance in the total average error is explained by the parameters N, k, r, γ, and ω. This isn't surprising, as we believe most of the error in our total average error is driven by stochastic Monte-Carlo methods.

When looking at the regression coefficients (the "marginal effect" on total average error when increasing the explanatory variable by 1) we find that increasing N and k always has a negative effect on error. We can interpret the regression coefficients for N, for each node added to the system, total average error falls by a population proportion of 0.0001. For k, increasing average degree of the system by 1 edge decreases total average error by population proportion of 0.0002.

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.0445	0.001	71.073	0.000	0.043	0.046
N	-0.0001	3.48e-06	-32.079	0.000	-0.000	-0.000
k	-0.0002	2.09e-05	-7.256	0.000	-0.000	-0.000

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.0391	0.001	37.582	0.000	0.037	0.041
N	-0.0001	3.46e-06	-32.207	0.000	-0.000	-0.000
r	0.0029	0.001	3.006	0.003	0.001	0.005
gamma	0.0020	0.001	3.762	0.000	0.001	0.003
omega	0.0038	0.001	7.190	0.000	0.003	0.005
k	-0.0002	2.08e-05	-7.285	0.000	-0.000	-0.000

Figure 3.3: 2 regressions with total average error as response variable, and model parameters as independent variables. Regressions performed on data with $r \in [0.5, 1]$, $k \in [10, 30]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$, thus predictions are only valid in that space. From top to bottom, $R^2 = 0.107, 0.115$.

When adding parameters r, γ, and ω, we see that the regression coefficients on N and k remain unchanged, and statistically significant. The coefficient on N is expected to be negative, because as N increases the stochastic variance in the system becomes less pronounced. The k coefficient is also expected to be negative, as increasing connectivity throughout the system brings us closer to our mean-field assumption of full connectivity.

In the regression controlling for other model parameters r, γ, and ω all maintain positive, and statistically significant, coefficients. This suggests that our model is most true to MFT at the bounds of the parameter space sweep, $r = 0.5$, $\gamma = 0.1$, $\omega = 0.1$. It appears that the agent-based model is more true to mean-field in slow, and widespread adoption. The large coefficients on these parameters as compared to N and k are not a cause for concern, as the coefficients represent the expected change in total average error with an increase of 1. However, r, γ and ω are all limited for our model on the range $[0, 1]$.

3.1.3 What Does "Error" Even Mean?

We see generally that under that our conception of error as described in section 3.1.1, our Monte-Carlo model is robust for large systems with high connectivity. But what about small systems small systems with non-random connectivity? In other words, what about networks which do not do a good job of approximating our MFT assumptions? Of course, in this scenario, our "error" as defined in section 3.1.1 is high, because our model does a poor job matching mean-field. It is not clear, for a small and sparse system, how we may go about understanding the validity of our model. The best way may be to compare model predictions with historical time-series innovation adoption data, with an unchanging network, and resulting in adopter-luddite dichotomies. However, to our knowledge, such a dataset does not exist.

3.2 Effects of Local Environment on Innovation Adoption Under the T-LISA Model

The power of implementing the T-LISA model on agent-based systems, is that it allows us to dig deeper into the characteristics of diffusion under the T-LISA model. In the mean field, local environment is not considered, however in the real world, innovation diffusion is almost completely driven by the first degree connections. For instance, you may adopt a political stance based on your nuclear family's politics. Or you may choose to like a particular sports team because your friends like that sports team. In this section of the thesis, we will use regression analysis to explore how local environment impacts an agent's likelihood of adopting innovation.

To quantify environments, analysts have developed several metrics. Some of these metrics exist at the local level: calculable for each node in the system; and some exist on a global level: calculable for the network system as a whole. In this section, we analyze how local metrics, as detailed in Table 3.1, impact a nodes tendency towards acceptance.

Metric	**Definition**
Degree [13]	The number of edges connected to a node.
Closeness Centrality [13] [26]	How close a node is to all other nodes in the network, based on the shortest paths between them. A high closeness centrality indicates that a node is close with many other nodes.
Betweenness Centrality [13] [3]	The proportion of all shortest paths in the network that pass through a given node, indicating the node's role as a bridge.
Eigenvector Centrality [13]	A measure of the influence of a node in a network, taking into account the centrality of its neighbors.
Clustering Coefficient [13] [23]	The degree to which nodes in a graph tend to cluster together, measured as the ratio of existing links connecting a node's neighbors to each other to the maximum possible number of such links.
Triangles [13]	The number of triangles that include the node as one vertex, indicating the node's tendency to form tightly knit groups.

Table 3.1: Node-level structural metrics used as independent variables to predict A_{tMax}.

3.2.1 Non-Noramlized Regression Results

To test how these metrics effect outcomes, we run large batches of stochastic simulations. Note, each simulation includes a new randomly generated ER graph, along with differently initialized seeding *susceptibles*. (On average, 20% of the population is initialized as *susceptible*.) For each simulation, we create a dataframe which includes rows for all nodes, their independent struc-

	coef	std err	t	P> |t|	[0.025	0.975]
const	-0.1483	0.499	-0.297	0.766	-1.126	0.829
degree	0.0017	0.009	0.194	0.846	-0.015	0.019
closeness	1.3345	1.185	1.126	0.260	-0.989	3.657
betweenness	-18.1948	18.391	-0.989	0.323	-54.242	17.853
eigenvector	2.8850	2.484	1.161	0.246	-1.985	7.755
clustering	-0.3822	0.605	-0.632	0.527	-1.568	0.803
triangles	0.0001	0.001	0.103	0.918	-0.003	0.003

Figure 3.4: Regression results for a system with $N = 300$, $k = 30$, $r = 0.9$, $\gamma = 0.08$, and $\omega = 0.08$. For this regression, 50 simulations were run, leading to 15000 observations.

