DIFFUSION IN HETEROGENEOUS NETWORKS

BY

FRANCISCO J. DOMINGUEZ SAGARENA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Yuliy Baryshnikov

ABSTRACT

We create a program to simulate diffusion in random graphs. Specifically, we create a generalization of *bootstrap percolation* which incorporates any number of societies λ and an arbitrary weight matrix $\mathbb{W}$. The code was created from the start to be as general as possible and to easily be modified with further complications. With it we simulate and analyze diffusion in networks consisting of one, two and three distinct societies with different parameters. We study the proportion of adoption Ω as a function of threshold θ, probability of initial activity α and connectivity P_c. We compare several different weight matrices in the two and three society cases to understand the effect of these changes on the diffusion process. The end result is a tool that we propose can be used with the aid of statistics in social or biological contexts to predict behaviors and make conjectures on scenarios not yet observed.

ACKNOWLEDGMENTS

The author wishes to express sincere gratitude to the Fulbright Association, without whose help my studies would not have been possible. Special thanks to Jeremy Goodsitt for his support and help in the polishing of the code used in this thesis. In addition, thanks to Victor Arnez for his help discussing a lot of the results and brainstorming possible simulation values, and Ivan Arnez for his creative help making some of the images. Thanks also to Professor Yuliy Baryshnikov who always accommodated the requirements of my studies in his busy schedule. And finally, thanks to my wife for her invaluable support through the sometimes taxing process of earning my master's degree.

TABLE OF CONTENTS

LIST OF SYMBOLS

G	Random graph with set of vertices V and set of edges E.
V	Set of vertices in a random graph G.
E	Set of edges on a random graph G.
d_v	Degree of node v; number of edges connecting to v.
$\vec{m}$	Population size vector, each entry m_i denotes the size of society i, also $m_i = \|V_i\|$.
$\mathbb{A}$	Adjacency matrix of the nodes in a random graph G.
θ	Threshold for which individuals in a network adopt the new behavior.
α	Probability of initial activity; this is the chance that a member of the society adopts the innovation at time zero.
A	Original behavior.
B	Innovation or new behavior.
λ	Number of societies interacting to make an overall social network.
$\vec{n}$	Population vector. It contains zeros for the individuals that have adopted the innovation and otherwise it contains ones.
q	Payoff parameter for game theoretic model.
d_i^x	Number of neighbors the node i has with behavior x.
$f_i(x)$	General functions that serve the purpose of threshold rules and weights in the *General Threshold Model*.
$g_i(j, X)$	General incremental functions that serve the purpose of threshold rules and weights in the *Cascade Model*.
$\mathbb{A}^I$	Internal adjacency matrix.
$\mathbb{A}^H$	Hybrid adjacency matrix.

${}_{\lambda}\mathbb{C}_{x}$	Combinations of λ in x.
$PopSym$	Symmetry of internal adjacency matrices.
P_c	Probability of connectedness. Parameter to determine how connected a given society is.
$PopSame$	Reciprocity in hybrid relations.
$\mathbb{W}$	Weight matrix.
r	Number of repetitions of each set of parameters that the program will run.
Ω	Proportion of adoption of the initial non-adopters.
μ_i	Initial number of non-adopters.
μ_f	Final number of non-adopters.
P_n	Probability of initial non-adoption. $P_n = 1 - \alpha$.
S_x	Society x.
θ_c	Critical threshold.

Flowcharts

$W[i, j]$	Entry (i, j) of weight matrix.
$A[i, j]$	Sub-matrix (i, j) of adjacency matrix. This refers to either an internal or hybrid adjacency matrix.
$n[j]$	Subset (j) of general population vector; this is the population vector for society j.
$m[i]$	Population size of society i.

CHAPTER 1

INTRODUCTION

When individuals are connected in a network it becomes possible for them to influence each other. This is seen in the spread of new ideas, decisions and behaviors throughout the population, thereby producing a collective outcome by the aggregate contributions of individuals.

These diffusion processes happen in practically every social setup and can be observed in many different aspects in which people are influenced by others: the spread of religious beliefs; the clothes people wear, and therefore fashion as a phenomenon; the proliferation of political ideas and its influence in policy making; the products bought and sold and their relationship with marketing and publicity; the adoption of new technologies in virtually every sector; the activities people pursue; the popularity of celebrities; the use of a specific social website; the emergence and implosion of financial market bubbles, and many other things. Even scenarios where no literal influence is taking place such as epidemics can be studied in the same context (see section 2.2.3).

All these situations share a basic set of qualitative properties, regardless of the specific scenario in which influence happens. They all begin with a relatively small set of early adopters which, in turn, influence their neighbors, friends and other relations, and in some cases manage to "convince" others to adopt this new behavior. An increasing number of people adopt this behavior by influence of previous adopters and so on. The result is, if successful, the contagious spread of the behavior through the network.

The creation of models to capture data on these diffusion processes is not only very helpful to aid our understanding at a fundamental level of how this phenomenon unfolds, but also as a predictive tool to know in advance the success or failure of a certain innovation or behavior. Based on the topology

of any given network, these models could also help us determine other key factors of diffusion in social networks (see for example [1]): key individuals with great influence capability, subsets of the network with the most impact, minimal and necessary conditions for maximum influence or "cascade", the speed of contagion and the reach of it, etc.

1.1 Diffusion Studies

Diffusion processes have been studied using many different approaches which have yielded several important applications. These processes were formally studied for the first time in a field known as *Diffusion of Innovations*; an area of sociology that developed in the mid-20th century. For a brief history of the field refer to [2].

General understanding of diffusion has grown immensely as mathematical tools develop (in the form of better models), and as bigger and more suitable environments for study arise. Some of these environments are large scale on-line communities, where the spread of ideas and cascading behavior can be studied in real time with population sizes never before studied, and with richer and more available data. Also, computer simulations based on both data and theory have formed a feedback loop with the rest of the tools, to advance the field and its understanding.

Because of this, research on diffusion has gone from anecdotal awareness to empirical studies, to models inspired by game-theoretic motivations, and finally rich mathematical models based on huge data samples. The book [2] provides a very comprehensive description, history and achievements of the field.

1.1.1 General model

One simple model that captures the essence of diffusion of behaviors is the following: let $G = (V, E)$ be a connected graph. Each node (or vertex) $v \in V$ in the graph represents an individual in the network. We say that given two

vertices i and j, $i \sim j$ if $(i,j) \in E$. This represents some kind of social relation between the individuals i and j. Let d_v be the degree of node v. Overall this graph G represents the entire social network. One example of such a network is shown in figure 1.1.

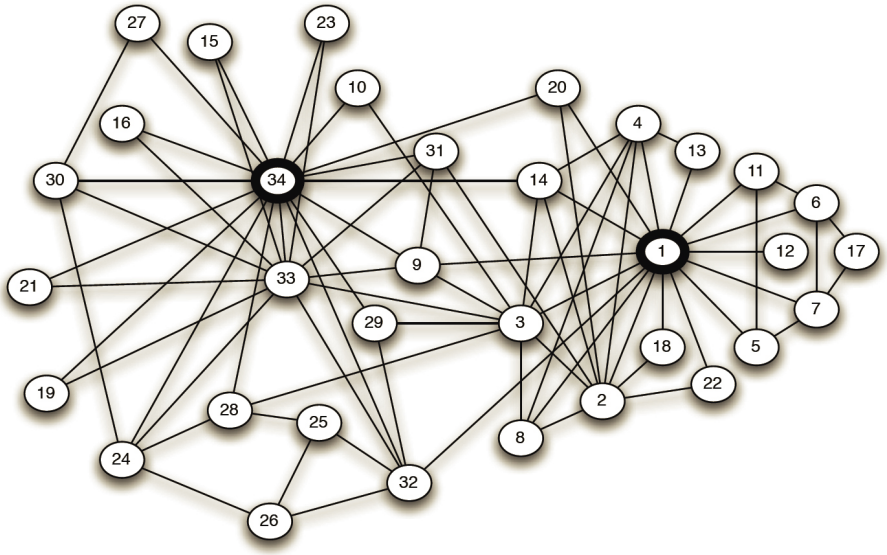

Figure 1.1: Social network representing relations in a karate club with 34 members. (Taken from page 2 of [3], originally from [4]. Added with permission of the Board of Regents of the University of New Mexico.)

The variables in this model are the population size: $m = |V|$, the nodes' degrees d_v (and therefore generally speaking the adjacency matrix of G: $\mathbb{A}$) and the following 2 parameters: $\theta : \mathbb{N} \to \mathbb{N}$ and $\alpha : \mathbb{N} \to [0,1]$.

We study the adoption of behaviors in some domain. Let us say that the original behavior is A and the new one is B (these behaviors can be referred to as inactive and active states as well). At the beginning of the process each node v adopts the new behavior B with probability $\alpha(d_v)$ independently from other vertices (to capture a realistic model, this probability is generally small so that the set of initial adopters is small). At subsequent times $t \in \mathbb{N}$ the state of each node i will be updated according to a deterministic process: if at time $t-1$ node i had behavior B (or was active), then it will remain active at time t. Otherwise, i will adopt behavior B (or become active) if at least $\theta(d_i)$ of its neighbors have adopted behavior B.

The parameter $\theta(d_v)$ is the "Threshold" of adoption; the number of neighbors that v requires to have in order to "switch" behaviors. The parameter

$\alpha(d_v)$ is related to the "Seed" population, or the initial subset of the network that are adopters.

In the case where both α and θ are constant, i.e. $\alpha(d) = \alpha$ and $\theta(d) = \theta$ $\forall\ d \in \mathbb{N}$, this model is called *Bootstrap Percolation* and it was first mentioned and studied in [5]; applications of this model in physics can be seen in [6]. Bootstrap Percolation in random graphs is studied in [7].

1.1.2 Generalizations

Several possible complications to the previous model can be made to enrich it and make it more realistic, perhaps to capture a specific scenario visualized in the world. This will be covered more deeply in section 2.2.2 but some examples are:

1. Relations between individuals might not be symmetric.
2. Weighted relations.
3. More than two possible behaviors.
4. Hybrid thresholds, etc.

1.2 Scope of the Research

In this study we focus on a generalization of Bootstrap Percolation that consists of the following 2 complications:

1. Populations with different parameters interact to form a collective social network that is not homogeneous. This means that we could have a collective network consisting of λ societies, each of them potentially having different values for connectivity d_i and seed probability α_i where $i \in [0, \lambda]$. Note that each population has homogeneous internal parameters. This could be interpreted not only as λ separate societies interacting but also as λ subsets of one single population. Therefore, from this point forward both scenarios will be referred to as societies.

2. Weight matrix. We can introduce weights in the relations between said populations to reflect different interests and influences between them. This way, for example, we can have a 3 society situation where society A impacts B γ times the impact it has on C for any arbitrary γ. This results in increased capability to adjust how meaningful each society is to the rest of them, and as a consequence we can indirectly adjust the threshold θ_i for different societies with respect to the others.

1.2.1 Unique aspects and limitations of the study

For this study a program was created in *Matlab* to simulate the diffusion process of Bootstrap Percolation with the two mentioned complications. The creation of such a tool and the application of it to these kind of scenarios is, as far as the author knows, unique. As far as limitations go, the following are the main ones:

1. The study was done in a qualitative way, analyzing the overall behavior of different scenarios. The best way to utilize this tool (or a tool like it), however, would be to relate this to statistical data retrieved in biological, sociological or other environments, to make models and predict behaviors similar to the ones observed. This is left for future studies.

2. Overall, the program could be adapted to incorporate any number of improvements, some of which are mentioned in sections 2.2.2 and 3.3.

1.3 Overview of the Thesis

Chapter 2 gives a brief theoretical background and discusses some basic models and studies in the field. Chapter 3 explains the code used to simulate the diffusion and all the specific parameters utilized in the study in general and in the program created; this chapter also discusses concepts and definitions used in the simulation and includes a diagram for the code. Chapter 4 presents the results of the research done which are introduced in section 4.1

(this section also describes the parameters examined and outputs collected in the different stages of the study); the results are discussed and interpreted as best as possible. Chapter 5 summarizes the results presented in chapter 4 and goes on to draw conclusions from them and makes suggestions for future studies. The appendices include all the code from the different programs created for the thesis.

CHAPTER 2

THEORY

2.1 Evolution Vector

If we take the general model from section 1.1.1, one of the obvious questions is: how do the nodes in the network evolve? We use a population vector $\vec{n}(t)$ to describe the state of each member of the social network at time t; $n_i(t) = 0$ if the node i is active at time t and $n_i(t) = 1$ otherwise. Therefore at time $t = 0$ the vector $\vec{n}(t)$ will have zero-entries only for the seeded population (the ones that were affected by α); in other words $\vec{n}(0)$ is a Bernoulli random variable with parameter α. Let $\mathbb{A}$ be the adjacency matrix of the social network; this means that $\mathbb{A}_{ij} = 1$ if $i \sim j$ (i and j are related) and $\mathbb{A}_{ij} = 0$ otherwise. The evolution of the vector $\vec{n}(t)$ is given by:

$$n_i(t+1) = n_i(t) + \big(1 - n_i(t)\big)\mathbb{1}\bigg(\sum_{j=1}^{n} \mathbb{A}_{ij} n_j(t) \geq \theta\bigg) \tag{2.1}$$

In the general case where we have a weight matrix affecting the interactions between society i and j then we have:

$$n_i(t+1) = n_i(t) + \big(1 - n_i(t)\big)\mathbb{1}\bigg(\sum_{j=1}^{n} \mathbb{W}_{ij}\mathbb{A}_{ij} n_j(t) \geq \theta\bigg) \tag{2.2}$$

2.2 Brief Theoretical Background

As was briefly mentioned in chapter 1, diffusion studies have made much progress in the past decades and have diversified into many different ap-

proaches and disciplines. In this chapter we introduce the reader to some studies and models created by the research done on Diffusion of Innovations and networks.

It is important to emphasize how interdisciplinary research on these topics is: it is done in the context of biology, sociology, marketing, economics, and mathematics, and it uses tools from statistics, graph theory, game theory, computer simulations and general mathematics. For example, a diffusion study with a small emphasis on marketing can be seen in [8],the work found on [9] empirically studies diffusion in social networks in the context of sociology and links this to phenomena such as riots, rumors, strikes, voting, migration etc. Other empirical studies can be seen in [10] (analysis of the diffusion of prescription drug) and [11].

2.2.1 Game theoretic model

Whenever we deal with a complex network we can divide its study into two parts: the understanding of the underlying structure and the links connecting it (generally studied in the context of graph theory) and also the interdependence of the adopted behaviors of the individuals inhabiting said network. The latter makes it so that decisions being made at an individual level in the network depend at least indirectly on the collective behavior of the rest of the network. This can be studied in the context of game theory.

To formulate a model using game theory we start with the same situation as we did in section 1.1.1: we have a connected graph $G = (V, E)$ with its nodes being members of some society, while edges in this graph represent a social interaction. Each node has the option to adopt one of two possible behaviors: A the original behavior and B the innovation. Given two nodes i and j with an edge $i \sim j$, there is a payoff if those two nodes match their behavior. Let the payoffs be as follows:

1. If both i and j adopt behavior A then they receive a payoff: q with $0 < q < 1$.

2. If both i and j adopt behavior B then they receive a payoff: $1 - q$ with $0 < q < 1$.

3. If i and j adopt different behaviors, they receive a payoff of 0.

This model is based on work found on [12] and can also be seen in [13] and in chapter 19 of [3].

Say now that an arbitrary node i is trying to select a behavior to adopt given that all other nodes have already picked one. If we have that the degree of node i is d_i and i has d_i^A of its neighbors with behavior A and d_i^B of its neighbors with behavior B, then the payoff node i gets from adopting behavior A is qd_i^A while the payoff for adopting B is $(1-q)d_i^B$. Suppose that $qd_i^A > (1-q)d_i^B$, since $d_i = d_i^A + d_i^B$ then node i should adopt behavior A if $qd_i > d_i^B$, and similarly i should adopt B if $qd_i < d_i^B$. To manage when we have an equality we say that i also adopts B if $qd_i = d_i^B$.

We can see that even though we started with a game theoretic model working with payoffs and decision making from each of our nodes, in the end we are dealing with a threshold q just as we did with θ in section 1.1.1; node i adopts the innovation B if at least a q fraction of its relations have done so at any step of the diffusion.

Note that in this model there is nothing stopping any node i from reversing its behavior from B to A. The case where we allow reversals or deactivations is called *non-progressive*, and the case where it is defined that once a node i adopts the innovation it does so forever is called the *progressive* case (see [13]). When studying which initial subsets of the network cause a global cascade or total adoption, it is important to ask whether or not it is easier for an innovation to spread in the *progressive* case. The *contagion threshold* of a social network is defined as the maximum value of threshold θ for which there exists a finite contagious set (this is a subset of the network that causes a global cascade or total adoption in the network when it is given the innovation at time zero). It is interesting to see that for any graph G the progressive and non-progressive models have the same contagion threshold, as can be seen in [12].

2.2.2 Other mathematical models

We have already presented a general mathematical model of diffusion in section 1.1.1. In this model all individuals in the network use the same threshold θ, and all neighbors have equal weight in influencing another node to reach said θ. Several other mathematical models exist:

1. *Linear Threshold Model*: directed graphs are used to have edges pointing in both directions between two nodes. Also, a non-negative weight is introduced in each of these directed edges to account for how much a certain node "matters" to another one. Finally each node i has its own threshold picked at random from a uniform distribution.

   This is done because in a real world setting relations might not be symmetric; even when there is a relation between node i and j, i.e. $i \sim j$, it could be the case that i influences j, but not the other way around. This can happen for example when opinions from celebrities influence followers, but not necessarily in reverse.

   More generally, the influence exerted by node i over j could not be the same as that exerted by node j over i. This is where weights matter; certain individuals could be more influential than others and the addition of this tool helps us model that.

   The importance of "influentials" and its comparison with easily influenced individuals can be further studied in [1]. In [9] it is discussed how the influence between individuals depends on the relationship they have. An analysis of the final proportion of adoption using random networks and the linear threshold model can be seen in [8].

2. *General Threshold Model*: In this model, hybrid threshold rules are introduced where if we separate the social networks into various groups (for example two different cities), we can create hybrid thresholds to adopt the new behavior where x number of neighbors from city χ, and y number of neighbors from city ψ, are required to adopt the new behavior. In this model each node i has function $f_i(x)$ with a domain equal to the neighbors of i. Since now we have general functions $f_i(x)$ serving the purpose of the weighted sum of thresholds we had before in the Linear Threshold Model, we now have a very general case that could

adapt to virtually any threshold rule, including hybrid thresholds.

3. *Cascade Model*: This model is equivalent to the General Threshold Model (see [13]). The basic difference is the interpretation and setup. Here we make the nodes adopt a behavior probabilistically, and therefore the model captures the notion of contagion more directly. In each step of the diffusion process, for each edge in the network where we have 2 nodes i and j connected such that j is active and i is not, there is a chance that j influences i. The probability of success depends not only on i and j but also on the set of nodes that have already tried and failed to activate i. Instead of using functions f_i, each node now has an *incremental function* $g_i(j, X)$ where j is a neighbor of i and X is a set of neighbors of i not containing j. The function $g_i(j, X)$ is the probability that the node j will activate i given that the entire set X has already tried and failed. This type of model is more intuitive to apply to situations where decision making is either not the key factor or even irrelevant, and contagion happens involuntarily.

These models are further studied in [13]. We could also have variations of all these models incorporating either the progressive or the non-progressive case. The latter, for example, would fit models trying to understand creation and implosion of financial bubbles among other things. Furthermore, all of these models could be modified to make use of n behaviors instead of only two, which would have applications and give insight in situations where competing businesses or technologies influence a social network.

2.2.3 Applications

The context of on-line communities and social networks provides very rich environments to study these kinds of processes; not only is the population size generally very big, but also, it is easy to monitor, data can be seen both in real time and in the span of months and influence happens constantly over a wide array of behaviors. As a first example we can see figure 2.1 where the network visualization of document exchange in a committee is displayed. Availability of this kind of data is immense in this and many other situations and when studied over time can give clear evidence and insight of diffusion

processes.

Figure 2.1: Network visualization of the ICIC (International Committee on Intellectual Cooperation) archives. Nodes are individuals (plenary committee, experts, etc.) and edges are documents exchanged. (Taken from https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png.)

One can study, for example, the probability of adopting behaviors as a function of the number of friends or relations that have done so previously. Figure 2.2 has this exact situation for an on-line blogging site called *LiveJournal*. We can see from figure 2.2 that the probability of joining a particular community increases with the number of neighbors who have already joined: k. Also we can see diminishing returns in this plot; after a certain number of friends, the probability does increase for every friend added but in a much more subtle way. We can also see that the first friends have a very pronounced effect; the probability of joining said community more than doubles if one has two friends compared to only one.

We can even study this kind of dynamic in biological scenarios. Cascading behavior and influence in general in the context of a social network is sometimes called "social contagion" because an innovation spreads from one

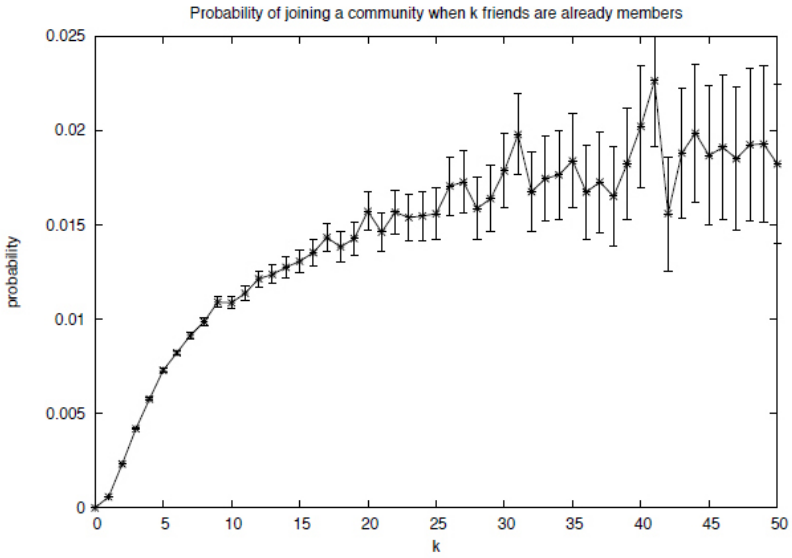

Figure 2.2: Probability of joining a *LiveJournal* community given that k members have already joined. (Taken from page 93 of [3], originally from [14]. Added with permission of the Association for Computing Machinery.)

person to another with the same dynamics of an epidemic in a biological context. Even though the fundamental mechanisms by which biological and social contagion happen are profoundly different (social contagion is linked to decision making and preference while biological is based in probability of being infected by a pathogen), their underlying network dynamics are very similar. This causes a lot of cross-discipline studies and interesting research questions.

In figure 2.3 we can see individuals in the context of a tuberculosis outbreak being infected. Black nodes are individuals with clinical disease, pink nodes indicate people exposed to the pathogen and with dormant or incubating infection, green nodes represent exposed people with no infection and finally gray represent close members to the rest of the network that have not been evaluated by medical personnel. This image displays clearly how the disease is being spread through the network.

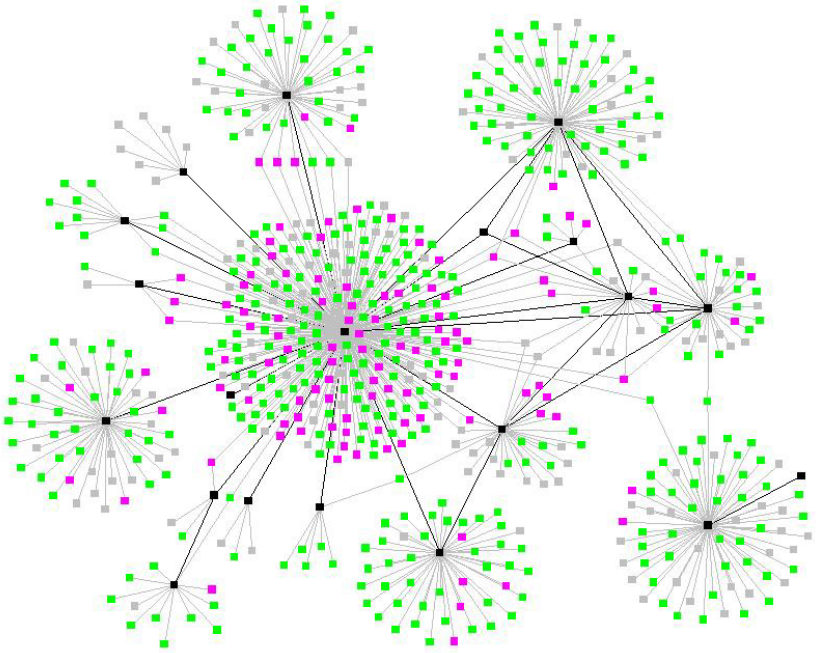

Figure 2.3: Spread of tuberculosis in a network. Colors indicate different degrees of infection. (Taken from page 15 of [3], originally from [15]. Added with permission of the American Public Health Association.)

CHAPTER 3

CODE

The entire code was written in *Matlab* and commented to be as descriptive as possible in regards to the function and use of each part of the code. It is added as an Appendix A and B and uploaded to https://github.com/F-Dominguez/Diffusion-Matlab-Code.

Effort was made to make the overall code as modular as possible, and therefore it has one main file and several nested functions do all the different parts of the process. All the files were commented extensively in order to explain everything as best as possible.

3.1 Required Variables

We start by making the following definitions:

1. An "Internal" adjacency matrix $\mathbb{A}^I$ is the one defined for the relations that happen between 2 members of the same society. In the general case of having λ societies, these are exclusively the ones of the form: $\{\mathbb{A}_{ii} \; with \; i \; \in [1, \lambda]\}$. These matrices can be symmetrical and that would represent that in every case where an individual i has a relation with j, then j has a relation with i or $\{if \; i \sim j \iff j \sim i\}$.

2. A "Hybrid" adjacency matrix $\mathbb{A}^H$ is the one that defines the relationship between members of different societies. Hence, these are the ones of the form: $\{\mathbb{A}_{ij} \; with \; i, j \; \in [1, \lambda] \; and \; i \neq j\}$.

3. A "General" adjacency matrix is the one composed of all the adjacency matrices in the entire network (all the internal and hybrid ones). Therefore, in the case of λ societies we have a general $\mathbb{A}$ composed of

λ^2 submatrices: λ internal ones and ${}_{\lambda}\mathbb{C}_2$ (or $\lambda^2 - \lambda$) hybrid ones.

In the main file (Percolation.m) one has to input all the different parameters for the simulation and the code in turn feeds that information to all the other functions. The relevant parameters are:

1. λ: The number of different societies interacting in the overall social network. Each society will have its own parameters, including an adjacency matrix.

2. $\vec{m}$: Vector with population size entries, where m_i is the population size of society i.

3. α: The probability of initial activity (or initial adoption of new behavior). This probability is defined for every society in the simulation and has to be reported in the range to be studied, i.e. we need initial and final values for every α_i with $i \in [1, \lambda]$, and the increment to take in between. There is a total of λ different α values to be reported. Similarly we can define $P_n = 1 - \alpha$ as the probability of initial non-adoption.

4. *PopSym*: Matrix of order λ stating whether each of the individual submatrices of the general adjacency matrix is symmetrical or not. It is important to note that the hybrid matrices should not be symmetrical for 2 reasons:

   - For a hybrid adjacency matrix $\mathbb{A}_{ij}^H$ to be square it is required that society i and j have the same population size; $m_i = m_j = m$. Since this does not necessarily happen, then $\mathbb{A}_{ij}^H$ can't be symmetrical.

   - Even in the case where the $\mathbb{A}_{ij}^H$ are square, it makes no sense to have hybrid adjacency matrices be symmetrical. This would have no application to model any real situation for it would represent the following example:

     – Let S_1 and S_2 represent society 1 and 2 respectively. If we make $\mathbb{A}_{12}$ symmetrical then we force that if $v_i^1 \sim v_j^2 \iff v_j^1 \sim v_i^2$ for $v_i^1, v_j^1 \in S_1$ and $v_i^2, v_j^2 \in S_2$.

     In other words, we are forcing relationships across both societies for different individuals based solely on their arbitrary position in the network.

Therefore this value is only really used to determine if the internal matrices $\{\mathbb{A}_{ii} \; \forall \; i \; \in \; [1, \lambda]\}$ are symmetrical.

5. P_c: the probability of connectedness. This is the probability that an arbitrary pair of nodes (i, j) is connected, and is used to determine the edges $\forall \; i \; \in \; G$ (giving us the degree d_i of each node i).

   The program uses $P_{c_{ij}}$ to generate the matrix $\mathbb{A}_{ij} \; \forall \; i, j \; \in \; G$ and then unifies them all in the general adjacency matrix $\mathbb{A}$. Thus we need values of $P_{c_{ij}}$ to determine the connectedness of both internal and hybrid relations.

   For the general case, we need a total of λ^2 values of $P_{c_{ij}}$. This only happens when $\mathbb{A}_{ij} \neq \mathbb{A}_{ji}$ (see PopSame below), otherwise, only $\lambda^2 - {}_{\lambda}\mathbb{C}_2$ values are needed. Each value of $P_{c_{ij}}$ needs to be input as a range (initial and final values) with increments.

6. *PopSame*: Reciprocity in hybrid relations. This vector of size ${}_{\lambda}\mathbb{C}_2$ states if the adjacency matrix $\mathbb{A}_{ij}$ is the same as $\mathbb{A}_{ji}$. In other words, it states whether or not societies are reciprocal in their relations.

   In addition to help us distinguish between two cases that are intrinsically different and have real world applications, this gives us the option to save considerable computing power.

7. θ: the threshold of each individual population. This has to be reported as a range with increments.

8. $\mathbb{W}$: Weight matrix. This is multiplied by the adjacency matrix and helps us introduce different degrees of influence in the social networks. The full matrix requires λ^2 values. With the aid of this entity we can adjust how important for society i is the behavior of society j for any $i, j \; \in \; [1, \lambda]$. In other words, we can alter the utility function that each society has for itself and the rest of the societies.

   This allows us to study various interesting scenarios as we will see later.

9. r: number of repetitions of each set of parameters. Since the process is random, we need to get an average of the proportion of adoption in order to get stable data; otherwise, it would change every single time we ran the program and no possible applications could occur from the

results we have. It was found that a value greater than five is necessary to get rid of very irregular data. The results presented in this thesis were run with an r of at least 30.

The observed variable in this code is Ω defined as the proportion of the initial non-adopters that are influenced at the end of the process. Let μ_i be the initial number of non-adopters, i.e. the total number of nodes in the network that possess the initial behavior (A) at time $t = 0$, and let μ_f be the final number of non-adopters. Therefore, to obtain the proportion of adoption we use the following equation:

$$\Omega = 1 - \frac{\mu_i}{\mu_f} \tag{3.1}$$

Thus, we get $\Omega = 1$ when all the initial non-adopters eventually convert and $\Omega = 0$ when no conversion happens at all. Since we are using μ and not some other variable counting number of adopters, it is convenient to use P_n as the probability of initial non-adoption instead of α.

Another output of the program for every set of parameters is the standard deviation associated with each average Ω depending on the number of repetitions r performed to obtain it.

3.2 Overview

The main file, *Percolation*, takes all the data and feeds it to different functions; a diagram of the pseudocode of the file *Percolation* can be seen in figure 3.1. The following is a brief description of the use of each one of these. For further information we refer the reader to the Appendices A and B where the full code is added with very detailed comments.

1. *Variation Perm*: This function is used to transform the matrices with all the values for P_c and α for all the societies in question, into a matrix containing all the possible permutations of values for these two parameters respectively. This is then fed to the rest of the program.

2. *Init Pop*: This function generates the vector of population and Adja-

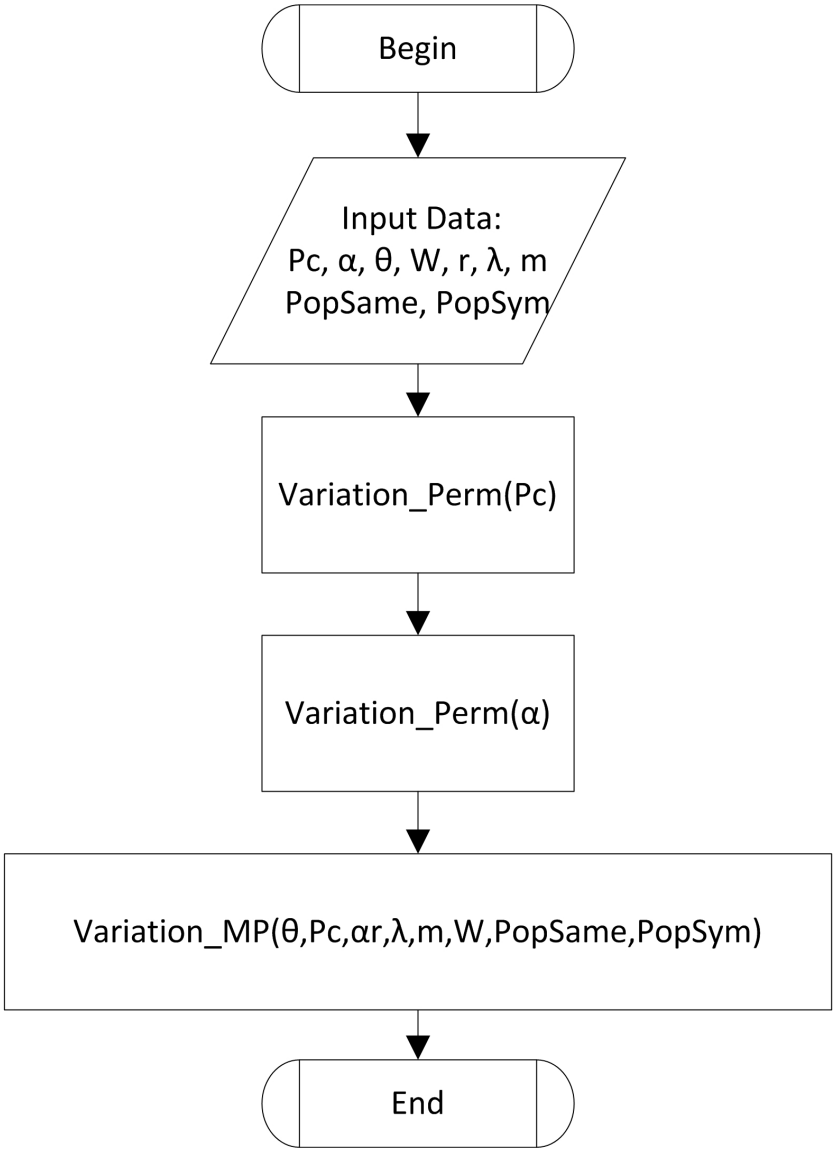

Figure 3.1: Flowchart describing the process of Percolation.m.

cency matrix.

- The population vector $\vec{n}$ is created using the population size, number of societies, and the different values of α. The end result is a vector with 0's for active new behavior (according to the probability of initial activity α) and 1's for original behavior, and has dimension equal to the total population size.
- The adjacency matrix is generated using the values of P_c for all the different submatrices. This part of the code takes into account

PopSym and *PopSame*.

3. *Variation MP*: This function is the main nested "for" loop. It calls *Init Pop* to generate the $\mathbb{A}$ and $\vec{n}$, and then feeds them into *Evolve MPopulation* where the diffusion takes place. It does this for all possible permutations calculated previously, and then saves the final proportion of adoption. A pseudocode flow chart can be seen in figure 3.2.

4. *Evolve MPopulation*: This is the core of the diffusion or cascade process. It applies the rule to activate if more than θ neighbors are activated. The program stops if, at any given cycle, no new members adopted the new behavior. This is separated for all the possible different cases of weight being applied to the diffusion. A pseudocode flow chart can be seen in figure 3.3.

Other than these files, which comprise the main program, we also have scripts to make any number of plots in two and three dimensions (*Graph2D* and *Graph3D*) and a script to make movies out of the plots obtained so that we can better visualize the evolution of the behavior. In order for all of these codes to get the required data out of the general array, *Find Perm Data* is used; this is incredibly useful to adapt the data extraction for a general case and it saves us a lot of time. We also have a graphical user interface and timebar code to estimate the time remaining until program completion.

Finally we have a *Visualization* code which is independent of the main program and is used to make a simulation of how the behavior spreads by allowing us to see how all the nodes switch from original to new behavior.

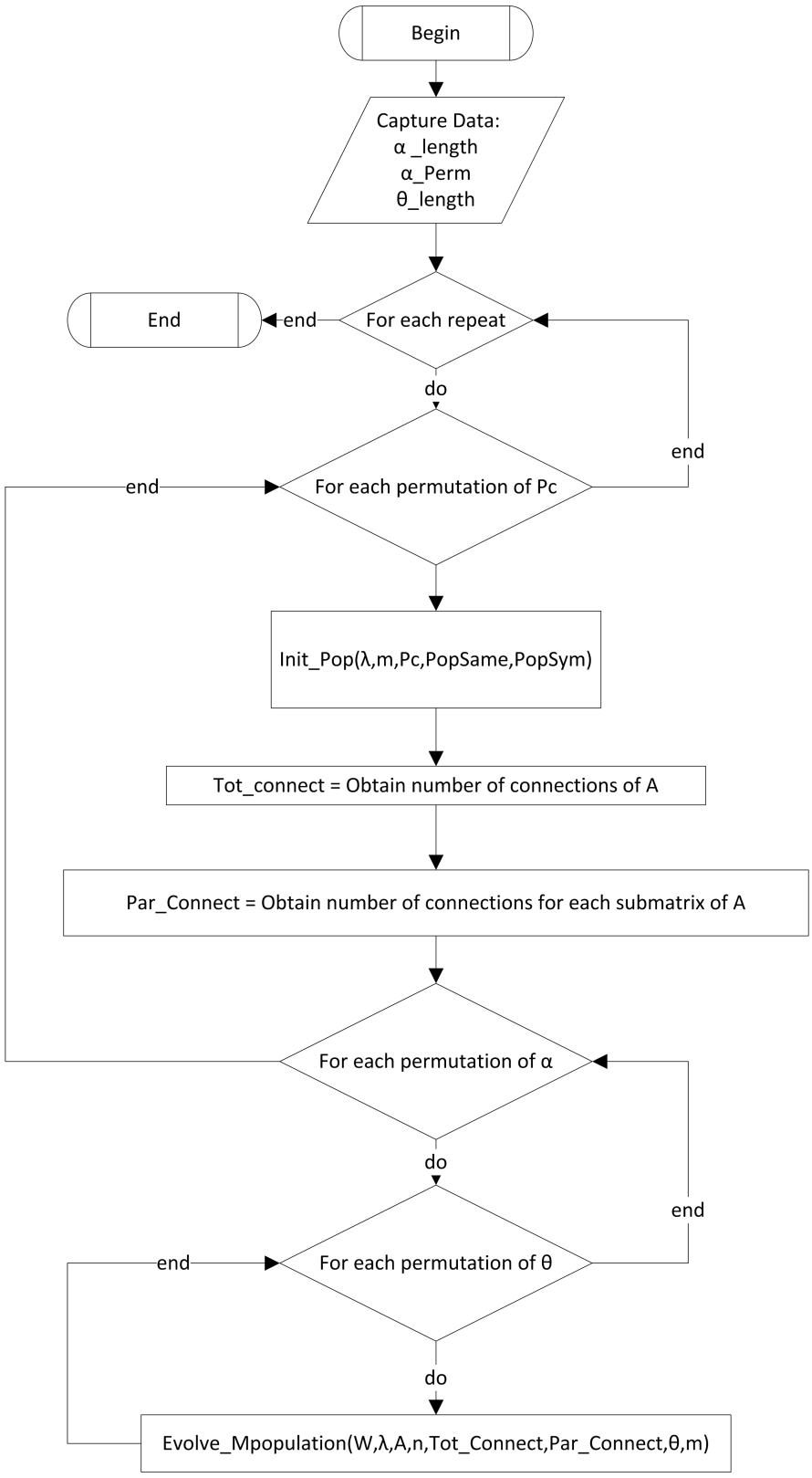

Figure 3.2: Pseudocode flowchart of Variation MP.m.