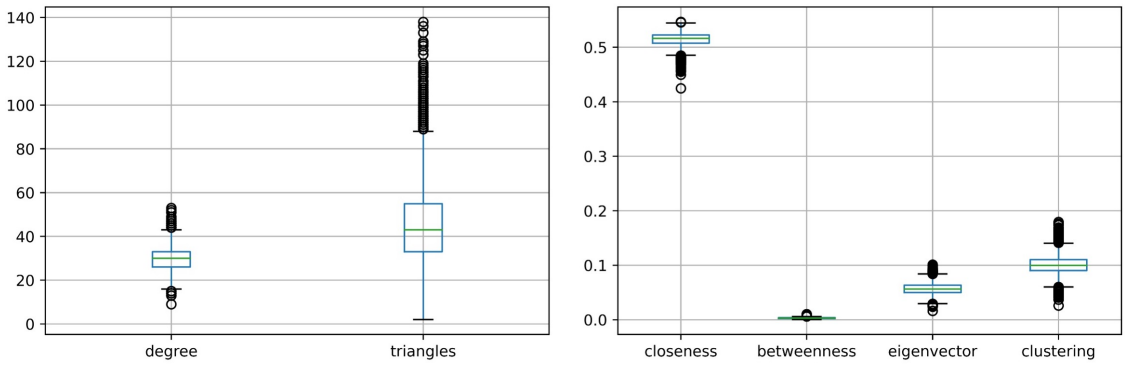

Figure 3.5: Distribution of explanatory variables included in regression described by Figure 3.4.

tural variables, and their state outcome. We use a linear probability model[3] (lpm) to analyze the effects of the metrics in Table 3.1, on the binary response variable *"adopted"*, where 1 represents adoption and 0 represents rejection. We use an lpm rather than logit or probit methods for easy probabilistic interpretation of coefficients. Note that due to our use of an lpm, we implicitly introduce heteroskedasticity into our regression, and therefore have unreliable standard errors and significance [14].

We begin by using our lpm for 20 instances of the T-LISA model, run on the parameters $N = 300$, $k = 30$, $r = 0.9$, $\gamma = 0.08$, and $\omega = 0.08$. The results can be seen in Figure 3.4. The magnitudes of the coefficients predicted by this regression are quite difficult to interpret. When examining the distribution of data, as described by Figure 3.5, it appears that the magnitude of the coefficients are largely a function of the mean and spread of data. For example, of course an increase of 1 in betweenness would have a massive effect when betweenness for our system only ranges from 0 to 0.01. This makes comparison of our coefficient magnitudes in Figure 3.4 fraught.

[3] An lpm is a regular OLS regression performed with a binary response variable.

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.4656	0.073	6.421	0.000	0.323	0.608
degree_normalized	0.0748	0.386	0.194	0.846	-0.682	0.831
closeness_normalized	0.1627	0.144	1.126	0.260	-0.121	0.446
betweenness_normalized	-0.1818	0.184	-0.989	0.323	-0.542	0.178
eigenvector_normalized	0.2443	0.210	1.161	0.246	-0.168	0.657
clustering_normalized	-0.0586	0.093	-0.632	0.527	-0.240	0.123
triangles_normalized	0.0196	0.190	0.103	0.918	-0.352	0.391

Figure 3.6: Regression results for a system with $N = 300$, $k = 30$, $r = 0.9$, $\gamma = 0.08$, and $\omega = 0.08$, with explanatory variables forced on the interval $[0, 1]$.

3.2.2 Normalized Regression Results

To counteract this issue, we use min-max normalization to force all node level metric distributions on the interval $[0, 1]$. Although this disrupts the typical probabilistic interpretation of an lpm (change in likelihood per unit increase) it facilitates useful relative comparison between the effects of our node level metrics. Figure 3.6 shows results from lpm run on the same dataset as Figure 3.4, but with normalized explanatory variables. As expected, the signs on all coefficients remain unchanged, but magnitudes change drastically.

Out of all explanatory variables in Figure 3.6, eigenvector centrality and closeness centrality appear to be the main drivers of adoption. This suggests that for a relatively large system with a fairly high interconnectedness, a node with connections to other highly connected nodes and who is close to many other other nodes in the network, is more likely than other nodes to become an *adopter*. The relatively large and negative coefficient on betweenness indicates that, out of our explanatory variables, it is the main driver of luddism. This suggests a node which is acting as a bridge between several other nodes is likely to reject innovation.

We then run a similar regression, with the same normalized explanatory variables, at a different point in parameter space. This allows us to understand node-level structural variables effects' sensitivity to global structural parameters. We reduce N to 50, and reduce k to 5, such that the probability of edge formation in ER graph formation remains the same (0.1.) The results for the lpm are shown in Figure 3.7. Notably, the signs on both normalized closeness and normalized triangles flips, indicating that at this smaller system size dynamics are clearly different. We also see changes in the drivers of adoption, with normalized degree becoming dominant. Our lpm says that at this smaller system size, a highly connected node is much more likely to adopt than a more isolated node. The main driver of rejection stays consistent, with bridging nodes of high in-betweenness appearing to turn *luddite* more often.