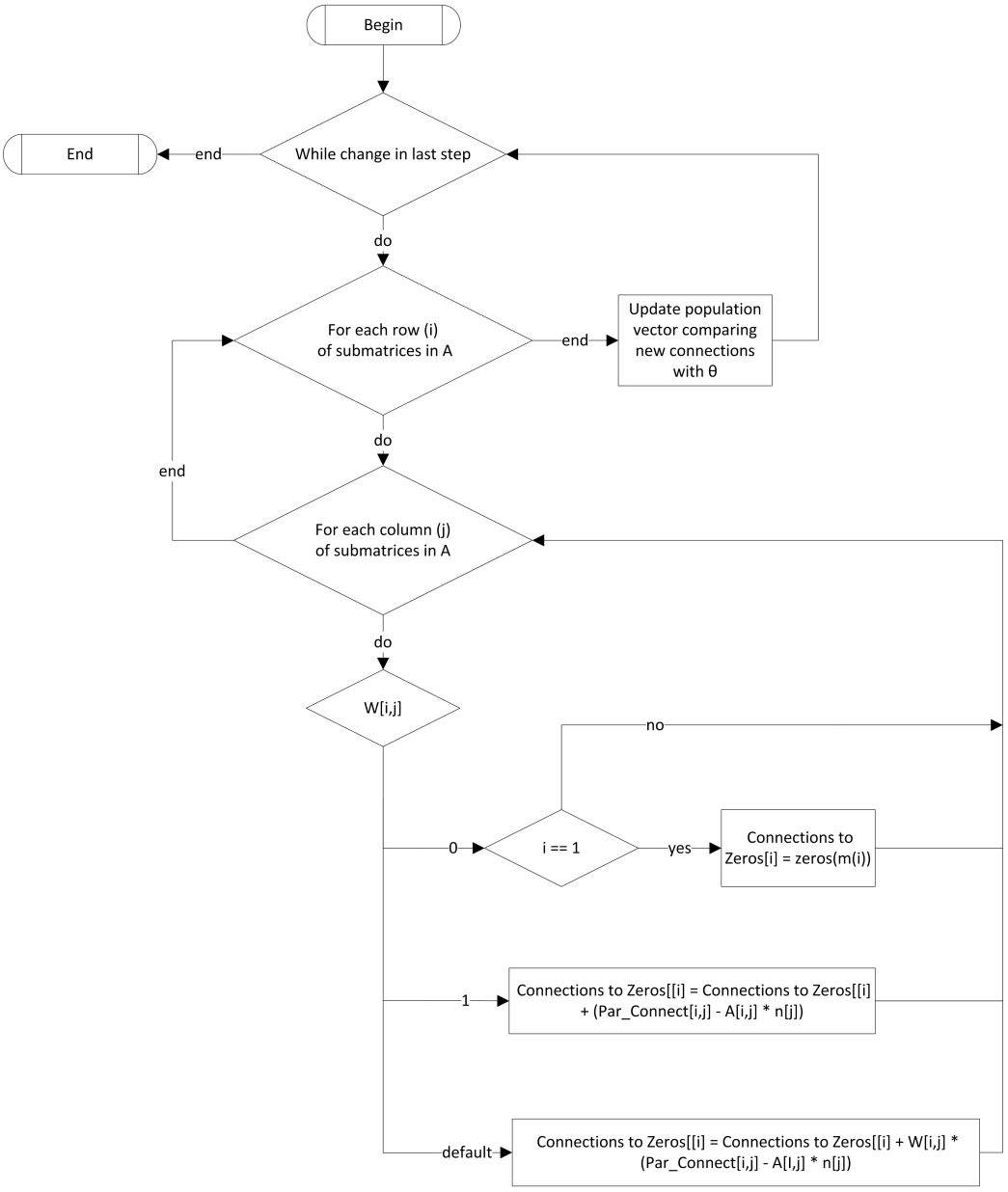

Figure 3.3: Pseudocode flowchart of Evolve MPopulation.m.

3.3 Possible Improvements

The program could be improved to support additional model parameters to explore diffusion phenomena. Some examples are:

1. *Hybrid thresholds.* We could adapt the program to study diffusion that occurs when more complicated threshold rules arise. For example: only adopt new behaviors when n neighbors of society S_1 and l neighbors of society S_2 have also adopted. This can be achieved to some degree with the use of the weight matrix $\mathbb{W}$ currently incorporated, but true hybrid thresholds would add complexity.

2. *Percentage threshold.* An easy fix. This would change the model, from one where x neighbors are required to influence the new behavior to one where $(X\%)$ of the total relations are required.

3. *Automatic variation of weight and population size.* This could allow us to get visualizations of the behavior of these diffusion processes with the variation of either of these parameters.

4. *Non-progressive case.* At the moment the program works progressively; once a node has adopted the innovation, it never goes back to the original behavior. Having the option to also study the non-progressive case would allow us to use this tool for a wider variety of cases.

5. *Multiple thresholds.* We could add complexity to the amount of different thresholds incorporated into the simulation. This could mean each society having a different threshold, individual randomized thresholds for each node in the global social network, or for example simply having two thresholds in the overall network representing the difference between "early-adopters" (or enthusiasts) and "late adopters."

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Development Process

In order to create a tool that is able to simulate the generalization of bootstrap percolation that concerns this study (mentioned in section 1.2), we started with a simple code that simulates the diffusion in a lattice network with only four geographical neighbors in total. This has certain applications but also allowed us to visualize the diffusion for the first time. After this we created a modified version that did a simulation for a case of one society with a general adjacency matrix, which is more related to the objective set out by this study. We then created the general program which simulates most of the work presented in this thesis. It is capable of getting the average proportion of adoption Ω for any number of cases of repeat r, connectivity P_c, initial activity α and weight matrix $\mathbb{W}$. This tool is meant to provide intuition about the influence of all these parameters in the diffusion process.

We begin analyzing a one society case by introducing visual simulations of the diffusion processes in a case with a lattice network, and then a general adjacency matrix. The proportion of adoption Ω as a function of θ and α is then studied. We also observe the impact of the number of repetitions in the reported standard deviation. To finish the study of one society we analyze how varying P_c affects the results and end with three-dimensional plots of the data. This is followed by the two society case where we study several different cases of weight matrices $\mathbb{W}$. Finally we compare two different weights for a three society case.

In the following sections an effort is made to present the results in increasing complexity. Therefore, we start with simple visual representations of the diffusion process in a single population and end with weight adjustments in

multiple societies.

4.2 One Society

With the one society case we aim to get some intuition and basic understanding of how the process of diffusion unfolds, so we can study more complicated cases.

4.2.1 Visualizations

In this section we will obtain a visualization of how the diffusion process occurs for two distinct models. We begin with a simple diffusion process. We generate a *Finite Lattice Network* (as seen in figure 4.1) $G = (V, E)$ with population size $m = |V|$; in a lattice network the edges are exclusively between first neighbors. In general infinite lattice networks have a fixed degree for all nodes in the graph of $\{d_v = 4\}$; if we have a finite lattice network, nodes will have degrees of two, three or four, depending on whether they are situated in corners, in the edge of the graph or if they are internal. In this case we have a finite lattice network but edges in the top are connected to the bottom and the right is connected to the left making it so that $\{d_v = 4 \ \forall \ v \in V\}$. See figure 4.2.

As previously explained, the society starts with an initial seed population determined by α, but in this simpler scenario, no random generation of adjacency matrix is required and therefore no average connectivity or P_c is needed.

We run the diffusion process for the following parameters:

1. One society, i.e. $\lambda = 1$.
2. Population size $m = 1,000,000$, represented as a 1000×1000 lattice.
3. Threshold $\theta = 3$.
4. Probability of initial activity $\alpha = 0.05$.

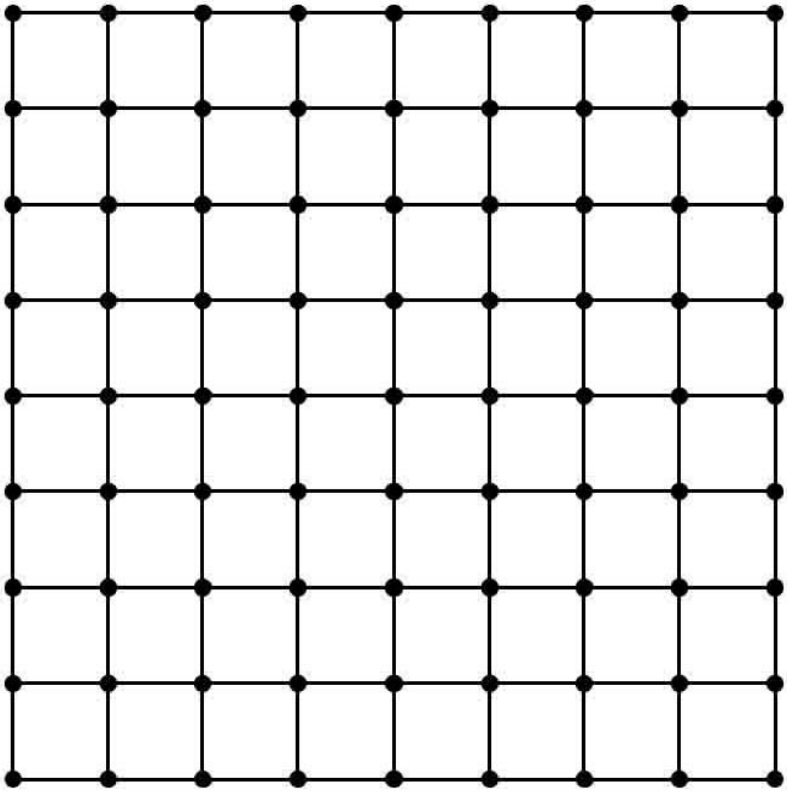

Figure 4.1: Finite lattice network.

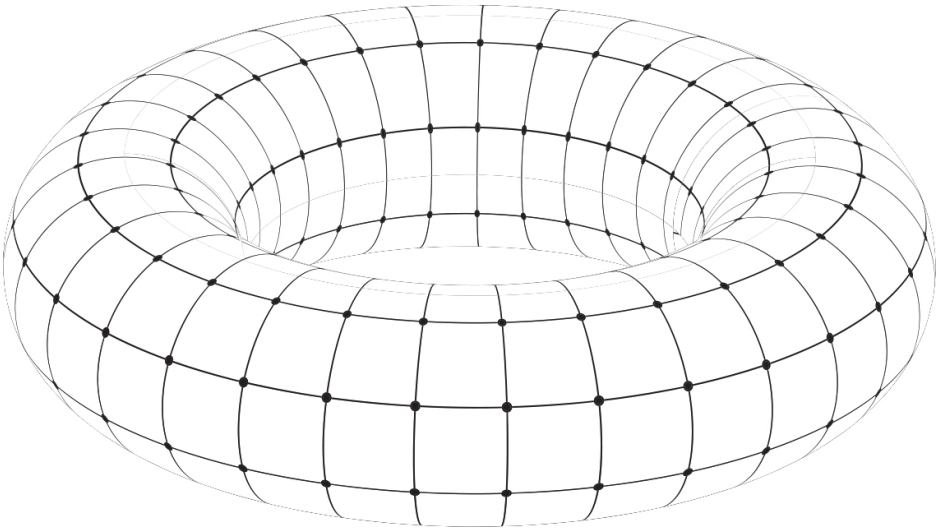

Figure 4.2: Finite lattice network when united in the extremes of graph.

The result of the diffusion process is captured in the images contained in figure 4.3. These have black pixels to represent individuals in the network with behavior A (original behavior) and white pixels to represent those with

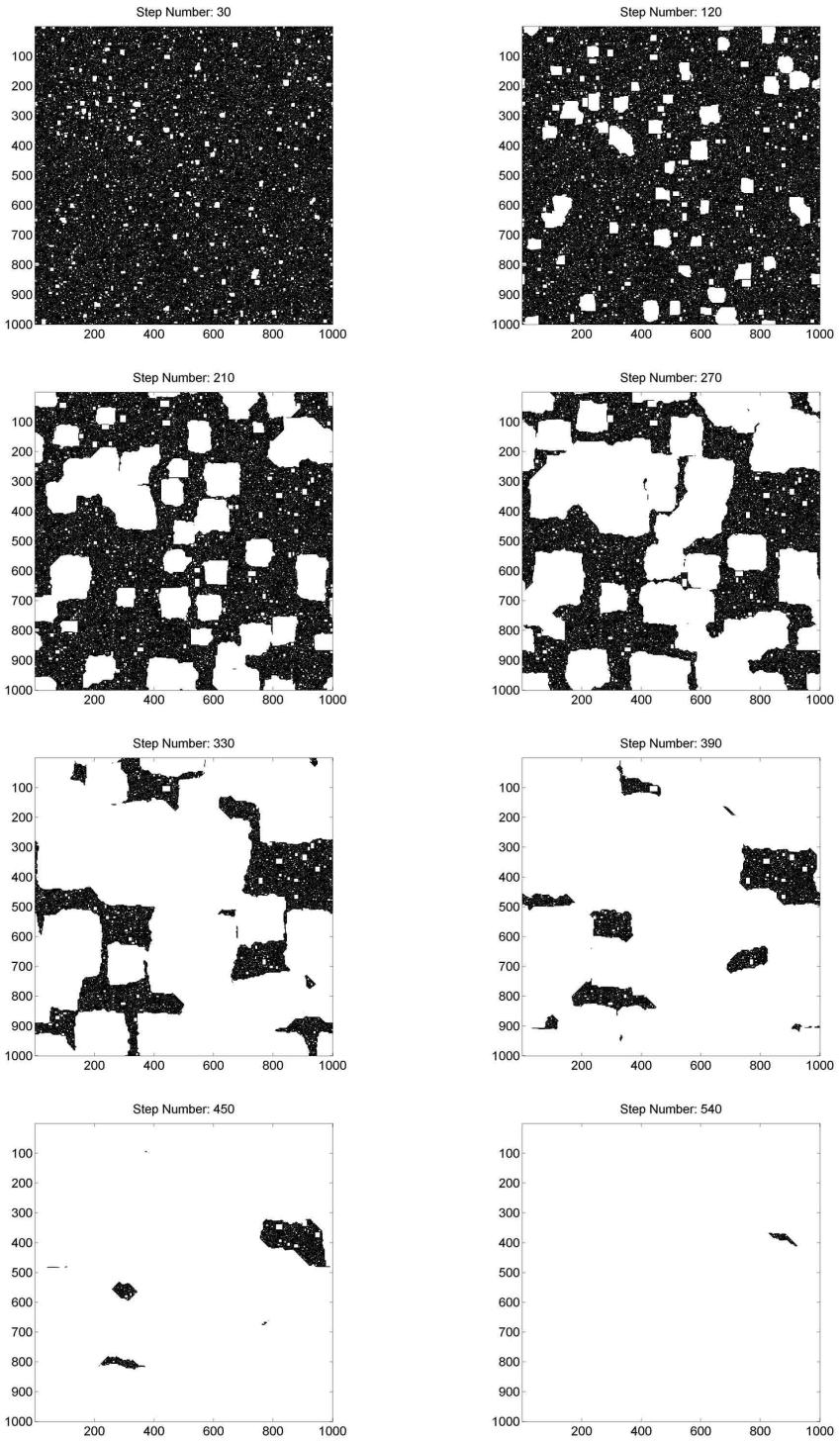

Figure 4.3: Visualization of diffusion in a Lattice network. $\lambda = 1$, $m = 1x10^6$, $\theta = 3$, $\alpha = 0.05$.

behavior B. Whenever a node has more than 3 of its neighbors with behav-

ior B it switches to B. We can see how given this set of parameters, the innovation spreads contagiously.

Models like these are especially useful to represent phenomena where geographical positioning is a fundamental aspect of the diffusion; in these cases neighbors take a literal meaning, for example, bacteria growth processes, contagion of disease, etc. For other applications we need a model that can capture any arbitrary number of relations between individuals, even when the nodes' position in a network is irrelevant or unknown.

For this we generate an adjacency matrix that contains all the relations of members in a social network. To visualize it, we simulate the diffusion with the following parameters:

1. One society, i.e. $\lambda = 1$.
2. Population size $m = 10,000$, represented as a 100×100 lattice.
3. Threshold $\theta = 3$.
4. Probability of initial activity $\alpha = 0.10$.
5. Probability of connectedness $P_c = 0.06\%$ (6 neighbors per node on average).

The results can be seen in figure 4.4. The difference is immediately noted, since the diffusion is occurring at the same time in the entire network. Positioning is now irrelevant; this model could represent a more general social network where influence is happening over several different media, and therefore not bound by adjacency.

In figure 4.4 the last image (step 74) is the final stage of the process; no changes happen after this. We can see that some nodes in the network are never affected in this situation. This makes intuitive sense since a subgroup of the entire network could be isolated from members of behavior B.

In both figures 4.3 and 4.4 we can see that at first, when the amount of nodes "infected" is low, the diffusion happens slowly, while in the last stages, the spread is much faster.

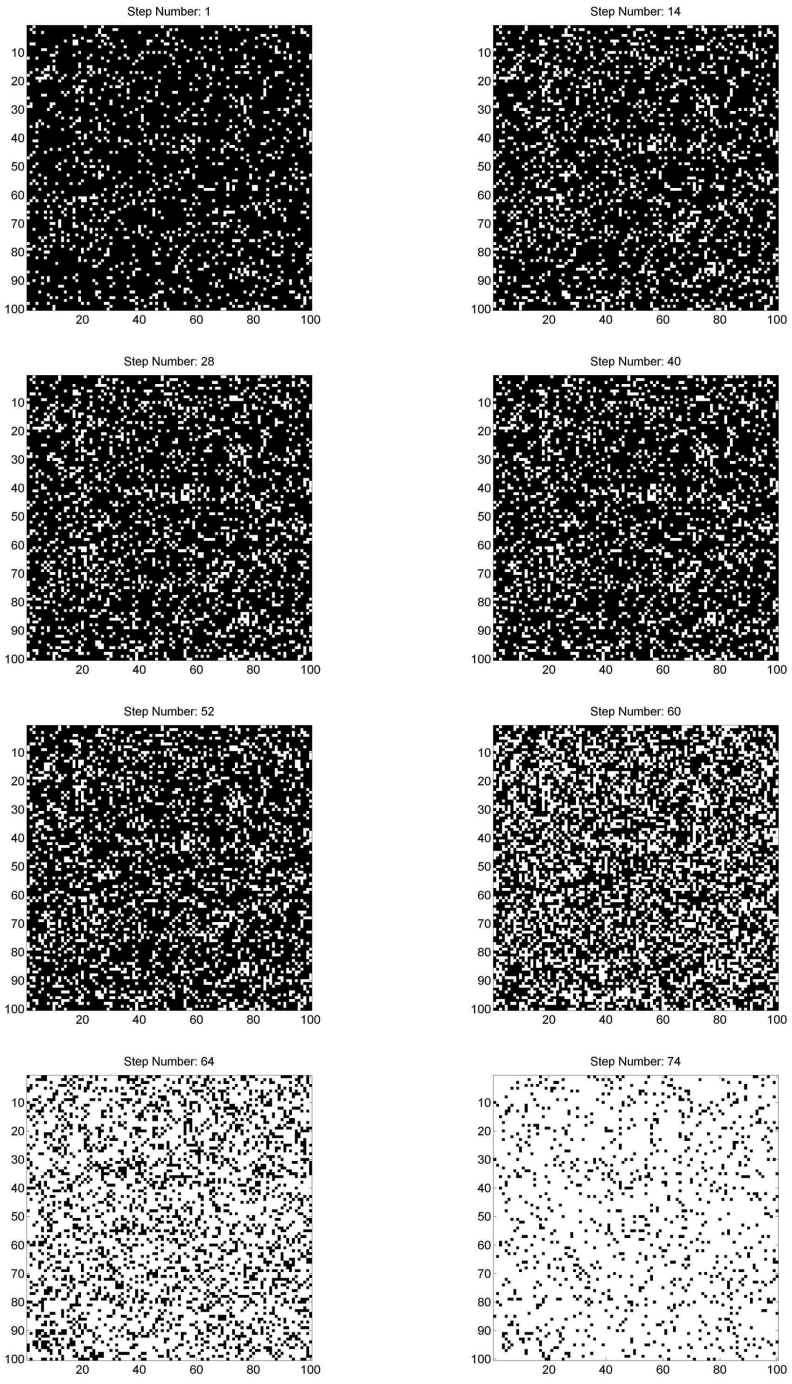

Figure 4.4: Visualization of diffusion in a random Network. $\lambda = 1$, $m = 1x10^4$, $\theta = 3$, $\alpha = 0.10$, $P_c = 0.06\%$.

4.2.2 Proportion of adoption vs. threshold (Ω vs. θ)

In the previous section we visualized how a behavior is spread through the network for a unique set of parameters. We did this for two different models; the first model is one with geographical neighbors more suited for biological scenarios for example, while the second one is Bootstrap Percolation and had a general adjacency matrix to represent the relations in the network. From now on, we will deal exclusively with the latter and the generalization explained in section 1.2. We also saw how a subset of the network could be isolated and therefore never influenced by the innovation. To study the amount of people converted to the new behavior we use the *proportion of adoption*: Ω. Using this, we can now study how Ω is affected by variation in θ, as we can see in figure 4.5.

Since the creation of both the adjacency matrix A and the population vector $\vec{n}$ is random, the process in itself will be random and with huge variance; therefore, we need to repeat the experiment several times to obtain an average and get some idea of the distribution. The error bars in 4.5, which are calculated with the standard deviation of the total number of repetitions done for each data set, are understandably big given the random nature of the experiment. Note that figure 4.5 was obtained with a *repeat* (r) value of 30; in contrast, figure 4.6 was done with 600 repetitions for each data set. Given that very little distinction is found between the two results, and to save computation time, all results from now on are produced with a value of around $r = 30$.