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.4744	0.057	8.388	0.000	0.364	0.585
degree_normalized	0.7503	0.102	7.367	0.000	0.551	0.950
closeness_normalized	-0.0880	0.108	-0.812	0.417	-0.301	0.125
betweenness_normalized	-0.2222	0.070	-3.161	0.002	-0.360	-0.084
eigenvector_normalized	0.0527	0.068	0.777	0.437	-0.080	0.186
clustering_normalized	-0.0033	0.037	-0.090	0.928	-0.075	0.068
triangles_normalized	-0.1556	0.071	-2.190	0.029	-0.295	-0.016

Figure 3.7: Regression results for a system with $N = 50$, $k = 5$, $r = 0.9$, $\gamma = 0.08$, and $\omega = 0.08$, with explanatory variables forced on the interval $[0, 1]$.

3.2.3 Local Structure Predictive Power

Ultimately, the power of our node-level structural explanatory variables to predict the adoption for a particular node is low. When fitting logit regressions with our explanatory variables from Figure 3.7, we find a McFadden's pseudo R^2 value of less than 0.01. This indicates that the local metrics which we have examined in this section do not have a major impact on a node's final state. We cannot effectively predict a single node's final state based on local structure, in our ER graph simulations. Thus, we turn to global structural variables.

3.3 Effects of Global Environment on Innovation Adoption Under the T-LISA Model

To test the effects of a global environment on the agent-based T-LISA model, we once again use standard OLS techniques. We run $30,000$ simulations across parameter space $N \in [50, 200]$ $r \in [0.5, 1]$, $k \in [10, 30]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$, with batch size 10, and use the proportion of Adopters in the final state as our dependent variable, A_{tMax}. Our new framework for regression, looking at results from full T-LISA model runs, allows us to include parameters r, γ and ω in our regression, along with other global network metrics as seen in Table 3.2.

We start with a null regression, including no structural variables, as seen in Figure 3.8. We expect that the coefficients on each of these parameters will be negative, as according to their definitions in Section 2.2.1, they positively scale the spread of luddism throughout the system for different reasons. (Thus, the proportion of adopters in the steady state, A_{tMax} should be smaller.) After running the regression we find that, indeed each of these coefficients are negative. According to the fit of the OLS model (R^2) r, γ and ω explain 41.7% of the variance in L_{tMax}. Furthermore, from the coefficients we are able to see that γ has the smallest average effect on the final proportion of Adopters in the system, and ω has the largest effect has the largest effect.

When we introduce the most basic structural variables into our model, N and k, we see regression fit shoot up to $R^2 = 0.807$. Thus we can conclude that these structural variables are incredibly important for understanding the results of the Monte-Carlo based T-LISA model. Together they explain roughly 33% of the variation in A_{tMax}. Figure 3.9 shows results from these regressions. These results give us the first strong indication of how structural variables effect our final adopter proportion.

Metric	Definition
N	The number of nodes within the graph.
k	The average degree of nodes within the graph.
Attribute Assortativity Coefficient [19] [13]	Measures the similarity of connections in the graph with respect to the node attribute 'state'. For a network where nodes have attributes, this coefficient quantifies how likely nodes with similar attributes are to be connected.
Average Clustering [25] [13]	Represents the overall level of clustering in the network. It is the average of the local clustering coefficients of all the nodes, reflecting the degree to which nodes in a graph tend to cluster together.
Average Shortest Path Length [13]	The average number of steps along the shortest paths for all possible pairs of network nodes. It gives a measure of the efficiency of information or connectivity spread on the network.
Transitivity [13]	A global measure of clustering that quantifies the fraction of all possible triangles in the network. This metric provides an indication of the probability that two neighbors of a node are neighbors themselves.

Table 3.2: Global network structural variables, used as independent regression variables to predict A_{tMax}.

The coefficient on N in Figure 3.9 suggest that as the system gets larger, the proportion of Adopters in the steady state falls. This likely reflects to the N in the denominator of Equation 2.2 ($I \to S$ propensity) as compared to the k_i in the denominator of Equation 2.1 ($I \to L$ propensity.) As N gets larger and k stays constant the pressure to become Luddite, as described by the T-LISA model, increases. The positive coefficient on k_i can be described using similar logic. As degree of node i increases under constant N, that node is more likely to become an Adopter, as described by Equations 2.2 and 2.1. Both of these effects are anticipated due to model rules, and our regression confirms them. Our regression also shows that changing k has a larger relative effect than changing N. On average, when controlling for the variables included in Figure 3.9, we see that increasing average degree by 1 resulted in a 0.0087 increase in final Adopter proportion, and increasing N by 1 node decreased the final Adopter proportion by 0.0014.

Next we add the remaining global structural variables (from Table 3.2) into the regression. Results can be seen in Figure 3.10. The inclusion of these new variables cause a 0.015 bump in R^2. Based on statistical significance, we can say that indeed, average shortest path length and attribute assortativity coefficient are helpful in predicting A_{tMax}, but only marginally so. Transitivity and average clustering coefficient do not have statistically significant effects.

The attribute assortativity coefficient measures the similarity of connections in the graph with respect to state. Note that the mean of our attribute assortativity coefficient across $30,000$ simulation is -0.014725, indicating that Adopter nodes on average are likely to have connections

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.8780	0.004	250.592	0.000	0.871	0.885
gamma	-0.1433	0.002	-65.026	0.000	-0.148	-0.139
omega	-0.2636	0.002	-119.644	0.000	-0.268	-0.259
r	-0.2124	0.004	-53.562	0.000	-0.220	-0.205

Figure 3.8: Regression results with A_{tMax} as a response variable, for $30,000$ simulation runs with $r \in [0.5, 1]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$. $R^2 = 0.417$.

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.8815	0.002	357.530	0.000	0.877	0.886
gamma	-0.1433	0.001	-113.122	0.000	-0.146	-0.141
omega	-0.2636	0.001	-208.083	0.000	-0.266	-0.261
r	-0.2125	0.002	-93.182	0.000	-0.217	-0.208
N	-0.0014	8.2e-06	-172.925	0.000	-0.001	-0.001
k	0.0087	4.94e-05	175.630	0.000	0.009	0.009

Figure 3.9: Regression results with A_{tMax} as a response variable, for $30,000$ simulation runs with $N \in [50, 200]$ $r \in [0.5, 1]$, $k \in [10, 30]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$. $R^2 = 0.807$.

	coef	std err	z	P> |z|	[0.025	0.975]
const	1.0636	0.010	107.109	0.000	1.044	1.083
gamma	-0.1427	0.001	-109.540	0.000	-0.145	-0.140
omega	-0.2637	0.001	-211.886	0.000	-0.266	-0.261
r	-0.2122	0.002	-96.267	0.000	-0.216	-0.208
N	-0.0006	1.9e-05	-30.273	0.000	-0.001	-0.001
k	0.0028	0.000	19.167	0.000	0.002	0.003
atr_assort_coef	0.0461	0.013	3.423	0.001	0.020	0.073
transitivity	0.0871	0.179	0.486	0.627	-0.264	0.438
avg_clustering	0.1744	0.179	0.975	0.330	-0.176	0.525
avg_shortest_path_length	-0.1132	0.004	-26.818	0.000	-0.121	-0.105

Figure 3.10: Regression results with A_{tMax} as a response variable, for $30,000$ simulation runs with $r \in [0.5, 1]$, $\gamma \in [0.1, 1]$, and $\omega \in [0.1, 1]$. $R^2 = .822$.

with Luddites in the final state [19]. We can interpret the regression coefficient associated with assortativity: when Adopters are on average more connected with other Adopters, there is an increase in the final amount of Adoption throughout the system. This indicates that there is a small synergy between connected adopters, which encourages broader adoption throughout the entire system.

Average shortest path length is more easily interpretable. Our regression coefficient for it suggest that networks with a smaller average shortest path length, where innovation has less far to spread, result in more widespread adoption. We also see that, because average shortest path length is inversely related to k, the magnitude of the coefficient on k drops from Figures 3.9 to 3.10.

Ultimately, we are left to conclude that the Monte-Carlo based simulation of the T-LISA model on ER graphs results in the highest final proportion of Adopters in small, highly connected, systems. It does not appear that other variables detailed in Table 3.2 have highly impactful effects, however around the margins, a short distance for innovation to travel, as well as synergy between clusters of Adopter, help innovation become more widespread. The positive coefficient on attribute assortativity coefficient suggests a synergy in adoption when multiple Adopter nodes are connected.

Chapter 4

Conclusion, Discussion, and Future Plans

In this thesis, we have proposed a novel model for innovation diffusion, the T-LISA model, and explored computational solutions to this model both in the mean-field and under stochastic simulation. Our model extends Melor et al.'s LISA model [17] by finishing in a stable state consisting of only Adopters and Luddites, without Ignorants. We accomplish this by including pressure from Adopters on Ignorants. We've develop agent-based rules, where propensities to adopt or reject innovation are based on neighboring nodes' states. We used Monte-Carlo methods to implement these rules on Erdos Renyi random graphs in stochastic batches, and then quantified the match between mean field and Monte-Carlo simulation. Finally, we used regression analysis to understand the main drivers of error and adoption in our network based models. We are proud of our interdisciplinary methods, as physics literature often sufferers from information siloing.

Although we expected both local and global network structure to play a strong role in adoption, our regression analysis shows that network structure, outside of parameters N and k, only plays a marginal role in diffusion under the T-LISA model. In part, this may be due to the fact that we've only explored our model's output on Erdos Renyi graphs, where many structural graph variables are, to some extent, homogeneous. We chose to analyze the Monte-Carlo based T-LISA model on ER graphs because the difference in MFT and Monte-Carlo results was readily quantifiable.

For ER graphs, our regression analysis of global structural variables N and k does lead us to the conclusion that, under T-LISA dynamics, innovation becomes most widespread in smaller and more densely connected graphs. This contributes to literature on the formation of highly polarized echo chambers in networks. Previous work has studied the formation of echo chambers in particular media networks [6]. We've provided a more abstract approach to the problem of innovation diffusion, which suggests that in larger graphs, Adoption of innovations, radical or nominal, are most likely to occur in small and densely connected subgraphs. Thus these small and densely connected subgraphs are also more likely to become polarized than large and sparse graphs.

In terms of other global structural variables other than N and k, we found statistically significant effects for both attribute assortativity coefficient and average shortest path length. These effects further support our finding that small and densely connected networks are likely to become polarized, especially those in which early adoption spreads without luddism. The effect of attribute assortativity coefficient also suggests that the outcome of small densely connected clusters is sensitive to the initial ratio of adoption to luddism in a the cluster, as Adopter pairs have synergistic

effects. In other words, the adoption or luddism of the first few nodes who reach the final state may be highly important in determining the networks final state, especially for small and densely connected networks. To further test this theory, time series analysis would be appropriate.

In future, we would certainly like to study the T-LISA model dynamics on previously collected network data, where both local and global structural metrics are more widely distributed. We believe that both social media networks and academic co-authorship networks would be interesting, as these networks are representative of the spread of ideas. We believe that the spread of ideas is the most salient and interesting use case for our model.