From figures 4.5 and 4.6 we can see that the contagion model behaves in a simple way; below certain threshold θ_1 (49 in the mentioned figures) the contagion is absolute and above a threshold θ_2 (53 in this case) essentially no diffusion is seen. Values in between these thresholds have a very heavy growth, making the overall function highly non-linear. This non-linearity plays a very important role in the study of diffusion of innovations in general and is studied further in [16].

This same non-linearity can be seen when we analyze what happens with the contagion as a function of α, i.e. Ω *vs.* α in figure 4.7

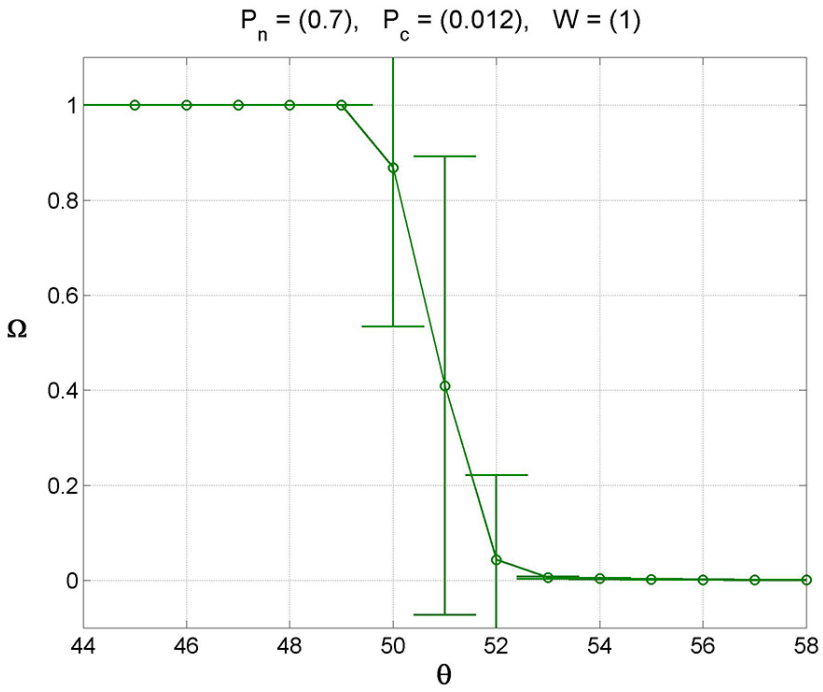

Figure 4.5: Ω *vs.* θ. $\lambda = 1,\ m = 1x10^4,\ \alpha = 0.30,\ P_c = 1.2\%,\ r = 30.$

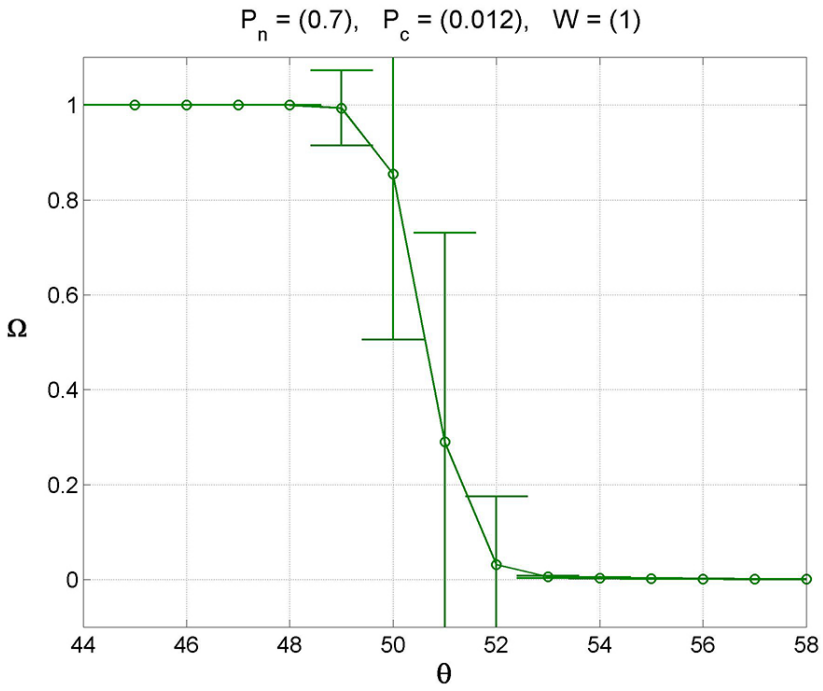

Figure 4.6: Ω *vs.* θ. $\lambda = 1,\ m = 1x10^4,\ \alpha = 0.30,\ P_c = 1.2\%,\ r = 600.$

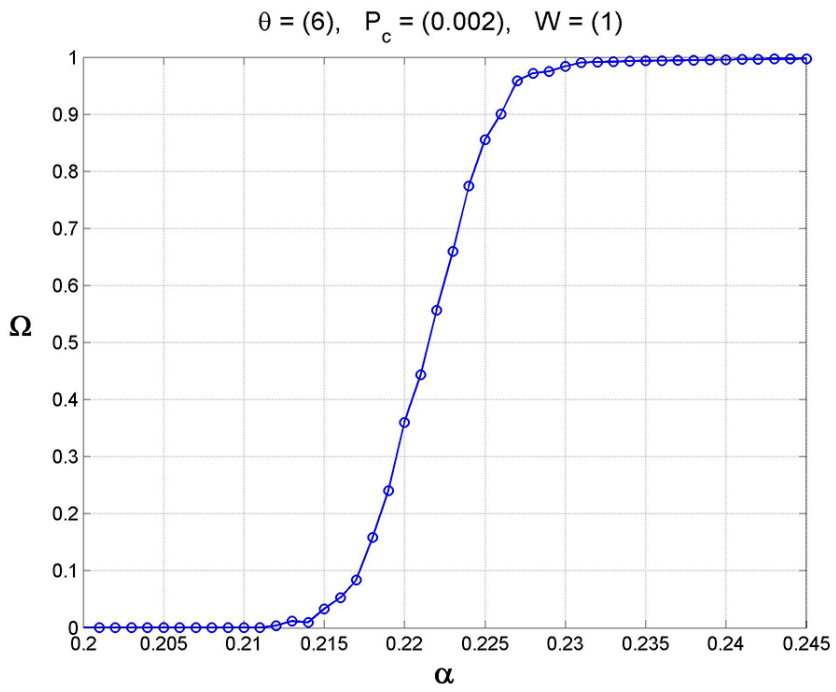

Figure 4.7: Ω *vs.* α. $\lambda = 1,\ m = 1x10^4,\ \theta = 6,\ P_c = 0.2\%,\ r = 30.$

4.2.3 Critical threshold: θ_c

We are now interested in finding the *critical threshold*, θ_c, the maximum threshold for which we can trigger a global cascade; this is when we have a proportion of adoption of 1, or more generally when $\Omega \geq q$ for $q \in [0, 1]$. From figures 4.5 and 4.6 we can see that if $q = 1$ then θ_c is 49.

A natural extension of figures 4.5 and 4.6 is to observe what changes when we vary P_c. As we can see from figure 4.8 (where the function from previous figures is the one in the middle) changes in the connectivity have an impact on the location of the θ_c. This is expected; if we increase P_c then we expect that influence will be easier to achieve even on a higher θ given that we have a bigger number of total connections, therefore the θ_c also increases.

If we take $q = 0.5$, i.e. we are interested in the θ_c that give us an Ω of at least 0.5 and we analyze critical threshold versus population size, we obtain figure 4.9. This behaves linearly for the range where we varied the population size (0 to $30,000$) and as we can see the adjusted linear plot has a linear correlation coefficient of $R = 0.9984$.

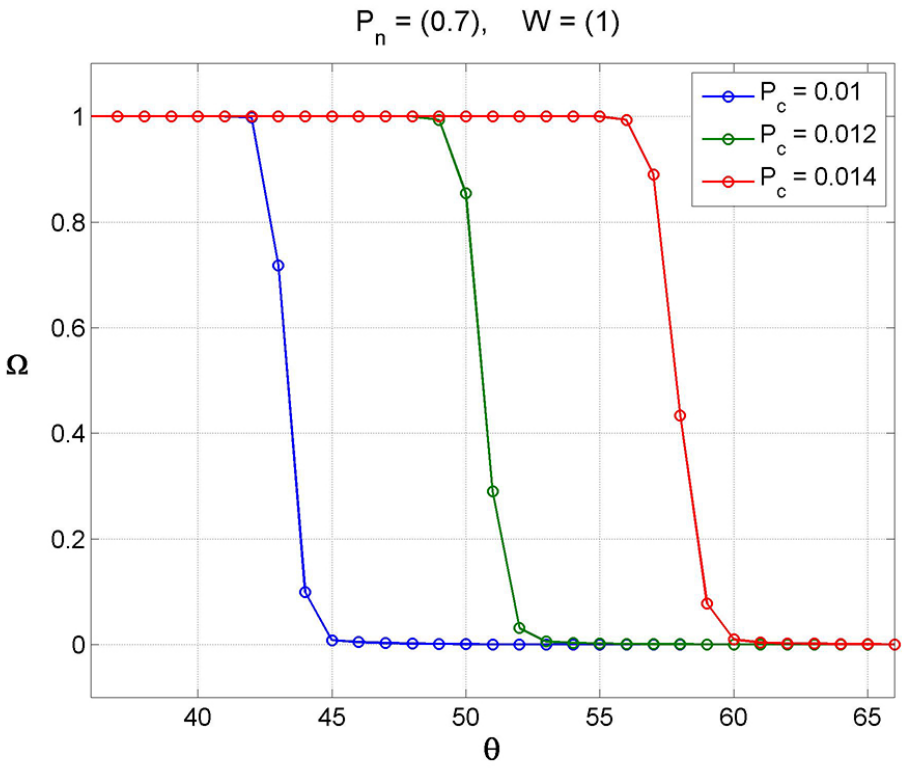

Figure 4.8: Ω *vs.* θ. $\lambda = 1,\ m = 1x10^4,\ \alpha = 0.30,\ r = 600.$

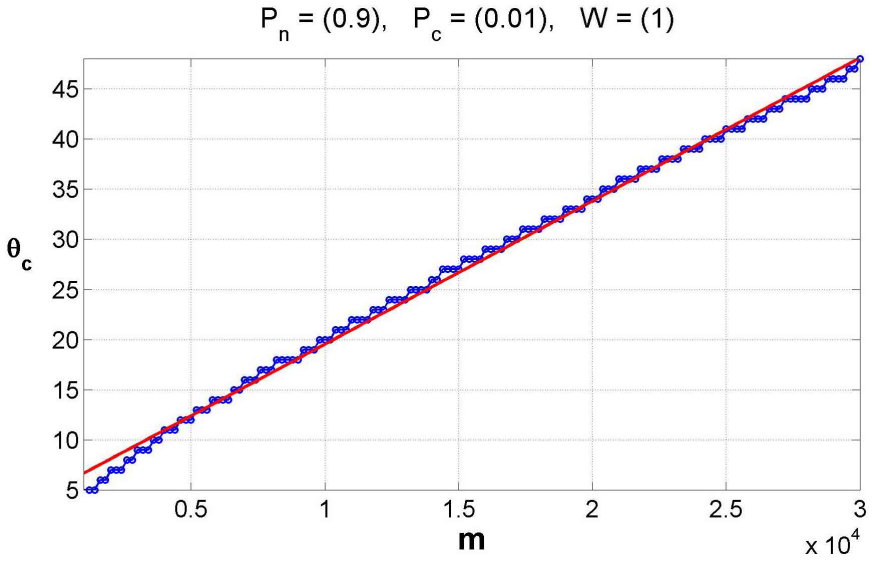

Figure 4.9: Critical threshold as a function of population size: θ_c *vs.* m. $\lambda = 1$, $\alpha = 0.10$, $P_c = 1\%,\ r = 30$. The adjusted plot has the equation: $\Omega = 0.0014m + 5.3$ with $R = 0.9984$.

4.2.4 Adding P_c

So far we have been able to observe the behavior of Ω with respect to θ (figures 4.5, 4.6 and 4.8), α (or P_n in figure 4.7), P_c (in figure 4.8) and indirectly with respect to m in figure 4.9. It was also observed that with respect to population size the behavior is linear for a given set of parameters. We now leave population size constant and try to analyze the same phenomenon in a more general way. Figure 4.10 and figure 4.17 will be the only images with this particular point of view.

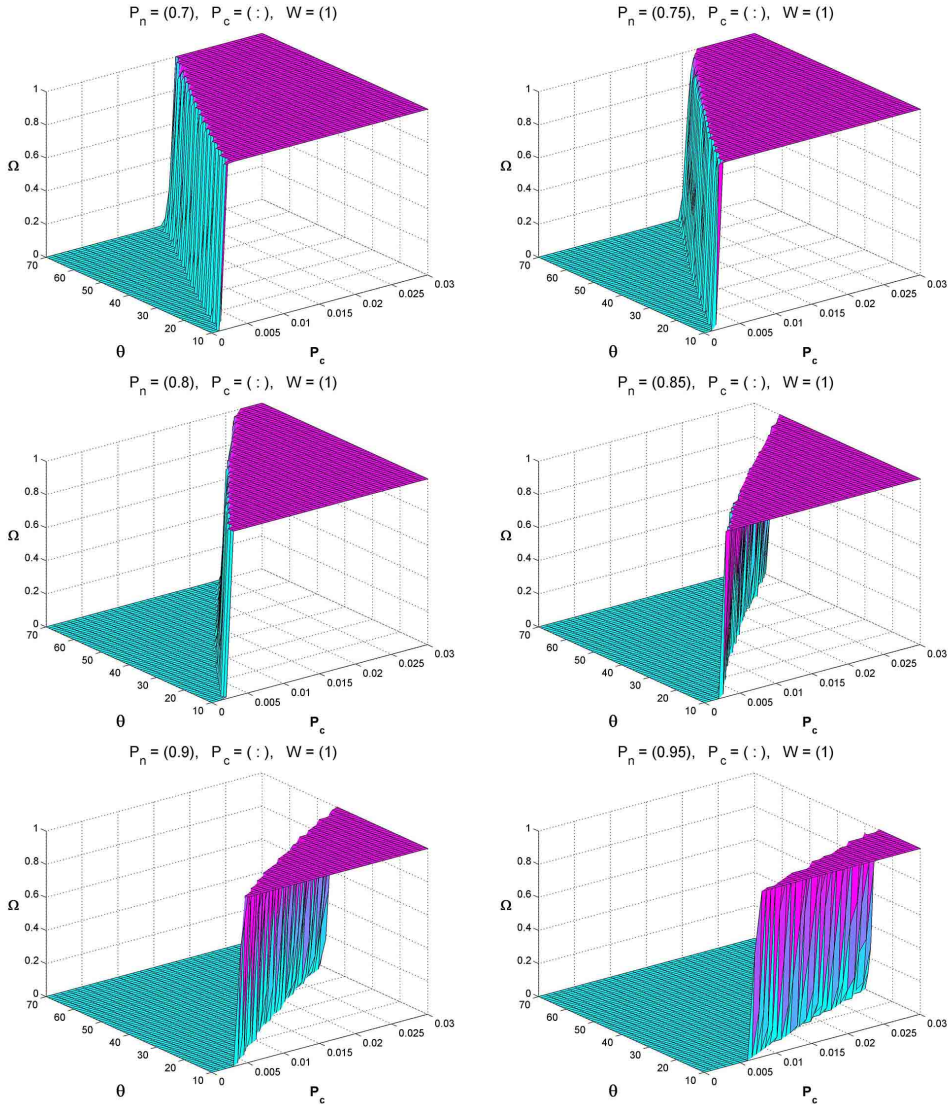

Figure 4.10: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 1,\ m = 10,000,\ r = 30$.

In figure 4.10 we can now see six 3-D plots of the diffusion process. Each image corresponds with a fixed value of $P_n = 1 - \alpha$. We can see in each image the Ω for a range of θ and P_c. Several things can be noted:

1. As expected the zones of low θ and high P_c have higher Ω.
2. As we reduce the value of α, the zone where $\Omega = 1$ shrinks to the corner where we have low values of θ and high values of P_c. This happens because in order to reach total influence given that we reduced the initial seed population, one or both of these parameters have to be changed accordingly.
3. For a given P_c the critical threshold θ_c decreases as we reduce α.
4. The θ_c is found at the intersection of the plane of high growth and a given P_c. We can see that as α is reduced, this plane lowers its projection into the θ-axis. Therefore for smaller values of α the diffusion happens faster around θ_c.

To finish this section we plot another Ω *vs.* (P_c, θ) in figure 4.11. This figure has a range of parameters including those used in figure 4.4. We can see that for $\theta = 3,\ \alpha = 0.10,\ P_c = 0.06\%$ we obtain a $\Omega \approx 0.85$ which is congruent with our results in figure 4.4. This figure and the rest included in this thesis (with the exception of figure 4.17) will have the same viewpoint.

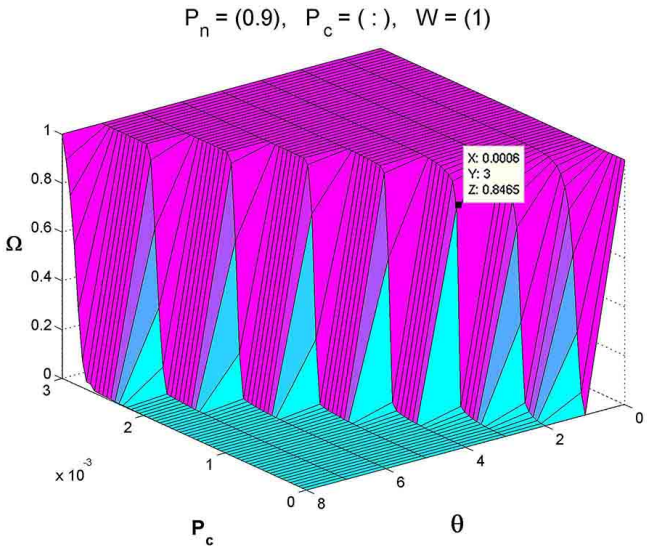

Figure 4.11: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 1,\ m = 10,000,\ r = 30$.

4.3 Two Societies

For two societies we now have substantially increased complexity. Instead of being scalars, both the weight and connectivity are now matrices. Also we can now choose between having reciprocal relations between different societies (or not); their populations can be different among other things.

We therefore start with the simplest possible case of equal population sizes: symmetric relations inside each society, a weight matrix of only ones and reciprocal hybrid adjacency matrices. The results of the simulation are plotted in the usual way and shown in figure 4.12. In this figure all six images have the same parameters with the only exception being P_{n_1}. We can note the following:

1. When compared to 4.10 we can notice that instead of just one plain (total adoption) we now have three of these regions: one for total adoption of both societies simultaneously and two for adoption of one of them individually.

2. Looking at the first three images, for low values of P_c (which refers to the connectivity of society 1 with itself) only society 2 manages to achieve total adoption. This makes sense since the connectivity of society 2 is constant at $P_{c_{22}} = 0.1$ which is higher than $P_{c_{11}}$ in that region. Once we increase this connectivity then both adopt fully unless we have high thresholds in which case only society 1 achieves total diffusion.

3. Very clearly, as we decrease α the zone of no adoption grows while the zones with $\Omega = 1$ and $\Omega = 0.5$ shrink.

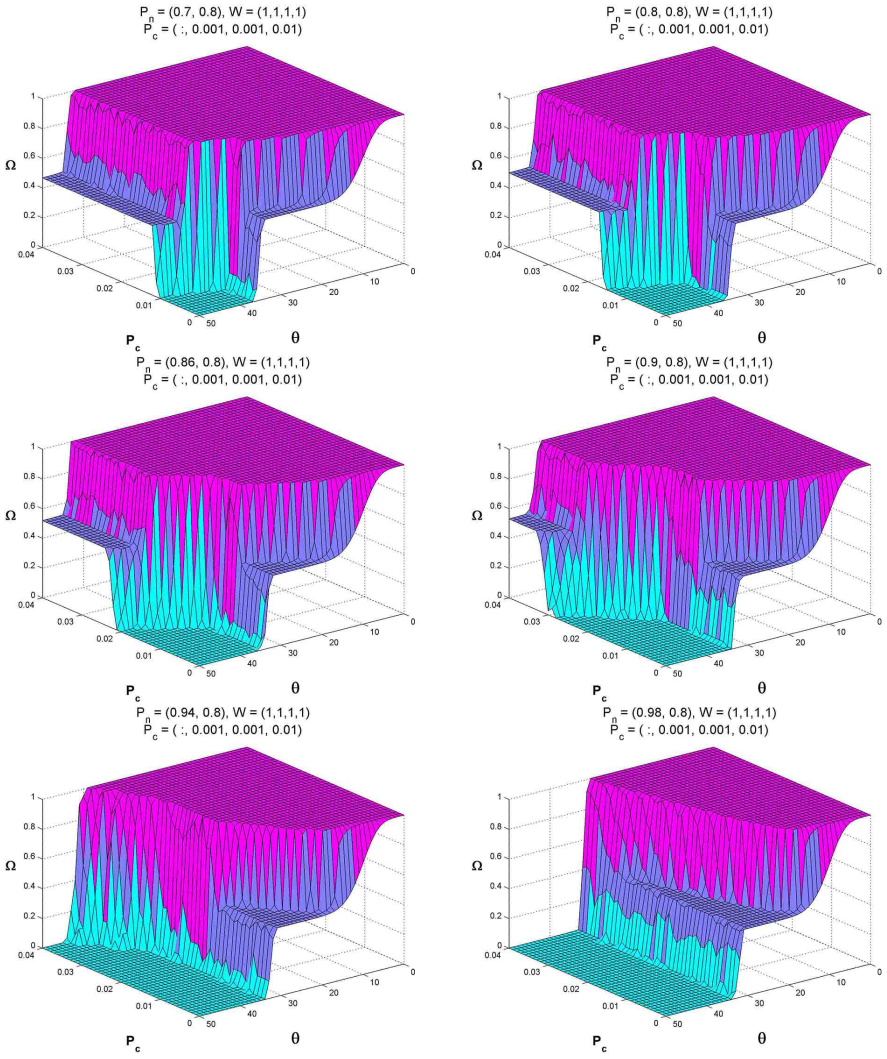

Figure 4.12: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 2,\ m = (10^4, 10^4),\ r = 30,\ P_{n_2} = 0.8$.

We also add figure 4.13 as another example. The parameters and images are all the same with the only change being that $P_{n_2} = 0.9$ instead of 0.8. We can see that this change decreased adoption of society 2 in all scenarios and also slightly affected society one through the hybrid relations between both.

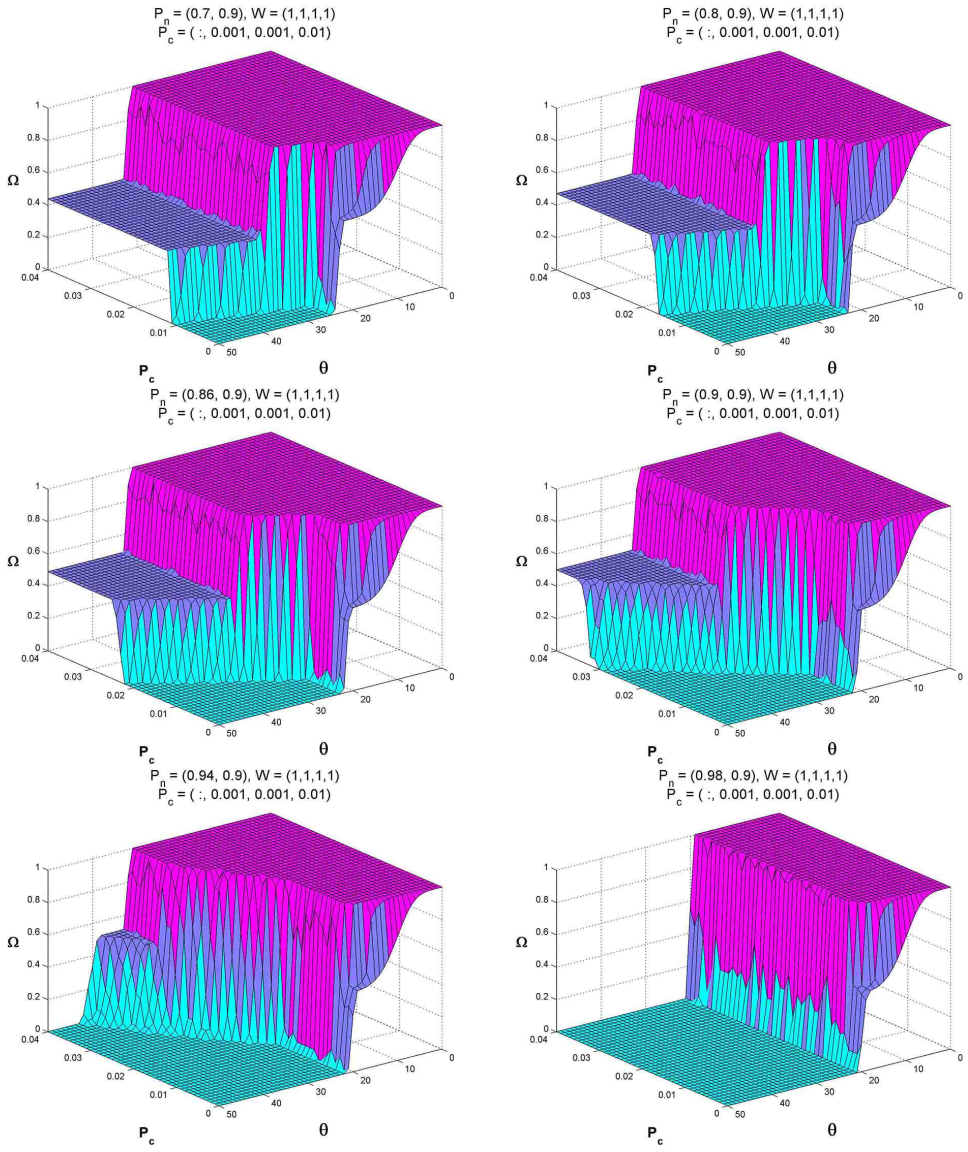

Figure 4.13: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 2,\ m = (10^4, 10^4)\ r = 30,\ P_{n_2} = 0.9$.

To show the local complexity of these plots we include figure 4.14. It shows a zoom made to a section of an image in figure 4.13.

To further emphasize the complex local features of these plots we also include figure 4.15 where we show a zoom of a different section of the domain of a plot taken from figure 4.13. If we take this zoom for all the images presented in 4.13, we produce figure 4.16.

Figure 4.16 shows how complicated the behavior of the diffusion can be at a smaller scale in the parameters θ and P_c. While all the previous figures give the impression that the process is simple and occurs in steps, we can see

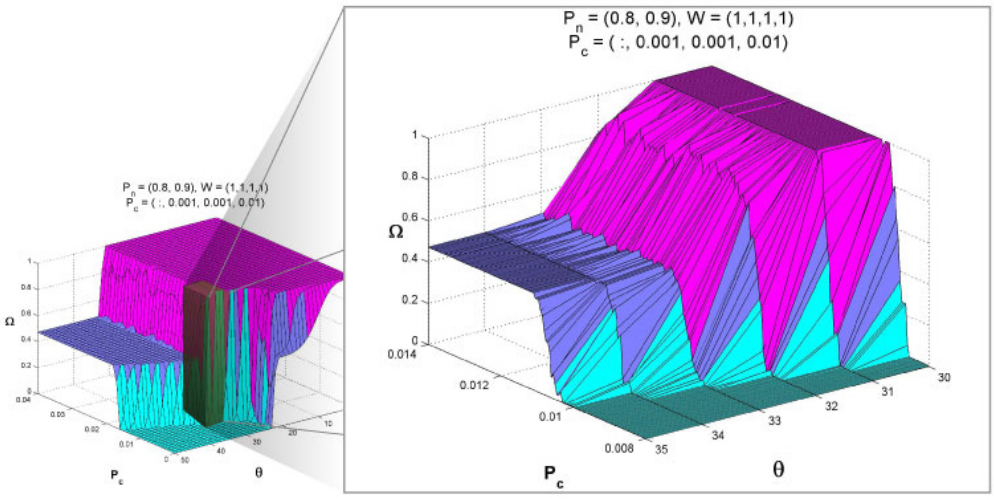

Figure 4.14: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 2,\ m = (10^4, 10^4)\ r = 30$.

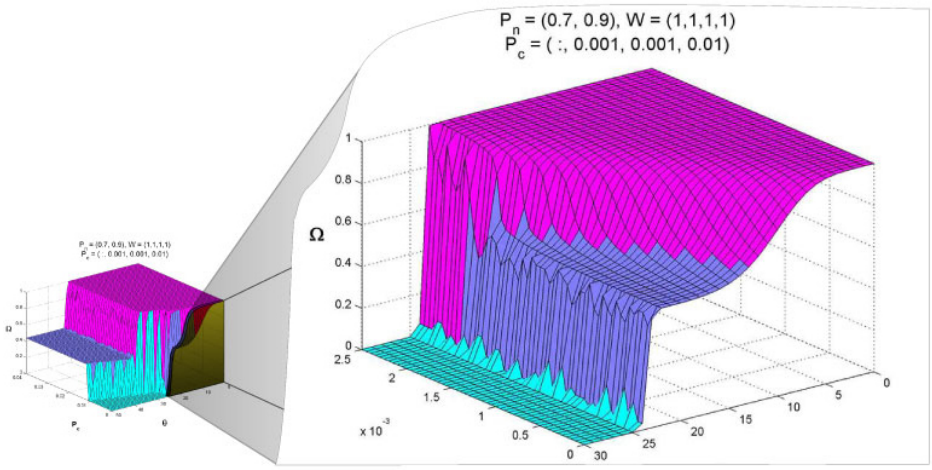

Figure 4.15: Zoom used in figure 4.13 to produce the images in figure 4.16.

that there are zones of smooth surfaces with slow growth.

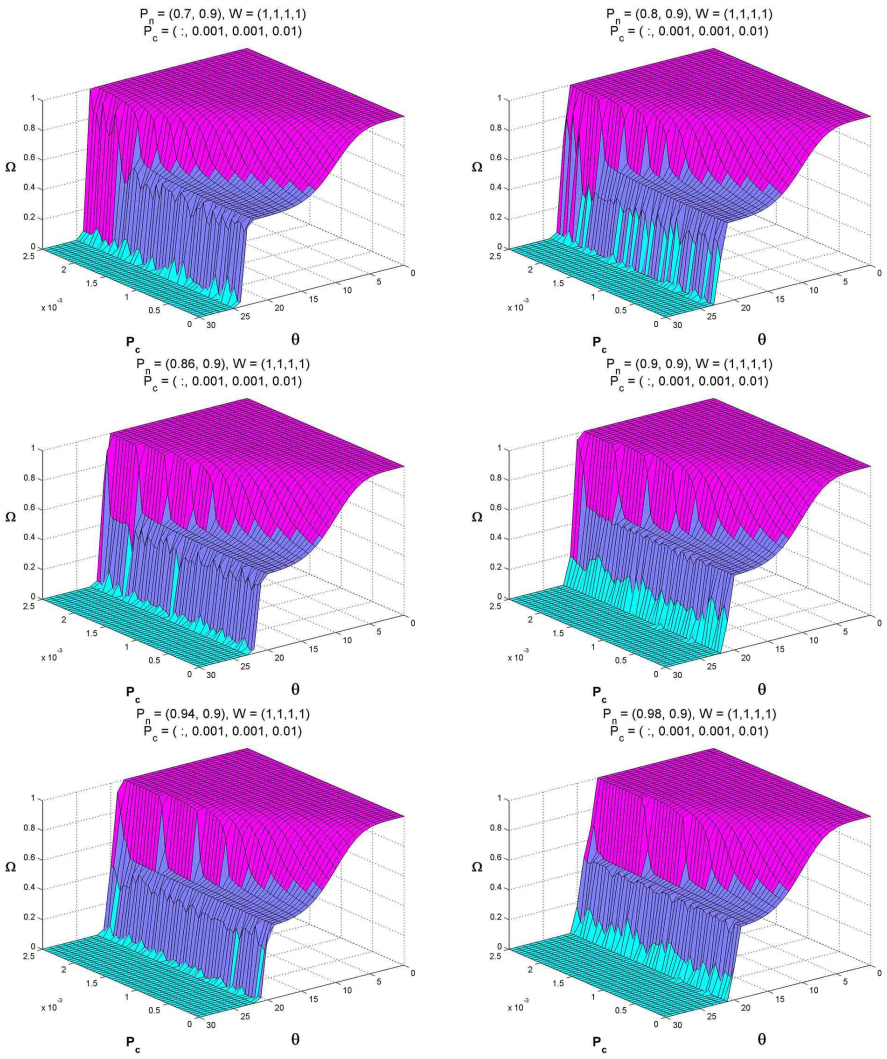

Figure 4.16: Ω *vs.* $(P_c,\ \theta)$. $\lambda = 2,\ m = (10^4, 10^4)\ r = 30$.

4.4 Weights

So far the only weight matrix used has been $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$. In this section we will now present several variations from this weight matrix and their results. Most of the following cases have real world applications while others have more implausible adaptations.

4.4.1 $\mathbb{W} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

This is probably the most simple weight variation we can do. It is basically the situation where S_1 is isolated and S_2 is deactivated entirely. This means that S_2 affects neither of the two societies present and S_1 only affects itself. The end result is that this is basically the same situation we had with one society but now the population size is twice what it was and half of that (the half of S_2) will never adopt the new behavior (except for initial conditions of α_2).

The results are included in figure 4.17 (right column) along with the previous results from 4.10 (left column) for easy comparison. As we can see, the behavior is identical, as it should be.

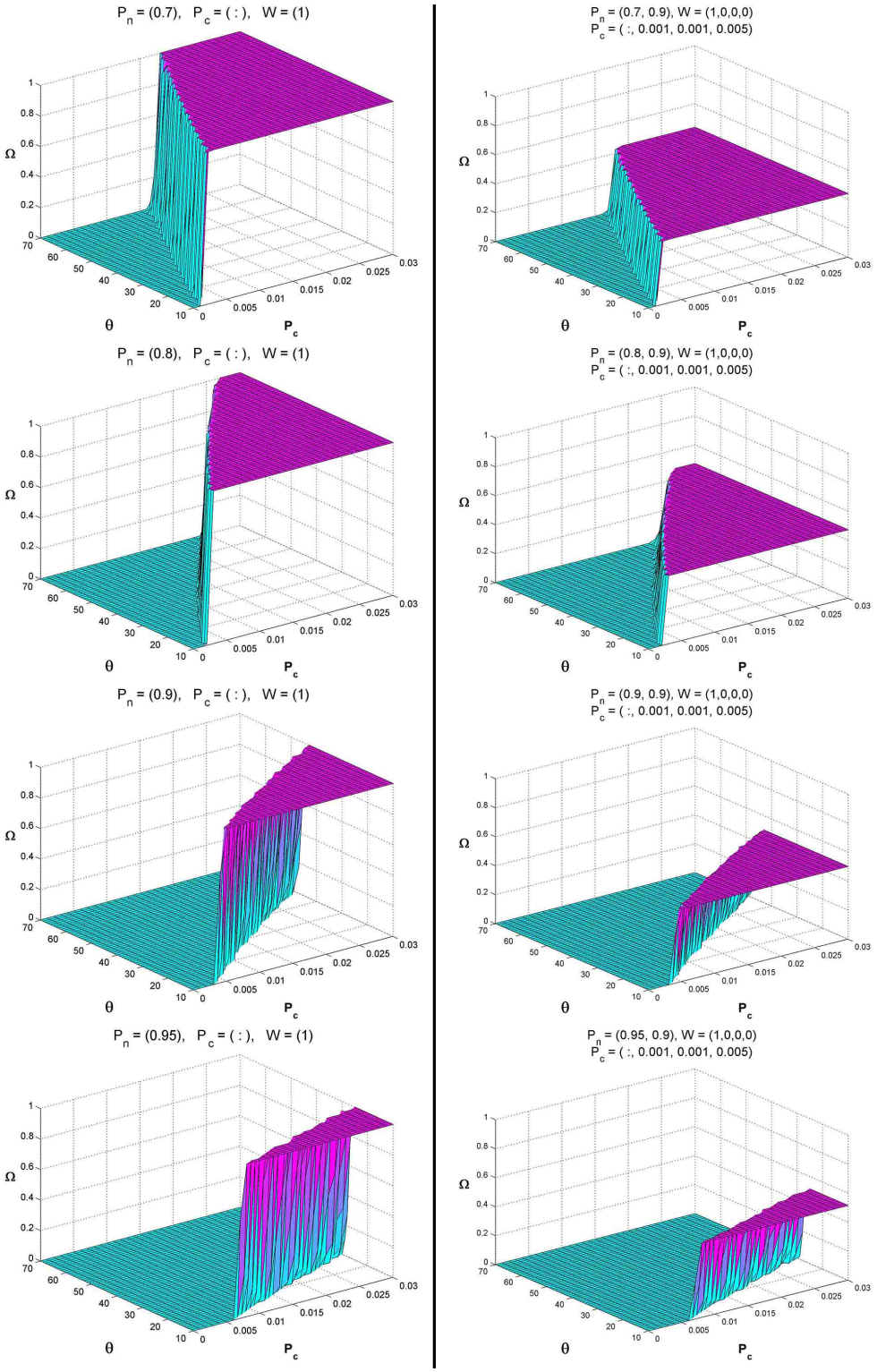

Figure 4.17: Comparison between one society and two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$.

4.4.2 $\mathbb{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

This case has both societies active but not interacting. We can now compare this situation with the canonical case $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ to understand the differences. The results are presented in figures 4.18 ($P_{n_2} = 0.8$) and 4.19 ($P_{n_2} = 0.9$).

What is expected occurs: since there is no interaction between the two societies and therefore no feedback to spread the innovation with greater speed, we see that the zone with $\Omega = 1$ reduced in size because it is harder for both societies to achieve a global cascade. Also, for the same reason, the zone with $\Omega = 0$ increased in size. And finally we now have bigger zones where only one of the societies fully adopts the new behavior.