We are also interested in comparing our theoretical model with real world time series data. It certainly would be possible to scrape network-based sentiment data through time, on a platform like Instagram, Twitter, or Reddit. Our model's parameters ω and γ could be then tuned to match real world adoption rates, and pressures from neighbors to reject. Thus our model could become predictive in nature, and help guide interventions during innovation diffusion.

Another important contribution of this thesis is the original work done in developing an algorithm to compare mean field and stochastic simulation results, as detailed in Section 3.1.1. As far as we are aware, such a quantitative comparison of total average error has not previously been done in statistical physics literature. Typically, comparison of mean field and numerical simulation is kept to a nondescript 'qualitative match.' The quantitative nature of the total average error measure allows for comparison of mean field and numerical simulation at different points in parameter space, across systems, and even across projects. Our algorithm is easily generalizable for any systems which have both mean field and numerical solutions.

Bibliography

[1] Frank M. Bass. A new product growth for model consumer durables. *Management Science*, 15(5):215–227, 1969.

[2] Staci L Benoit and Rachel F Mauldin. The "anti-vax" movement: a quantitative report on vaccine beliefs and knowledge across social media. *BMC Public Health*, 21(1):1–11, 2021.

[3] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[4] ULRIK BRANDES, GARRY ROBINS, ANN McCRANIE, and STANLEY WASSERMAN. What is network science? *Network Science*, 1(1):1–15, 2013.

[5] Ronald S Burt. Structural holes and good ideas. *American journal of sociology*, 110(2):349–399, 2004.

[6] Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118(9):e2023301118, 2021.

[7] Nihal Durmaz and Engin Hengirmen. The dramatic increase in anti-vaccine discourses during the covid-19 pandemic: a social network analysis of twitter. *Human vaccines & immunotherapeutics*, 18(1):2025008, 2022.

[8] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 2010.

[9] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960.

[10] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, 78(6):1360–1380, 1973.

[11] Mariangela Guidolin and Piero Manfredi. Innovation diffusion processes: Concepts, models, and predictions. *Annual Review of Statistics and Its Application*, 10:451–473, 2023.

[12] Mark Hoffman. Methods for network analysis, 2021.

[13] Networkx algorithms generation documentation.

[14] Mark W. Watson James H. Stock. *Introduction to Econometrics*. Pearson, 2020.

[15] Jonas S Juul and Mason A Porter. Hipsters on networks: How a minority group of individuals can lead to an antiestablishment majority. *Physical Review E*, 99(2):022313, 2019.

[16] Helmut Mehrer. *Diffusion in solids: fundamentals, methods, materials, diffusion-controlled processes*, volume 155. Springer Science & Business Media, 2007.

[17] Andrew Mellor, Mauro Mobilia, S. Redner, Alastair M. Rucklidge, and Jonathan A. Ward. Influence of luddism on innovation diffusion. *Phys. Rev. E*, 92:012806, Jul 2015.

[18] M. Newman. *Networks*. OUP Oxford, 2018.

[19] M. E. J. Newman. Mixing patterns in networks. *Phys. Rev. E*, 67:026126, Feb 2003.

[20] Alberto Palloni. Diffusion in sociological analysis. *Diffusion processes and fertility transition: Selected perspectives*, pages 67–114, 2001.

[21] Everett M. Rogers. *Diffusion of Innovations, 5th Ed.* New York: The Free Press, 2003.

[22] Francis Russell. Pox populi: Anti-vaxx, anti-politics. *Journal of Sociology*, 59(3):699–715, 2023.

[23] Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and Janos Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007.

[24] Ivo F Sbalzarini. *Analysis, modeling, and simulation of diffusion processes in cell biology*. PhD thesis, ETH Zurich, 2006.

[25] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[26] Stanley Wasserman and Katherine Faust. Social network analysis: Methods and applications. 1994.

Appendix A

Python Tools

A.1 `odeint`

`odeint` is a function from `scipy.integrate` which allows users to find numerical solutions to first-order ordinary differential equations (ODEs) [**?**]. The function uses the initial configuration of a system to iteratively calculate succeeding configurations. This process outputs a graph of the dependent variable versus the independent variable. For most physical systems, the independent variable is time, allowing the user to determine the state of a system described by a first-order ODE at any time. While `odeint` can only process first-order equations, it can accept systems of ODEs. Thus, we can use Python to consider higher-order equations by simply expressing them as systems of first-order equations.

`odeint` is imported using

```
from scipy.integrate import odeint
```

and called using

```
y = odeint(F, y0, t)
```

F is a Python function F(y, t) which accepts a one-dimensional array (y) that describes the current state of the system and a scalar (t) which describes the current value of the independent variable. y0 is a one-dimensional array that describes the initial state of the system (initial value of y). t is the array of independent variable values, the first of which corresponds to y0. Each subsequent value in t corresponds to subsequent states of the system, which are successively stored in y. Below we include a sample program from [**?**], which is designed to find a numerical solution of the position of a simple harmonic oscillator:

```
#solve_ode.py
"""ODE solver for harmonic oscillator."""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Define Oscillator Function
def F(y, t):
```

```
    """
    Return derivatives of second-order ODE y'' = -y.
    """

    dy = [0, 0]          #Create a list to store derivatives.
    dy[0] = y[1]                 #Store first derivative of y(t).
    dy[1] = -y[0]                #Store second derivative of y(t).

    return dy

#Create array of time values to study
t_min = 0
t_max = 10
dt = 0.1
t= np.arange(t_min, t_max, dt)

#Set initial conditions
y0 = (1.0, 0.0)

plt.figure()                     #Create figure; add plots later.
y = odeint(F, y0, t)             #Call odeint. Pass it the function, the
    initial conditions, and the time array.
plt.plot(t, y[:, 0], linewidth=2)       #Plot the resulting numerical
   solution.
plt.title("Numerical Solution of SHO")
plt.xlabel("t")
plt.ylabel("y")
```