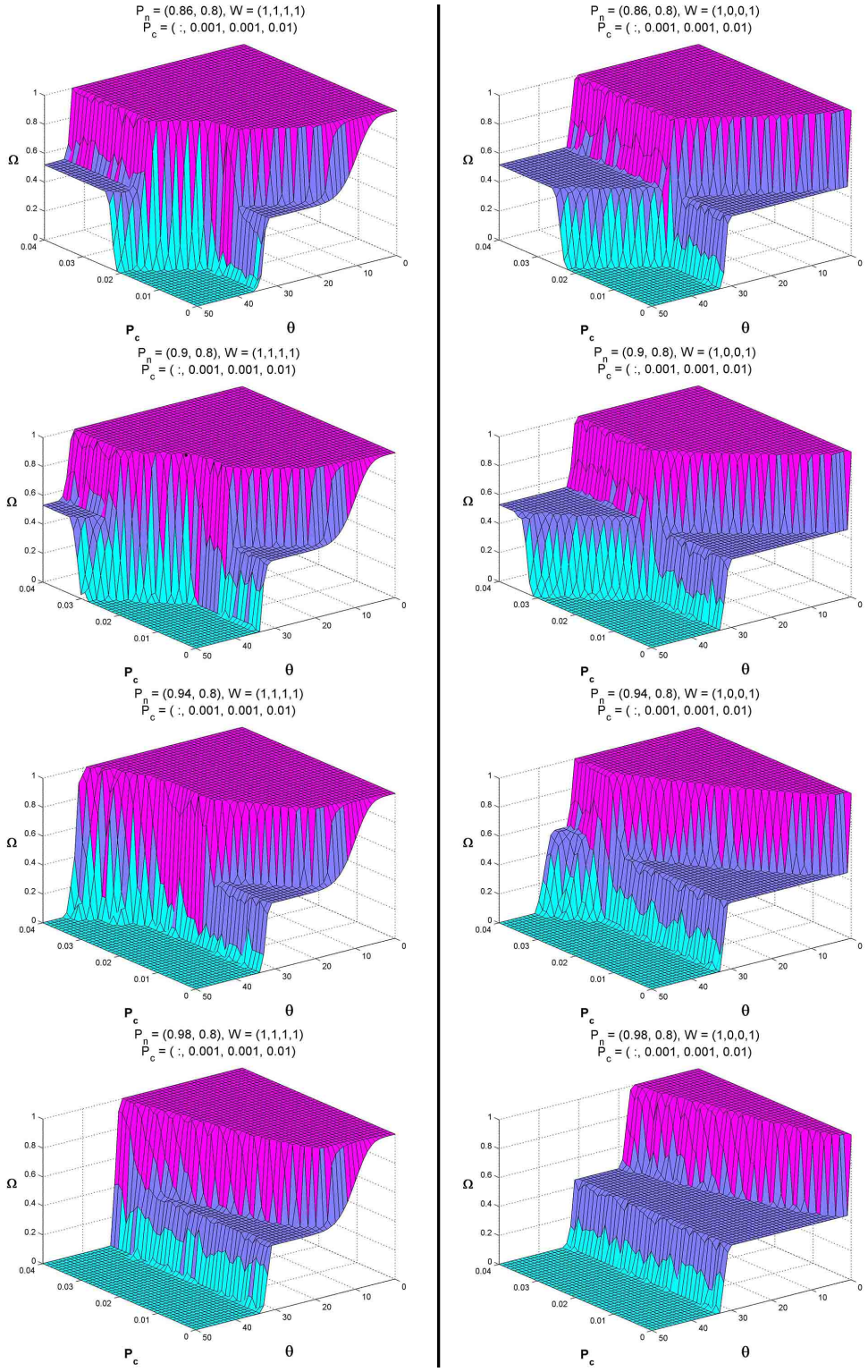

Figure 4.18: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

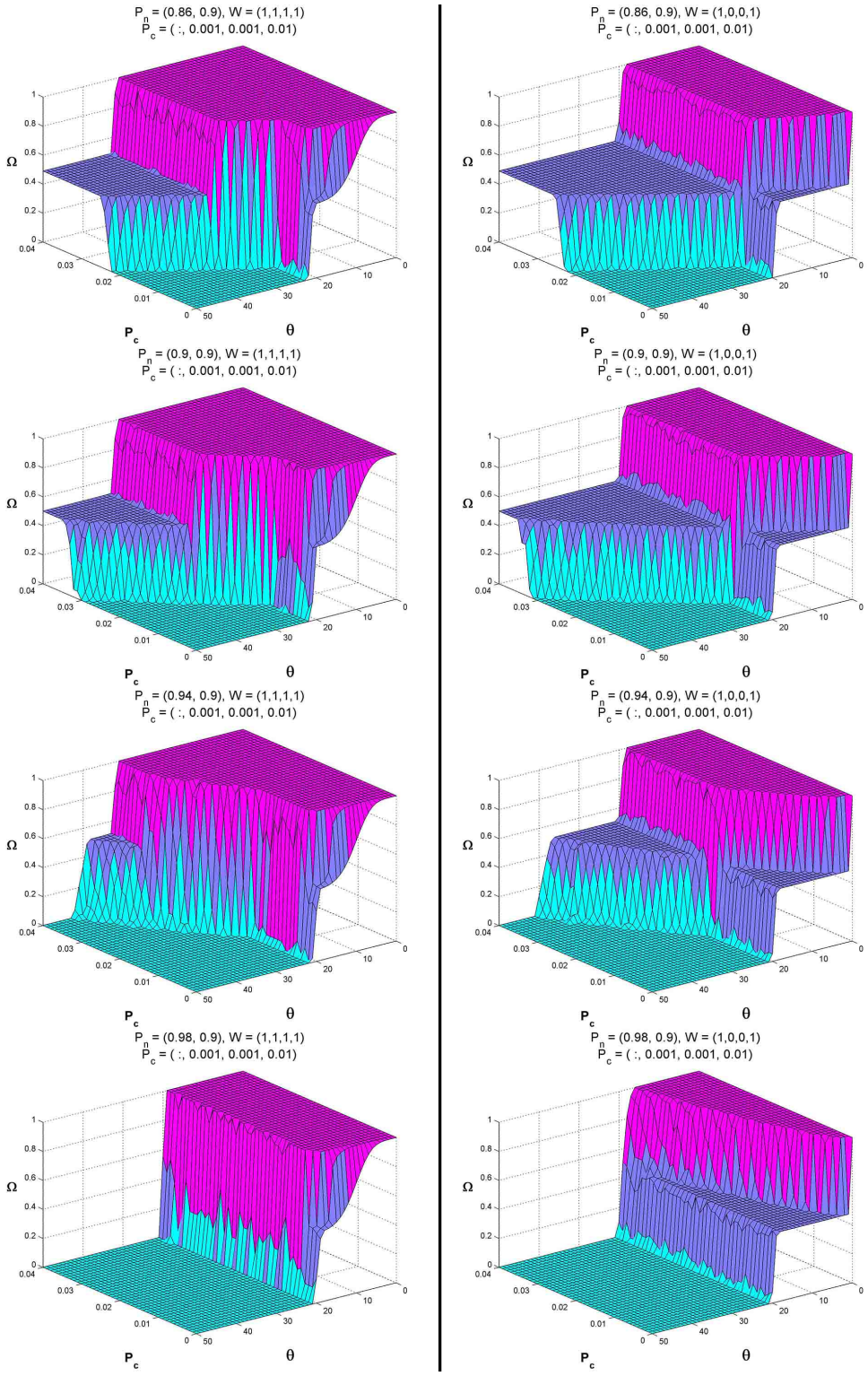

Figure 4.19: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.3 $\mathbb{W} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$

In this scenario we have both societies interacting and active. Since $\mathbb{W}_{21} = 1$ then S_1 influences S_2 positively, meaning that S_2 will be influenced by S_1 to adopt the innovation; in contrast, $\mathbb{W}_{12}$ = -1 which means that S_1 has, effectively, a more complicated threshold rule where members of S_2 contribute in a negative way. Basically S_1 is less likely to adopt the innovation given that S_2 is adopting it. Since the program currently cannot make individuals switch back to the original behavior, then we never get a scenario where S_1 abandons the innovation caused by stronger adoption in S_2.

The results are included in figures 4.20 and 4.21 for $P_{n_2} = 0.8$ and $P_{n_2} = 0.9$ respectively. We can notice that now the plateau linked to full adoption by society two alone has increased in size, which makes sense given that S_1 has a lower chance of adoption. Notice how the plateau linked to full adoption on S_1 is identical to the canonical case.

Obviously the decrease in zones $\Omega = 1$ close to the plateau for S_2 is even bigger than in the previous case $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ since a contribution 0 is bigger than the negative contribution in this case.

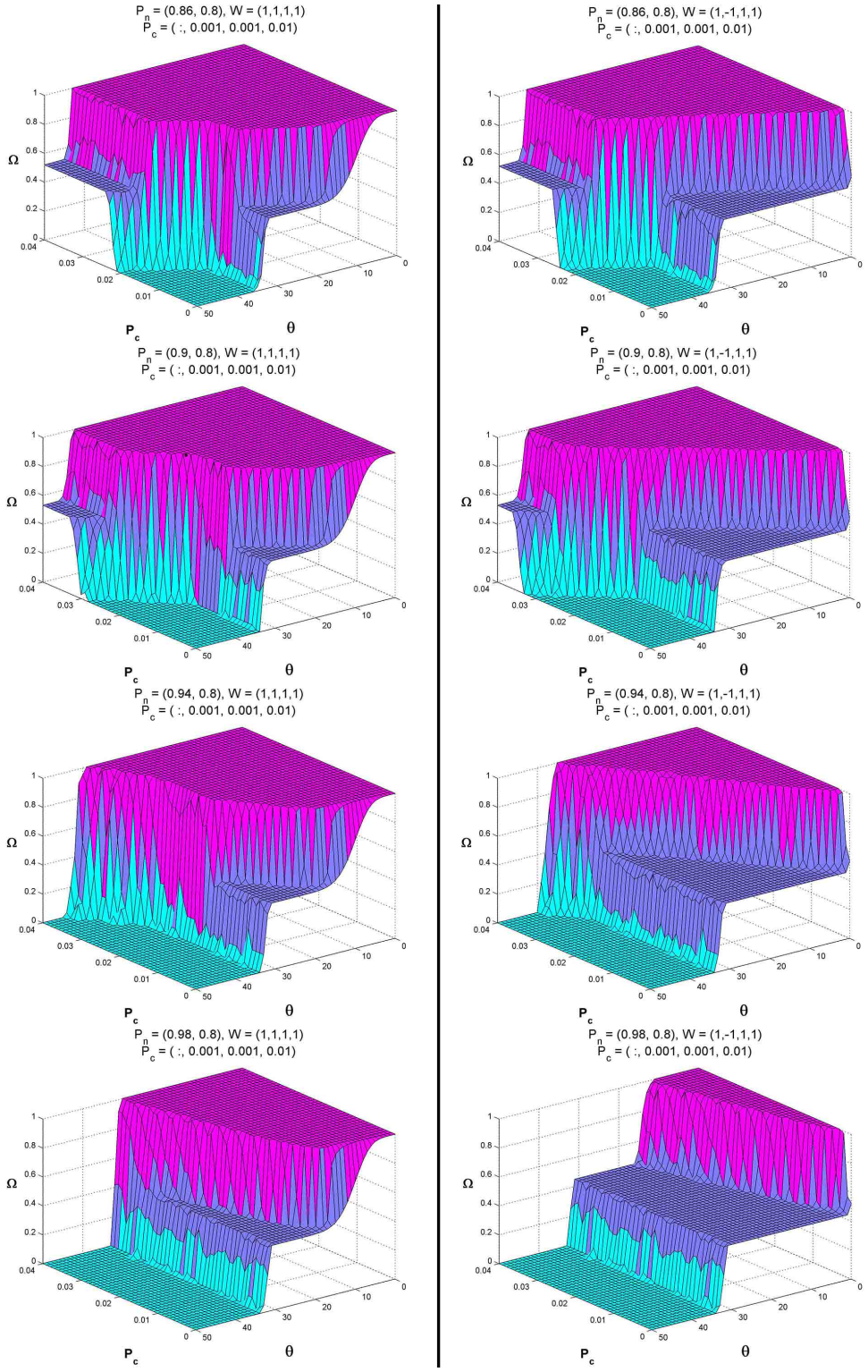

Figure 4.20: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ 1 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

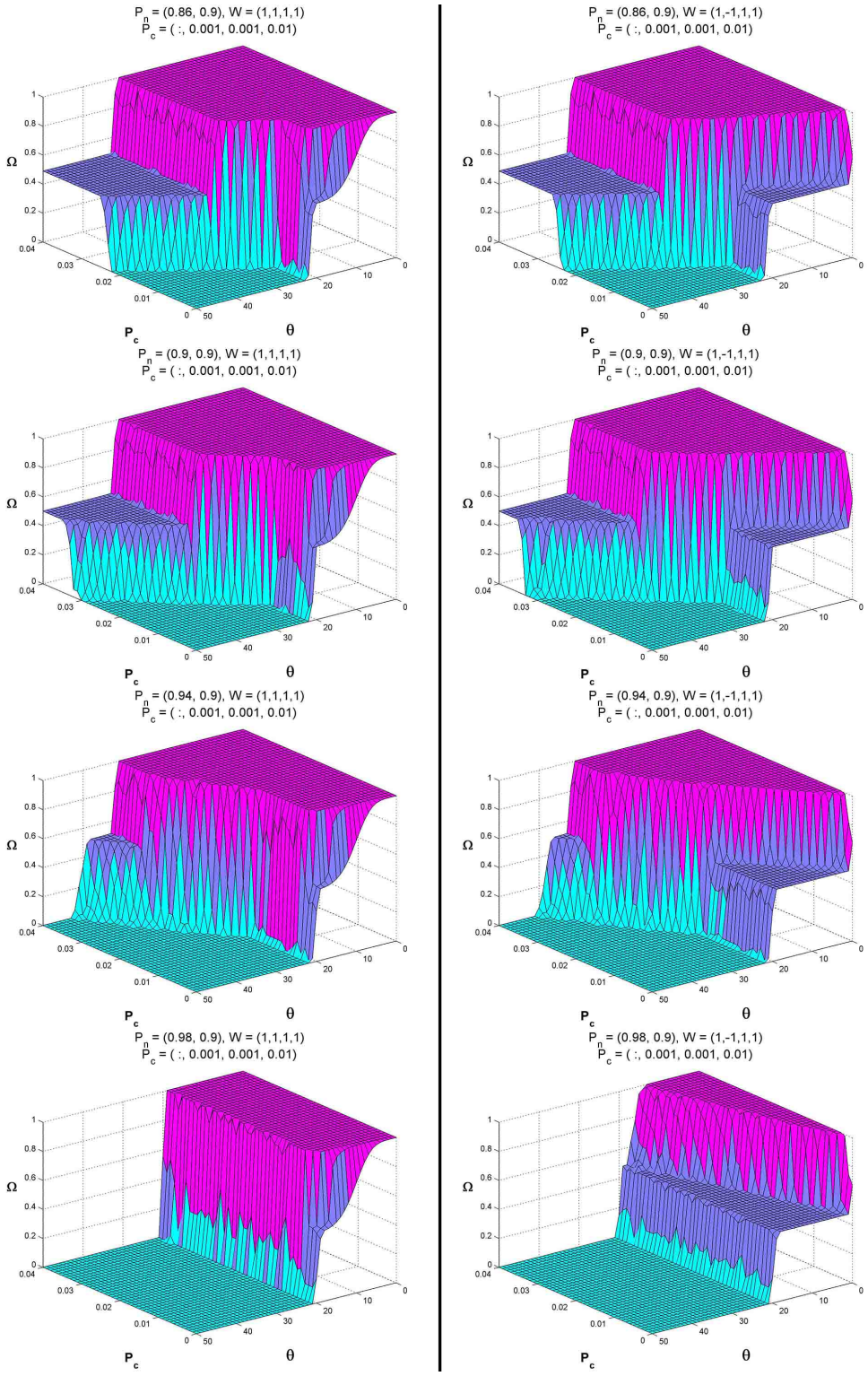

Figure 4.21: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ 1 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.4 $\mathbb{W} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

We now have a negative interaction between both societies so we expect that what happened in the last scenario with S_2 will now occur for both S_1 and S_2. To verify this and summarize the last three weights we include a comparison in figure 4.22.

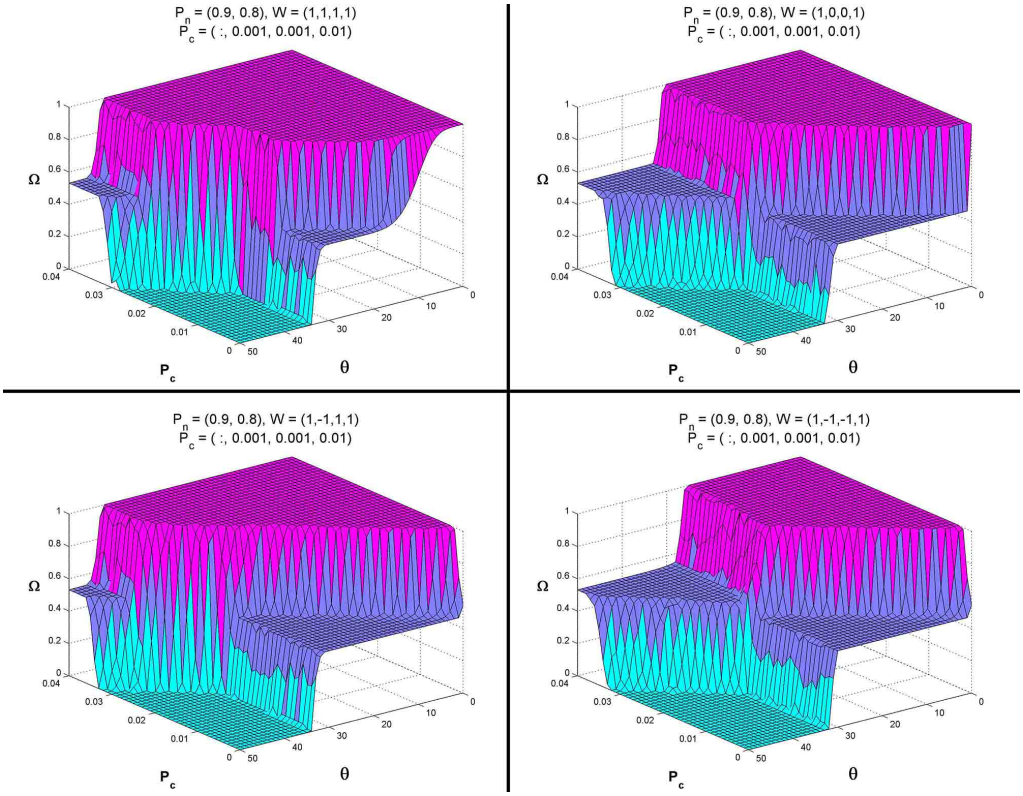

Figure 4.22: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ 1 & 1 \end{smallmatrix}\right)$ and $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right)$.

As we can see, the reduction in the zone of total adoption $\Omega = 1$ varies with the weight matrix and the negative contribution of $\mathbb{W}_{12} = \mathbb{W}_{21} = -1$ creates the biggest reduction of adoption. The full results for the weight matrix of this section are plotted in figures 4.23 and 4.24.

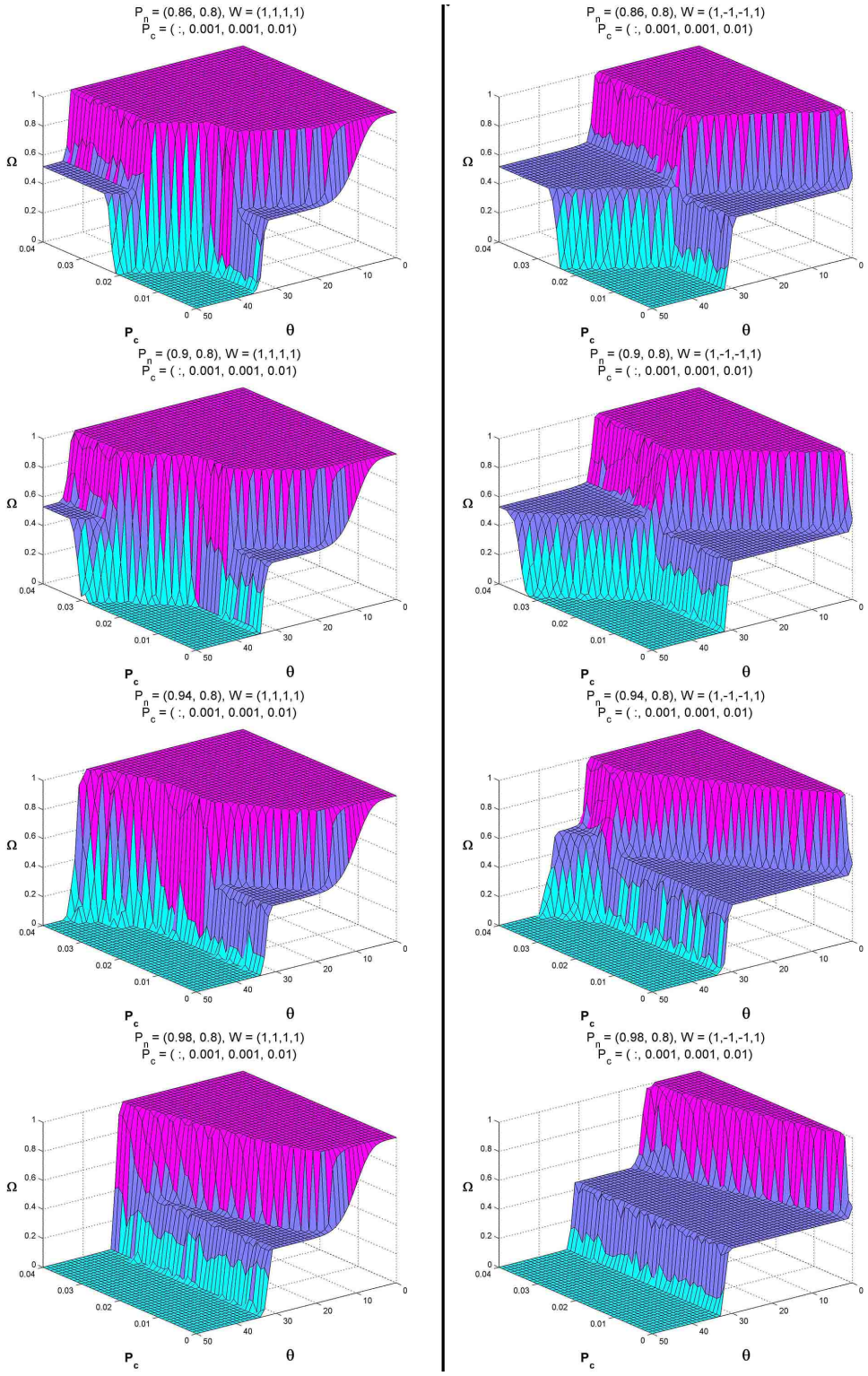

Figure 4.23: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

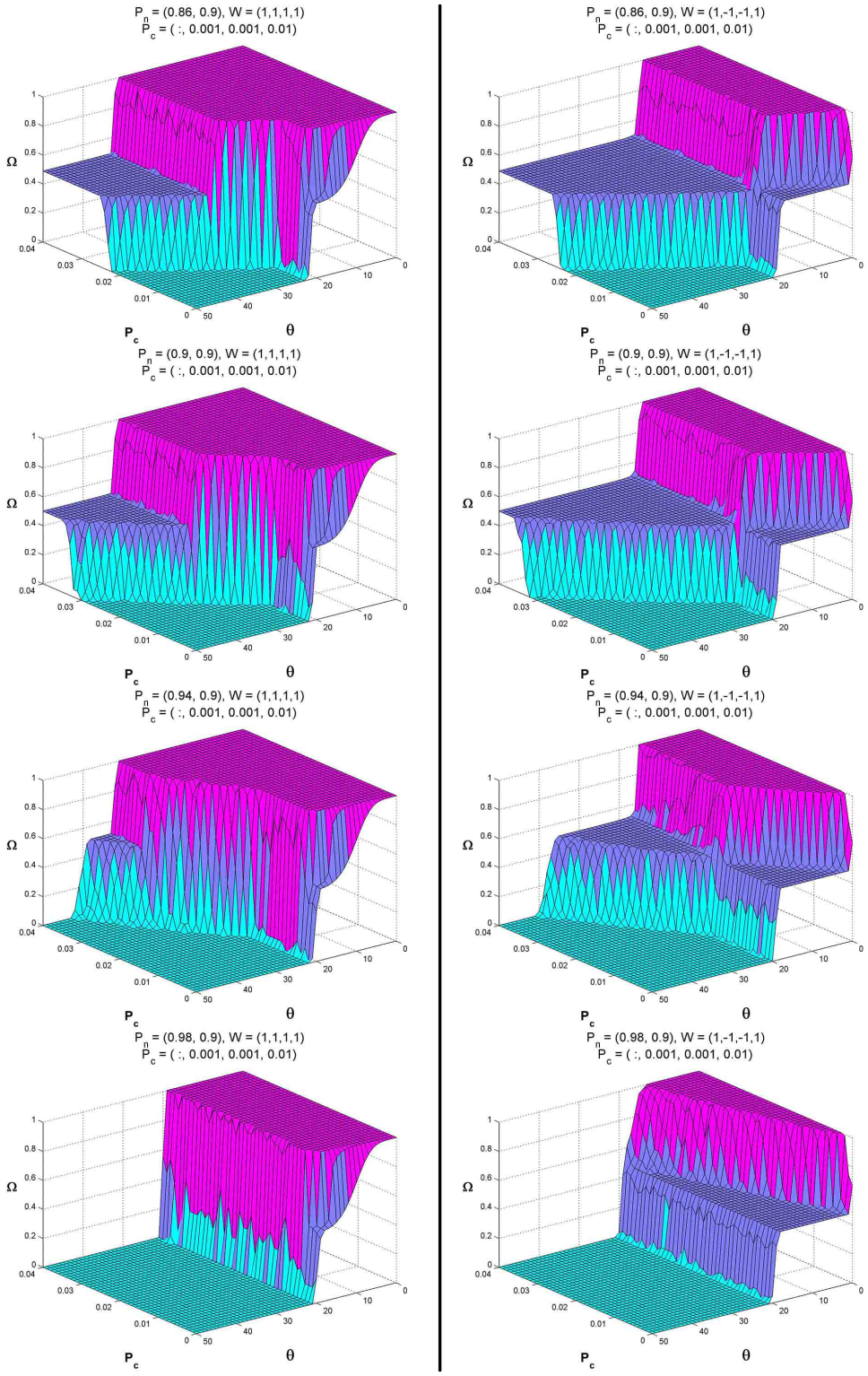

Figure 4.24: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.5 $\mathbb{W} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$