Appendix B

Python Code for Running Simulations

B.1 Erdos-Renyi Random Graph Simulation Class

```
@author: markosuchy

#%%
import networkx as nx
import matplotlib.pyplot as plt
import random
import pandas as pd

import scipy, scipy.integrate
import numpy as np


#%%
class ER_Graph():
    def __init__(self, N, simulations, k, inital_conditions, r, gamma,
        omega):
        self.N = N
        self.gamma = gamma
        self.omega = omega
        self.r = r
        self.simulations = simulations
        self.k = k
        self.inital_conditions = inital_conditions

        #Iitialize a bunch of lists that will be plotted
        self.L_solution = []
        self.I_solution = []
        self.S_solution = []
        self.A_solution = []
```

```
        P = k/(N-1)
        self.P = P
        self.G = nx.erdos_renyi_graph(N, P)

        state_list = ['L', 'I', 'S', 'A']
        initial_state = random.choices(state_list, weights=
            inital_conditions, k = N)
        nx.set_node_attributes(self.G, dict(zip(self.G.nodes(),
            initial_state)), 'state')

    def simulation_step(self):
        for n in range(self.N):
            neighbors = list(self.G[n].keys())

            #Get the noode n's state
            node_n_state = nx.get_node_attributes(self.G, "state").get(
                n)

            #Get lists of suceptible neighbors
            suceptible_neighbors = []
            adopted_neighbors = []
            for neighbor in neighbors:
                if nx.get_node_attributes(self.G, "state").get(neighbor
                    ) == 'S':
                    suceptible_neighbors.append(neighbor)
                elif nx.get_node_attributes(self.G, "state").get(
                    neighbor) == 'A':
                    adopted_neighbors.append(neighbor)



            #Define propensities
            #IN THIS VERSION OF CODE, ADOPTED NODES ARE FACTORED IN!
            S_to_A = self.gamma
            #I_to_L = r*gamma*(len(suceptible_neighbors)+len(
                adopted_neighbors))/len(neighbors)# - Pre Nov. 2023
                update
            #This if-else block was added 2/18/24
            if len(neighbors) > 0:
                I_to_L = self.r*(self.gamma*len(suceptible_neighbors) +
                    self.omega*len(adopted_neighbors))/len(neighbors)
            else:
                I_to_L = 0
            I_to_S = (len(suceptible_neighbors)+len(adopted_neighbors))
                /self.N
```

```
            if node_n_state == "S":

                #Define possible new states
                possible_states = ["S", "A"]

                #choose a state based on probability
                state_assignment = random.choices(possible_states,
                    weights=(1 - S_to_A, S_to_A), k = 1)
                nx.set_node_attributes(self.G, {n:state_assignment[0]},
                    'state')



                #Assign that state to the node at hand

            if node_n_state == "I":
                possible_states = ["I", "L", "S"]

                state_assignment = random.choices(possible_states,
                    weights=(1 - (I_to_L + I_to_S), I_to_L, I_to_S), k =
                    1)
                nx.set_node_attributes(self.G, {n:state_assignment[0]},
                    'state')


    def run_one_simulation(self, animate = False):
        #re-initialize the graph, so states are all re-randomized.
        self.__init__(self.N, self.simulations, self.k, self.
            inital_conditions, self.r, self.gamma, self.omega)

        #make sure solutions lists are empty
        self.L_solution = []
        self.I_solution = []
        self.S_solution = []
        self.A_solution = []

        #Run the simulation:
        for simulation in range(self.simulations):

            #Actuallly do the simulation, and print progress
            self.simulation_step()

            if (simulation + 1) %10 == 0:
```

```
            print("ran simulation step:", simulation + 1, "/", self
                .simulations)

        #Store the amount of number of each node in a lists, which
            will then be plotted
        S_nodes = [n for n,v in self.G.nodes(data=True) if v['state
            '] == 'S']
        self.S_solution.append(len(S_nodes))

        L_nodes = [n for n,v in self.G.nodes(data=True) if v['state
            '] == 'L']
        self.L_solution.append(len(L_nodes))

        I_nodes = [n for n,v in self.G.nodes(data=True) if v['state
            '] == 'I']
        self.I_solution.append(len(I_nodes))

        A_nodes = [n for n,v in self.G.nodes(data=True) if v['state
            '] == 'A']
        self.A_solution.append(len(A_nodes))

        if self.check_stable_state():
            print(f"STABLE STATE REACHED at step {simulation + 1}")
            break

        if animate:
            self.draw_graph()


def run_n_simulations(self, data_path, n = 1, store_data = True):
    for n in range(1, n + 1):
        print(f"running simulation {n}...")
        self.run_one_simulation()

        if store_data == True:
            self.save_data(path = data_path, simulation_batch_num =
                n)

def check_stable_state(self):
    if self.I_solution[len(self.I_solution) - 1] == 0 and self.
       S_solution[len(self.S_solution) - 1] == 0:
        return True
    else:
        return False
```

```
def draw_graph(self, with_labels = True):
    #set position of now network should be plotted each time
    pos = nx.spring_layout(self.G)

    color_state_map = {"L": 'blue', "I": 'green', "S": 'yellow', "A
       ": 'red'}
    nx.draw(self.G, node_size = 100,
            node_color=[color_state_map[node[1]['state']] for node
               in self.G.nodes(data=True)],
            with_labels = with_labels, pos = pos)
    plt.title('graph')
    plt.show()

def plot_curves(self):
    L_pop_fraction = [element / self.N for element in self.
       L_solution]
    I_pop_fraction = [element / self.N for element in self.
       I_solution]
    S_pop_fraction = [element / self.N for element in self.
       S_solution]
    A_pop_fraction = [element / self.N for element in self.
       A_solution]

    #plot each line with a different label
    plt.plot(L_pop_fraction, label = "Luddite", color = "blue")
    plt.plot(I_pop_fraction, label = "Ignorant", color = "green")
    plt.plot(S_pop_fraction, label = "Succeptible", color = "orange
       ")
    plt.plot(A_pop_fraction, label  = "Accepted", color = "red")

    #title = "T-LISA Simulation: " + "N = " + str(N) + ", k = " +
       str(k) + ", $\gamma{}$ = " + str(round(gamma, 5)) + ", $\
       omega{}$ = " + str(round(omega, 5)) + " $r$ = " + str(r) + "
        sim#: " + str(simulation_number)
    title = "title"
    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlabel("simulations")
    plt.ylabel("Population Proportion")

    plt.show()

def save_data(self, path, simulation_batch_num = "not batched!"):
```

```
#This function will throq an error if run_simulation hasn't
    been run prioir
#Consider try-except block?


sim_list = list(range(1, self.simulations+1))

#save paramaters that the model was run on
self.params = [[self.N, self.simulations, self.k, self.P, self.
    inital_conditions, self.r, self.gamma, self.omega]]

##note, this weird dataxtructure (dictionary nested in a list)
    is so that
self.params = {"N": self.N,
                "simulations": self.simulations,
                "k": self.k,
                #"P": self.P, -P is calculated from k and N,
                    and makes naming worse, so it's excluded
                "inital_conditions": self.inital_conditions,
                "r": self.r,
                "gamma": self.gamma,
                "omega": self.omega,
                "Simulation_batch_num": simulation_batch_num}
#df["paramaters"] = self.params

#df["L_sol"] = self.L_solution

#Set up the dict that will
self.data = {#"simulation": sim_list,
             "L_sol": self.L_solution,
             "I_sol": self.I_solution,
             "S_sol": self.S_solution,
             "A_sol": self.A_solution}

#define df
df = pd.DataFrame(self.data)
df["simulation"] = df.index + 1
df["params"] = pd.Series([self.params])

#Name the datafile
file_name = str(self.params).replace(":","=").replace("'", "")
    + ".json"
file_path = path + file_name

#Save the file
```

```
        df.to_json(file_path)

        return df
```