This is the case where society S_1 is affected by itself and S_2 but society S_2 is immune to influence of either societies. The results are included in figures 4.25 and 4.26. We can observe the following:

1. There is no zone with $\Omega = 1$, with the exception of $\theta = 0$ which is a trivial case and will produce $\Omega = 1$ regardless of the parameters utilized. This is reasonable given that S_2 is not affected by either society.

2. The general behavior of S_1 is almost the same as when the case was that of one society or two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ (figure 4.17). The only difference is that now, the initial population seeded by $\alpha_2 = 0.8$ (or $\alpha_2 = 0.9$) is affecting S_1 as well as its own seed population. This achieves higher adoption than in the isolated case.

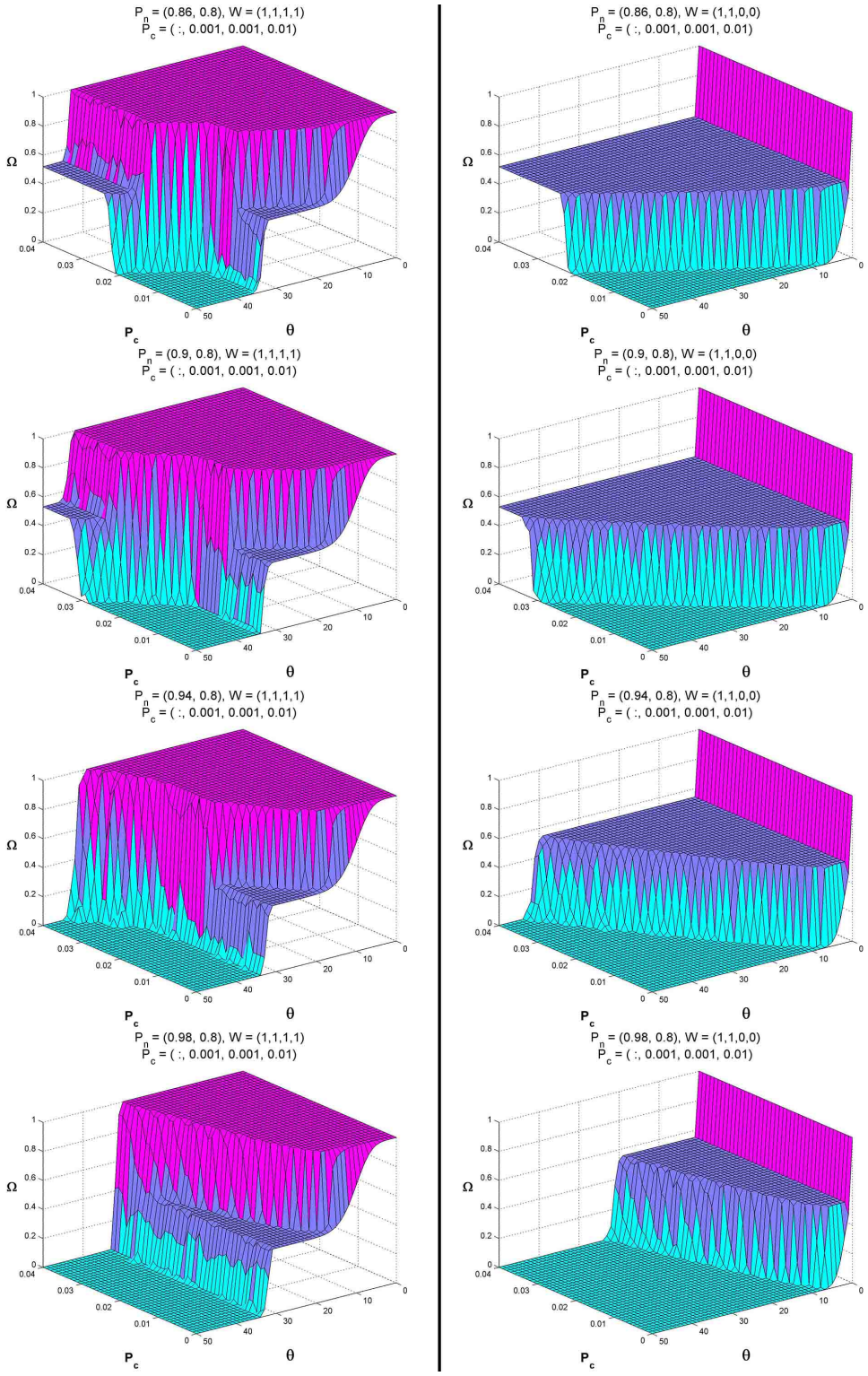

Figure 4.25: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

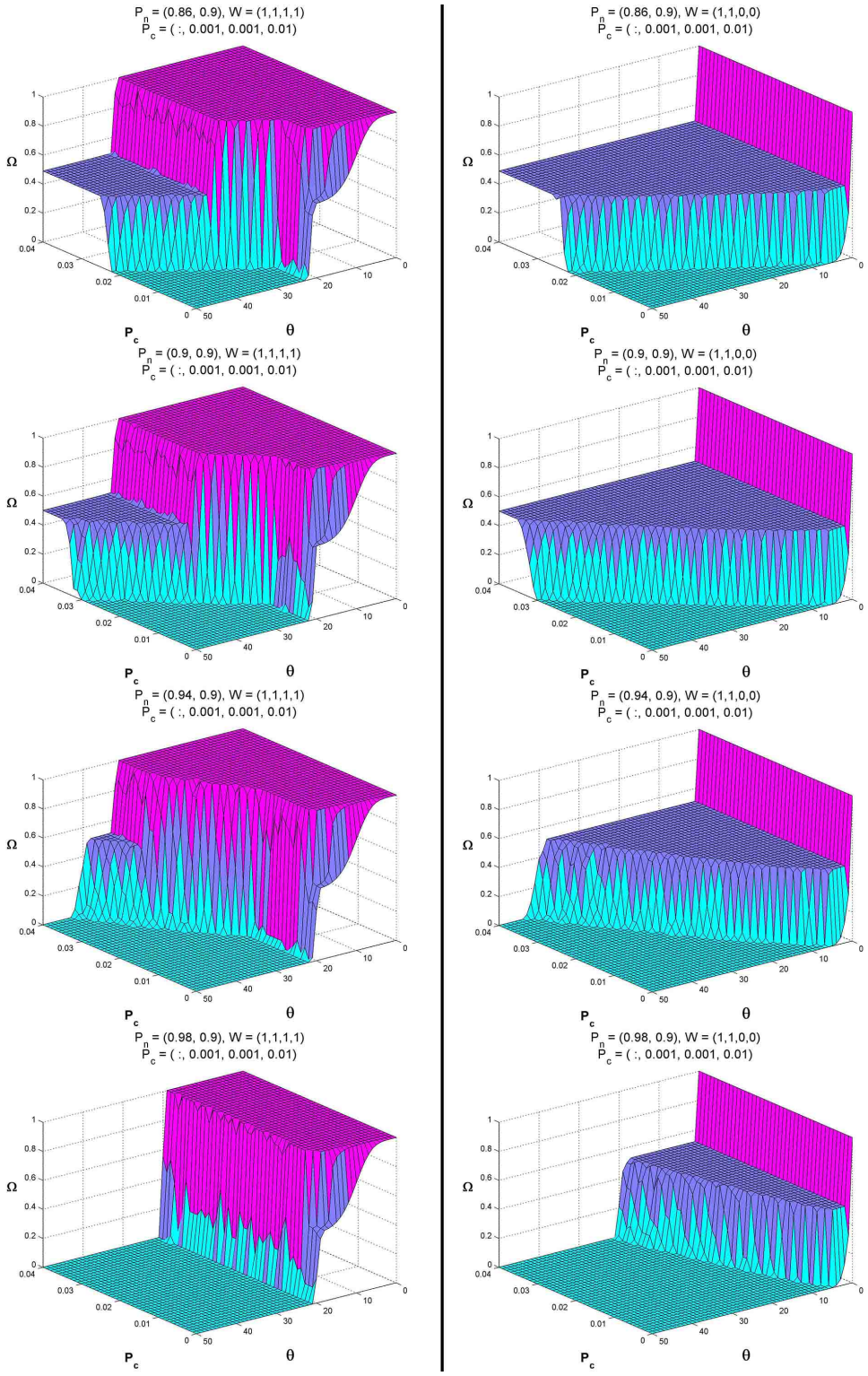

Figure 4.26: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.6 $\mathbb{W} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$

Now we have the scenario where only the influence from S_1 matters to both societies, but none are isolated. In contrast to the last weight observed, for low values of θ, society S_2 will adopt the innovation. That is, if S_1 grows enough it also affects S_2. These are basically the only notable differences between these two cases.

Results are plotted in figures 4.27 and 4.28. If we play close attention to these figures and compare them to figures 4.25 and 4.26 we can see that the behavior is the same as it was before, but with a smooth "hump" attached in the zone with low thresholds. This makes sense because now S_2 can also reach full adoption for low values of θ and when the influence of S_1 is high enough to affect the population.

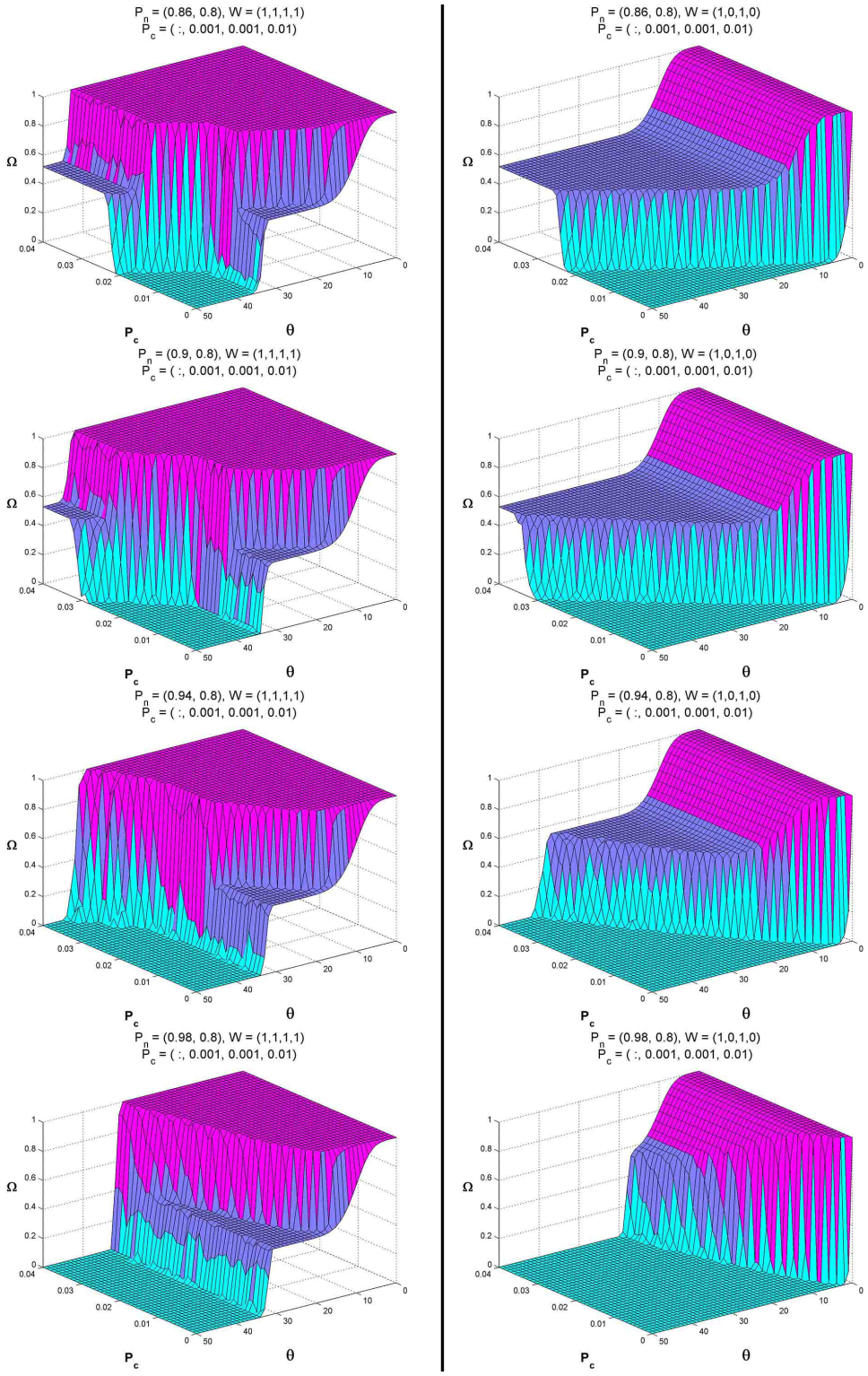

Figure 4.27: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

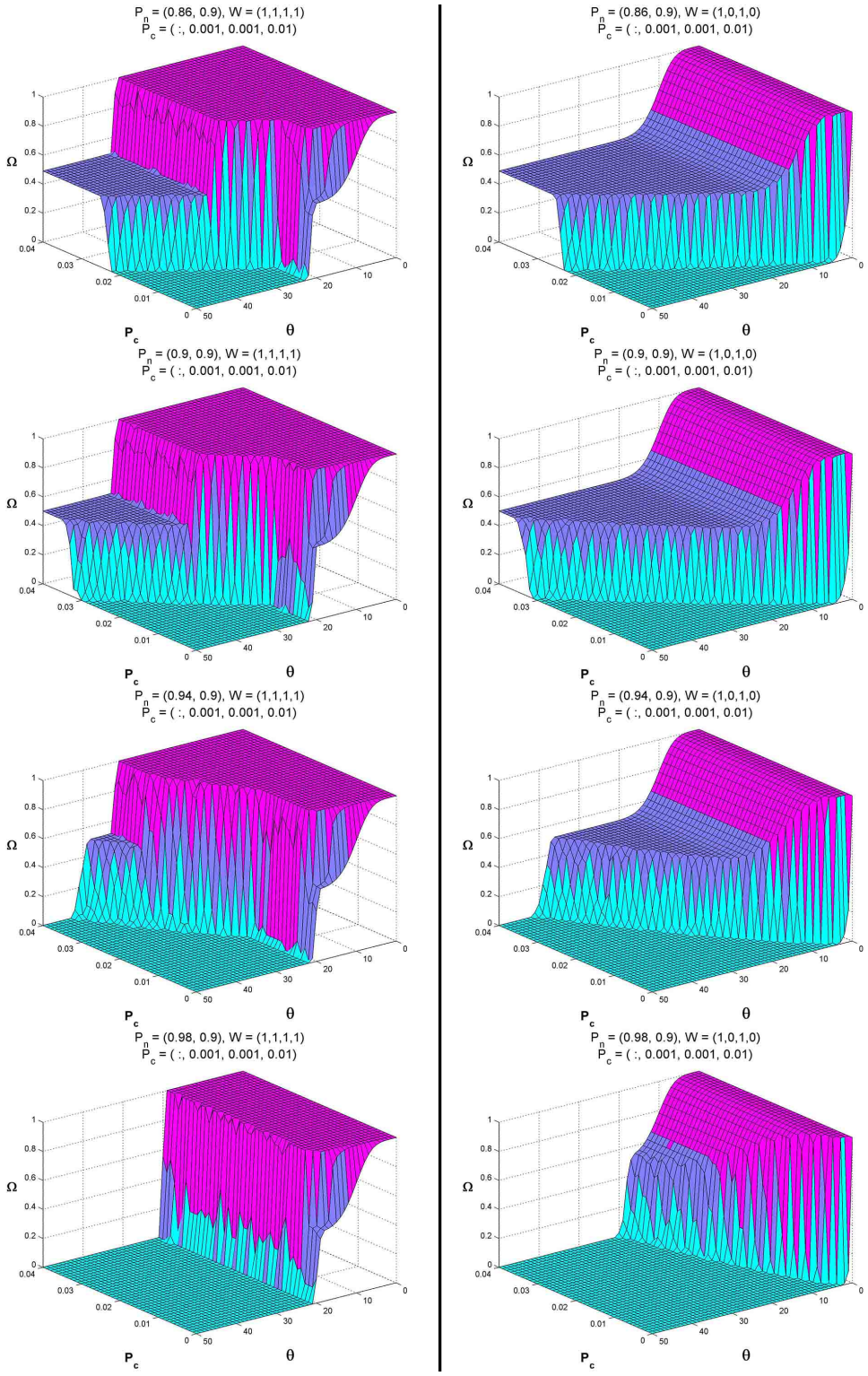

Figure 4.28: Comparison between two societies with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and with $\mathbb{W} = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.7 $\mathbb{W} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

The only relations in this example are hybrid ones; the societies interact with one another and are influenced that way but no internal influence exists. Since $\mathbb{W}_{11} = \mathbb{W}_{22} = 0$, then neither $P_{c_{11}}$ or $P_{c_{22}}$ will be used in the plots of the results because Ω is independent of those two values. We use $P_{c_{12}}$ instead. The results, shown in figure 4.29, are very different from all the previous results and this comes from the fact that we used a new variable to plot them. Figure 4.30 shows contour plots of figure 4.29 and it is included to better appreciate the resulting behavior.

Finally, for completeness, figure 4.31 is included with $P_{n_2} = 0.9$ as we have done before for all other cases.