B.2 Simulation Class for Rescaled MFT

```
class Rescaled_MFT():
    def __init__(self, N, tMax, k, inital_conditions, r, gamma, omega):
        self.N = N
        self.k = k
        self.inital_conditions = inital_conditions
        self.r = r
        self.gamma = gamma
        self.omega = omega
        self.tMax = tMax

    # This defines a function that is the right-hand side of the ODEs
    # Warning!  Whitespace at the begining of a line is significant!
    def rhs(self, Y, t, omega, gamma, r):
        '''
        SIR model.

        This function gives the right-hand sides of the ODEs.
        '''

        # Convert vector to meaningful component vectors
        # Note: Indices start with index 0, not 1!
        L = Y[0]
        I = Y[1]
        S = Y[2]
        A = Y[3]
        # The right-hand sides
        dL = self.gamma * self.r * S * I + self.omega * self.r * A * I
        #dI = -(S+A)*I - self.gamma * self.r * S * I - self.omega *
            self.r * A *  - Unscaled version
        dI = -((self.gamma * self.r) + self.k/self.N)*S*I - A*(self.k/
            self.N)*I - self.omega * self.r * A * I
        #dS = (S+A)*I - self.gamma * S - unscaled version
        dS = S * ((self.k/self.N) * I - self.gamma) + A * (self.k/self.
            N) * I
        dA = self.gamma*S
        # Convert meaningful component vectors into a single vector
        dY = [ dL, dI, dS, dA ]

        return dY
```

```
def solve_MFT(self):
    # Time vector for solution
    self.T = np.linspace(0, self.tMax, self.tMax*20 + 1)
    solution = scipy.integrate.odeint(self.rhs,self.
       inital_conditions, self.T, args = (self.omega, self.gamma,
       self.r))

    self.L_sol = solution[:, 0]
    self.I_sol = solution[:, 1]
    self.S_sol = solution[:, 2]
    self.A_sol = solution[:, 3]

#Note - I don't actually need to save data here bc this isn't
   stochastic and can be run very quickly.
#But... maybe I should?
def concat_data(self):
    #Set up the inital df
    data = {"T" : self.T,
            "L_sol_MFT" : self.L_sol,
            "I_sol_MFT" : self.I_sol,
            "S_sol_MFT" : self.S_sol,
            "A_sol_MFT" : self.A_sol
        }
    self.df = pd.DataFrame(data)

    #add params
    self.params = {"N": self.N,
                   "tMax": self.tMax,
                   "k": self.k,
                   #"P": self.P, -P is calculated from k and N,
                       and makes naming worse, so it's excluded
                   "inital_conditions": self.inital_conditions,
                   "r": self.r,
                   "gamma": self.gamma,
                   "omega": self.omega
                   }
    self.df["params"] = pd.Series([self.params])

    ##NEED THE ACTUAL PA


def plot_curves(self):

    plt.plot(self.T, self.L_sol, label = "Luddites", color = "blue"
```

```
        )
        plt.plot(self.T, self.I_sol, label = "Ignorants", color = "
            green")
        plt.plot(self.T, self.S_sol, label = "Susceptibles", color = "
            orange")
        plt.plot(self.T, self.A_sol, label = "Adopters", color = "red")

        plt.show()
```

B.3 Parameter Sweep

```
#Constants
simulations = 100
inital_conditions = [0, .8, .2, 0]
data_path = "/Users/markosuchy/Desktop/Honors Thesis/json data/
    Erdos_Renyi/Param_Sweep_Data/"
batch_size = 5

#Parameter Space Setup
N_space = np.linspace(50, 200, num = 8, dtype=int)
r_space = np.linspace(.5, 1, num = 5)
gamma_space = np.linspace(.1, 1, num = 5)
omega_space = np.linspace(.1, 1, num = 5)
k_space = np.linspace(10, 30, num = 3)


output_csv = pd.DataFrame(columns = ("N", "r", "gamma", "omega", "k", "
    L_avg_err", "I_avg_err", "S_avg_err", "A_avg_err"))

#Run the sweep!
for N in N_space:
    print(f"N space at {N}")
    for r in r_space:
        print(f"r space at {r}")
        for gamma in gamma_space:
            print(f"gamma space at {gamma}")
            for omega in omega_space:
                print(f"omega space at {omega}")
                for k in k_space:
                    print(f"k space at {k}")
                    print(f"r space at {r}")
                    print(f"N space at {N}")
                    print(f"gamma space at {gamma}")
                    print(f"omega space at {omega}")
```

```
#RUN THE SIMULATION AND COMPARE STUFF!
er_graph = ER_Graph(N, simulations, k,
    inital_conditions, r, gamma, omega)
er_graph.run_n_simulations(data_path, n=batch_size)



#Run MFT for the same
MFT = Rescaled_MFT(N, simulations, k,
    inital_conditions, r, gamma, omega)
MFT.solve_MFT()
MFT.concat_data()
MFT_df = MFT.df

#Compare each data peice to MFT
for batch in range(1, batch_size+1):
    params = {"N": N,
              "simulations": simulations,
                    "k": k,
                    #"P": self.P, -P is calculated
                        from k and N, and makes
                        naming worse, so it's
                        excluded
                    "inital_conditions":
                        inital_conditions,
                    "r": r,
                    "gamma": gamma,
                    "omega": omega,
                    "Simulation_batch_num": batch}
    file_name = str(params).replace(":","=").
        replace("'", "") + ".json"
    file_path = data_path + file_name
    er_graph_df = pd.read_json(file_path)

    #Convet er graph solutions to population
        proportions

    er_graph_df["L_sol"] = er_graph_df["L_sol"] / N
    er_graph_df["I_sol"] = er_graph_df["I_sol"] / N
    er_graph_df["S_sol"] = er_graph_df["S_sol"] / N
    er_graph_df["A_sol"] = er_graph_df["A_sol"] / N

    ##COMPUTE ERROR
    #Inner join er graph df and MFT df on Time/
```

```
    simulation
comparison_df = pd.merge(er_graph_df, MFT_df,
    how = "inner", left_on=("simulation"),
    right_on=("T"))

#calculate error
comparison_df["L_difference"] = comparison_df["
    L_sol"] - comparison_df["L_sol_MFT"]
comparison_df["I_difference"] = comparison_df["
    I_sol"] - comparison_df["I_sol_MFT"]
comparison_df["S_difference"] = comparison_df["
    S_sol"] - comparison_df["S_sol_MFT"]
comparison_df["A_difference"] = comparison_df["
    A_sol"] - comparison_df["A_sol_MFT"]

#compute average error
L_avg_error = comparison_df["L_difference"].
    mean()
I_avg_error = comparison_df["I_difference"].
    mean()
S_avg_error = comparison_df["S_difference"].
    mean()
A_avg_error = comparison_df["A_difference"].
    mean()

##PLOT THE THANG
if batch == 1:
    plt.plot(comparison_df["T"], comparison_df[
        "L_sol_MFT"], color = "blue", label = "
        Luddite")
    plt.plot(comparison_df["T"], comparison_df[
        "I_sol_MFT"], color = "green", label = "
        Ignorant")
    plt.plot(comparison_df["T"], comparison_df[
        "S_sol_MFT"], color = "orange", label =
        "Susceptible")
    plt.plot(comparison_df["T"], comparison_df[
        "A_sol_MFT"], color = "red", label = "
        Adopted")

    plt.xlabel("time (t)")
    plt.ylabel("Population Proportion")

    plt.legend()
    plt.title("MFT vs. Agent-Based Simulation")
```

```
                plt.plot(comparison_df["simulation"],
                    comparison_df["L_sol"], color = "blue",
                    alpha = 0.3)
                plt.plot(comparison_df["simulation"],
                    comparison_df["I_sol"], color = "green",
                    alpha = 0.3)
                plt.plot(comparison_df["simulation"],
                    comparison_df["S_sol"], color = "orange",
                    alpha = 0.3)
                plt.plot(comparison_df["simulation"],
                    comparison_df["A_sol"], color = "red", alpha
                     = 0.3)


                ##ADD TO OUTPUT CSV
                #Output a row of CSV Data
                output_csv.loc[len(output_csv)] = [N, r, gamma,
                     omega, k, L_avg_error, I_avg_error,
                    S_avg_error, A_avg_error]




                print(f"r space at {r}")

output_csv['avg_err'] = (abs(output_csv['L_avg_err']) + abs(output_csv[
    "I_avg_err"]) + abs(output_csv["S_avg_err"]) + abs(output_csv["
    A_avg_err"])) / 4
```