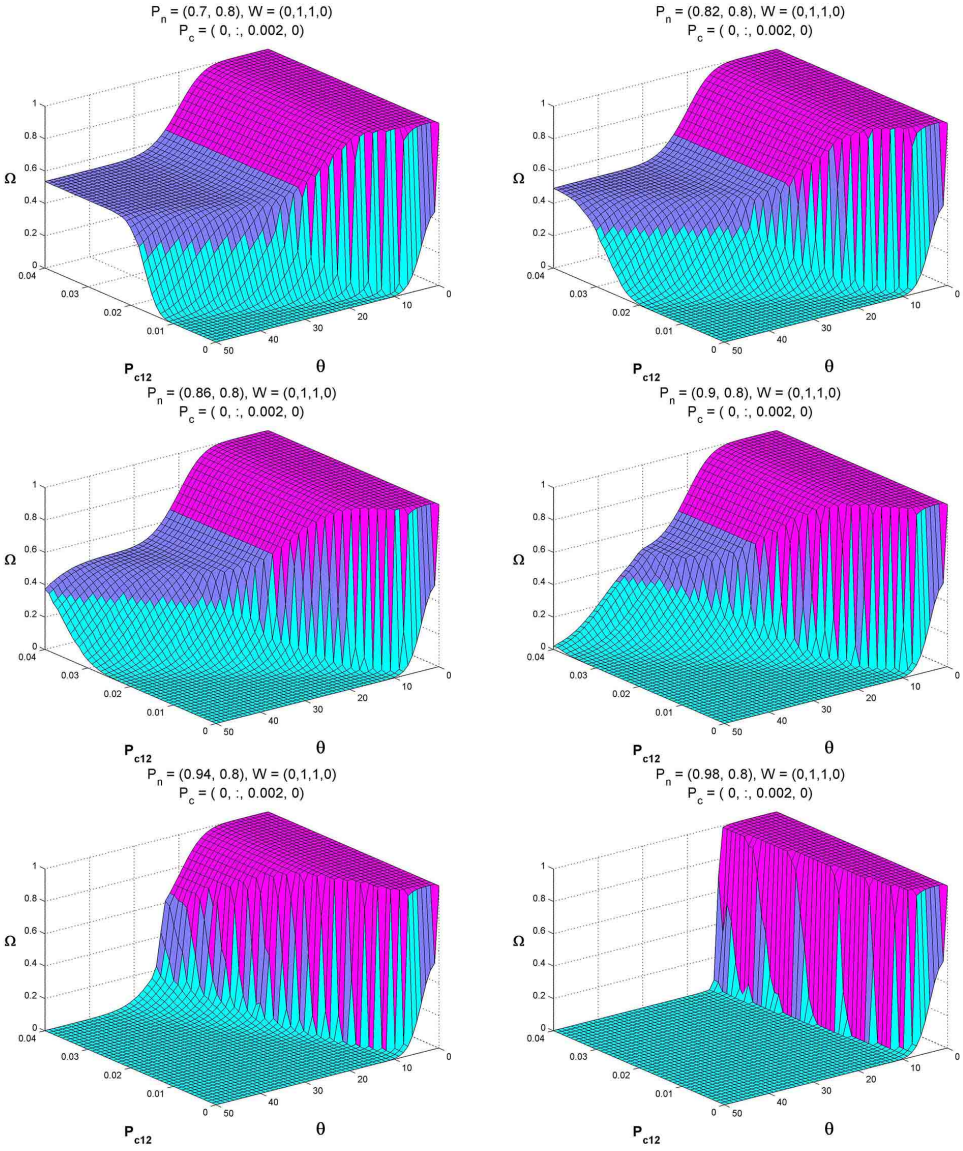

Figure 4.29: Two societies with $\mathbb{W} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.8$.

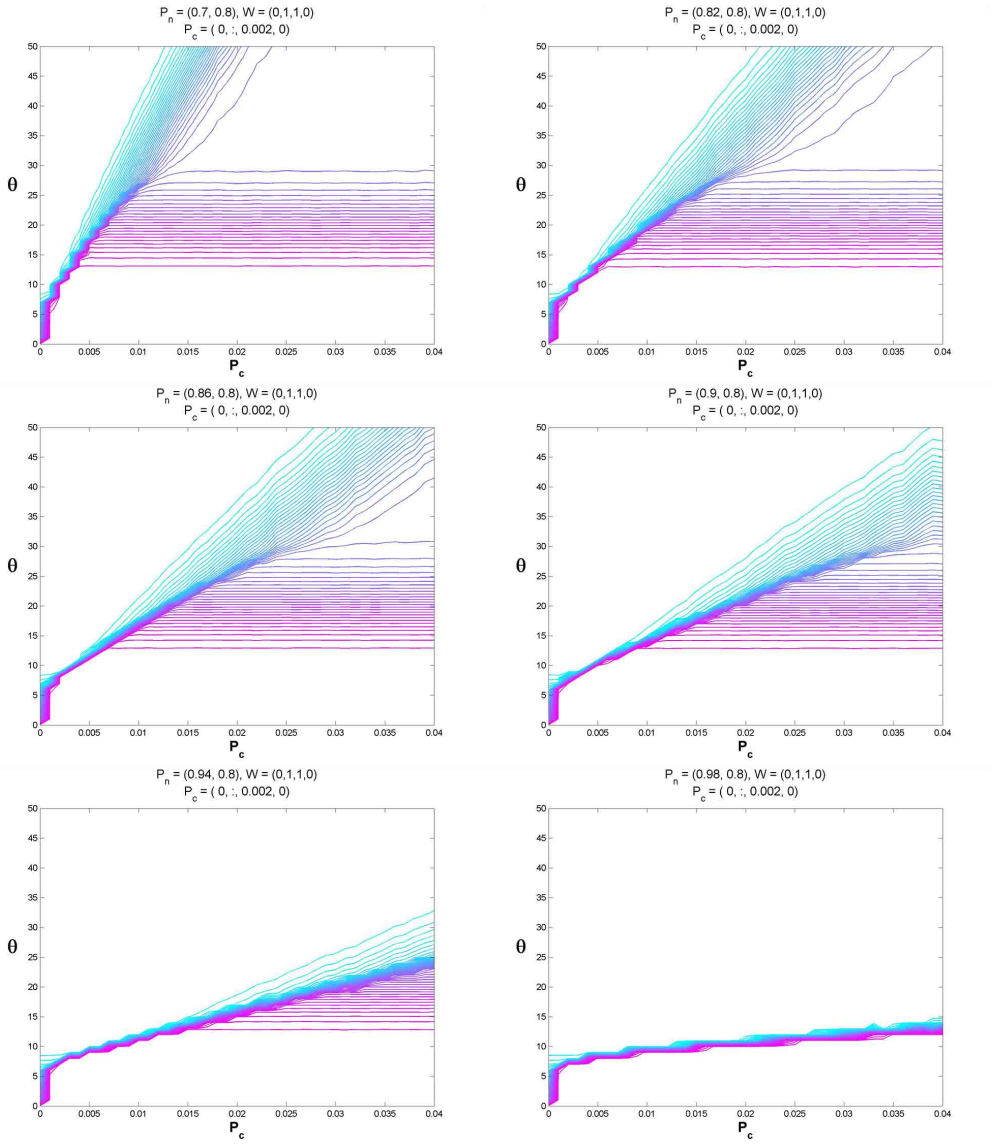

Figure 4.30: Contour plot of images in figure 4.29.

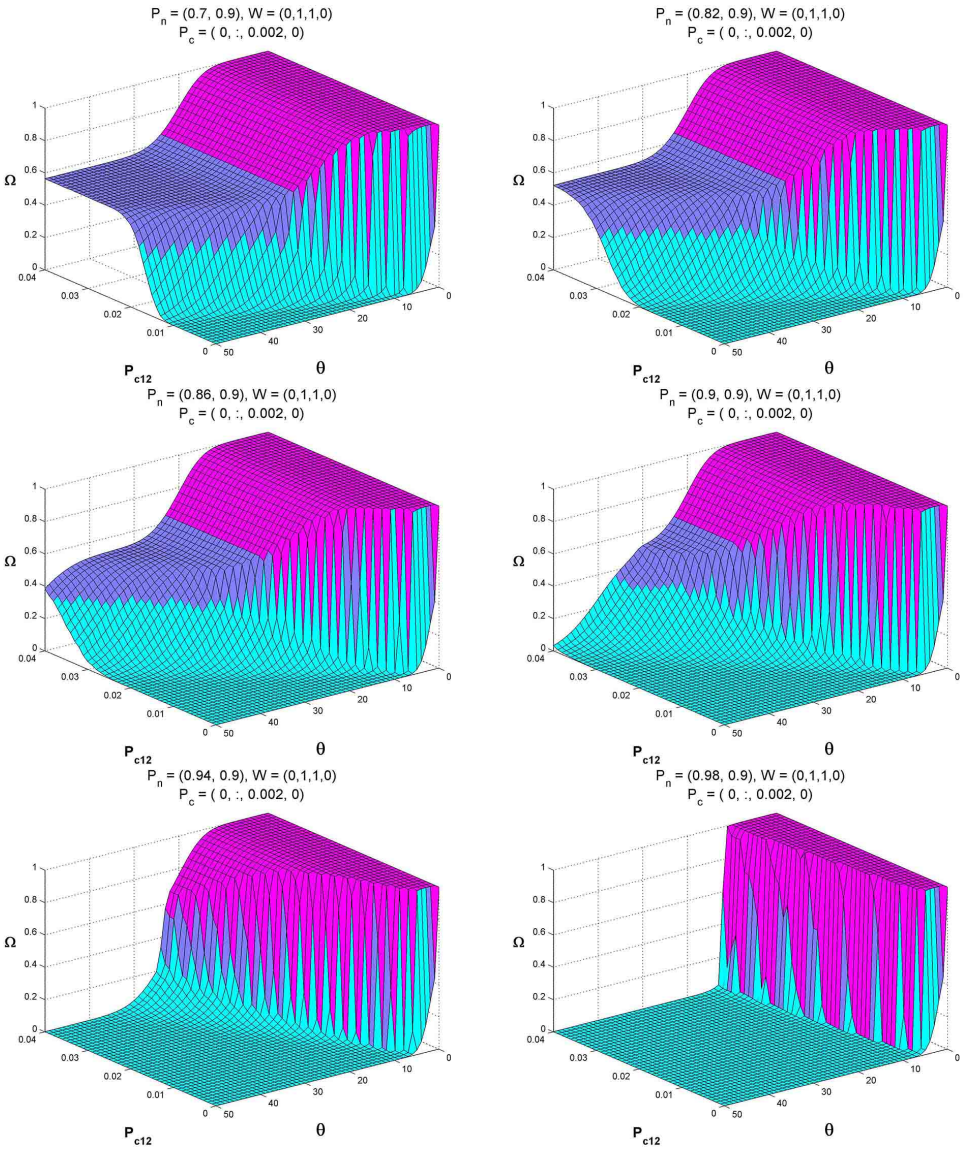

Figure 4.31: Two societies with $\mathbb{W} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ with $P_{n_2} = 0.9$.

4.4.8 Other cases

What about cases left out in these observations? The variation in question in the last section has been the weight matrix $\mathbb{W}$. Therefore we have that the total set of possible $\mathbb{W}$ is the linear space of matrices of dimension 2, so even if we had no intuition about the phenomena studied, we have already covered a base of that space in the following elements:

$$a = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \; b = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \; c = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; d = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Using this we can now get some intuition of what happens with an arbitrary case, for example $\mathbb{W}_1 = \left(\begin{smallmatrix} 2 & 0 \\ 2 & 1 \end{smallmatrix}\right)$, by writing it as a function of linear combinations of the proposed base of the space:

$$\mathbb{W}_1 = 2 \cdot b \; + \; c \; - \; a \tag{4.1}$$

The behavior of the diffusion with the matrix $\mathbb{W}_1$ can be predicted given that we understand what happens with the base elements.

4.5 N Societies

The program created for this study can handle N societies in total, each of them with unique parameters. Also the observations made about the two societies scenario also apply to N societies; there was nothing intrinsic to $\lambda = 2$ about the things observed or discussed. Therefore this analysis could be made for any number of societies using the tools developed for this study.

As an example of this and to give an intuition about how these phenomena scale, we studied a three society case. This case involves the following weight matrix $\mathbb{W}$ and population vector $\vec{m}$:

$$\mathbb{W} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \vec{m} = \begin{pmatrix} 10,000 & 10,000 & 10,000 \end{pmatrix}$$

The results are included in figure 4.32. We can see three zones of adoption now: S_1 adopting alone, S_1 and S_2 adopting together and all three of them with full adoption.

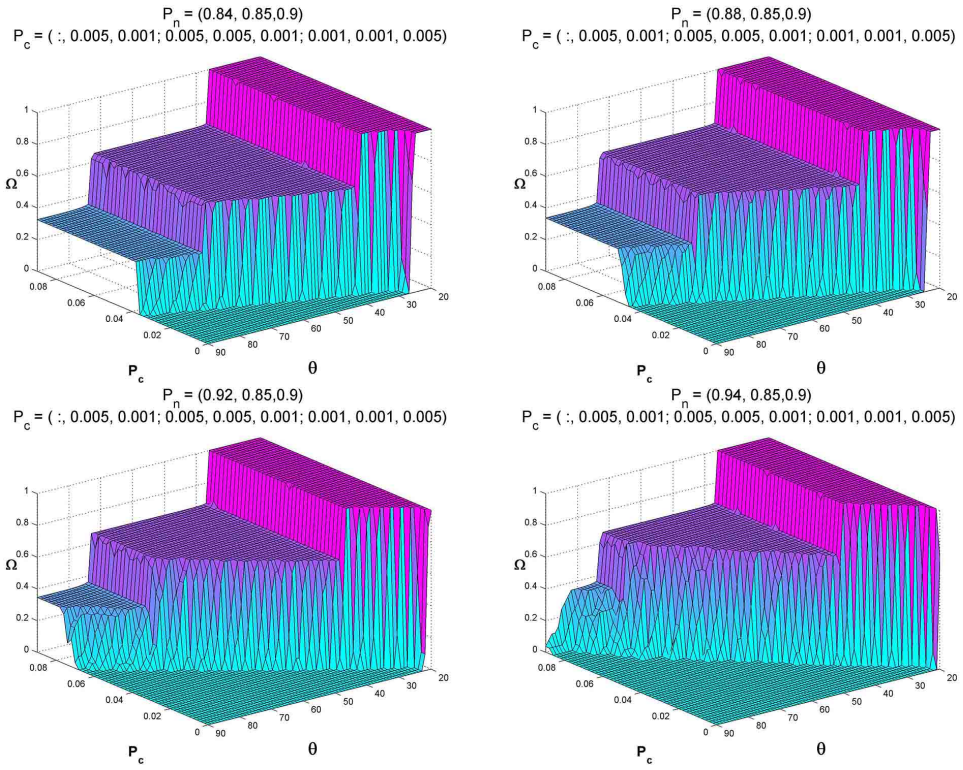

Figure 4.32: Three societies with $\mathbb{W} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ and $P_{n_2} = 0.8$, $m = (10^4, 10^4, 10^4)$.

Also, as we increase P_{n_2} (therefore decreasing α), the zone of non-adoption increases in size and as happened in figures like 4.17, 4.25, 4.27 etc., the plane of "high growth" rotates so that higher connectivity values are needed to maintain the same adoption as before.

Finally, to emphasize how the program can handle different population sizes and any arbitrary weight matrix, we simulate the diffusion process with the following $\mathbb{W}$ and $\vec{m}$:

$$\mathbb{W} = \begin{bmatrix} 1.8 & 0.1 & 0.5 \\ 0.6 & 3.1 & 0.0 \\ 0.0 & 0.3 & 2.2 \end{bmatrix} \quad \vec{m} = \begin{pmatrix} 5,000 & 10,000 & 15,000 \end{pmatrix}$$

The results from the simulation with these values are included in figure 4.33 along with the results of figure 4.32 to allow for easy comparison; as usual both columns have images plotted with the same parameters (except $\mathbb{W}$ and $\vec{m}$). The following can be noted:

1. The plateaus found are of different size given that we changed the population size of the societies.

2. We now notice five different plateaus instead of the three we had before. As before we can see society one fully adopting (a), societies S_1 and S_2 with full adoption (b) and all societies with $\Omega = 1$ (c), but now we can also see a region with the adoption of both S_2 and S_3 (d) and a zone with adoption from only S_2 (e). These regions did not appear in the first scenario.

3. Since we changed the weight matrix to incorporate several entries with values $\mathbb{W}_{ij} \in [0, 1)$, most of the adoption plateaus changed:

   - The adoption for S_1 and S_2 simultaneously (b) shrank in size, i.e. it is now harder for those societies to fully adopt the innovation.

   - Possibly as a consequence of the last point, plateau a increased in size.

4. All the zones of high growth now happen slower. Notice how these regions have several points in them, in contrast to the one or none in the original case. This creates a situation where more needs to be invested to push the populations to adopt the innovation.

In order to appreciate the different heights in another perspective we include figure 4.34 where we can see contour plot versions of the plots in figure 4.33. Here we can see that the growth between plateaus happens at a slower rate than in other cases.

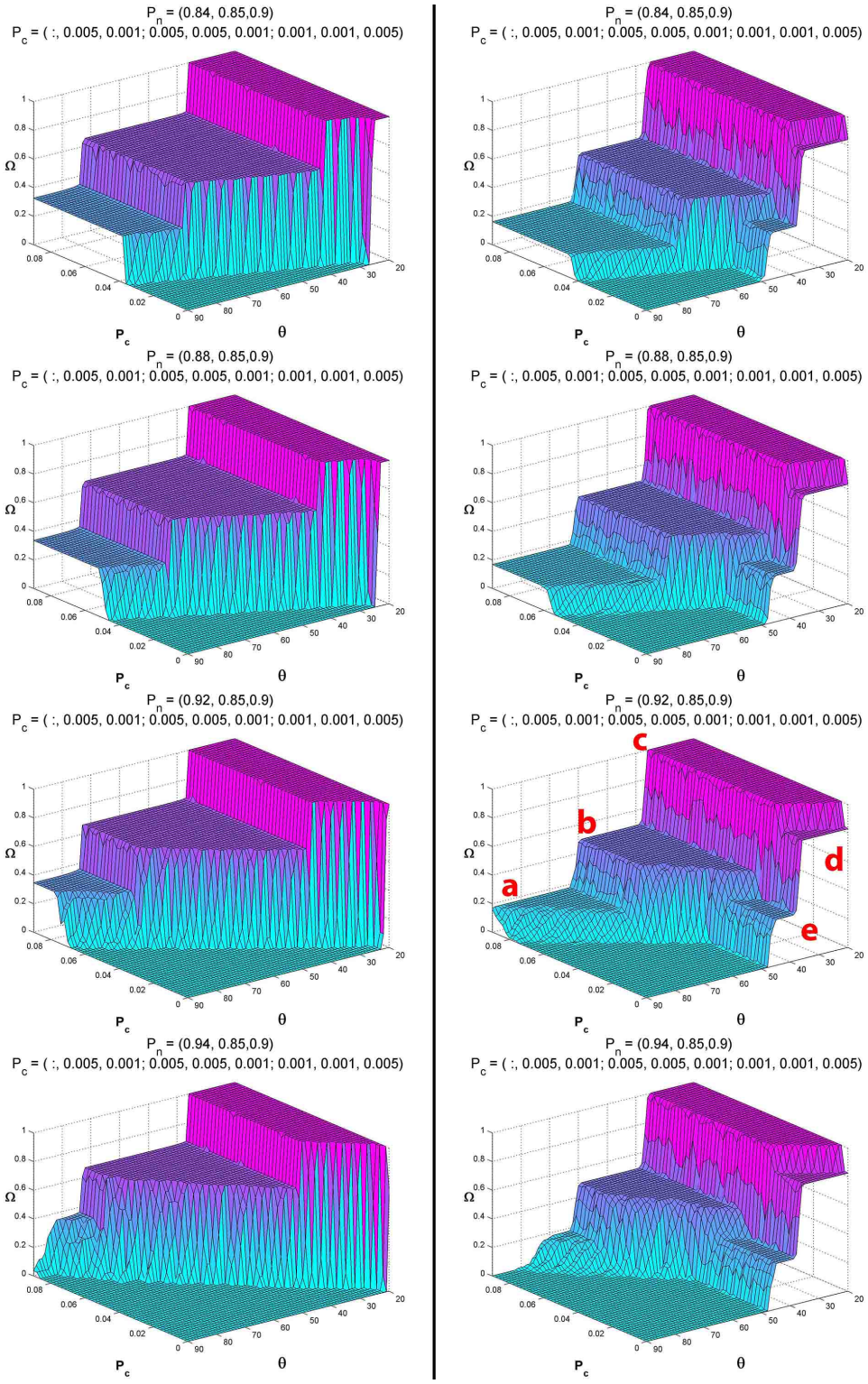

Figure 4.33: Comparison between three societies with $\mathbb{W} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ and $\mathbb{W} = \begin{pmatrix} 1.8 & 0.1 & 0.5 \\ 0.6 & 3.1 & 0.0 \\ 0.0 & 0.3 & 2.2 \end{pmatrix}$; population sizes also vary.

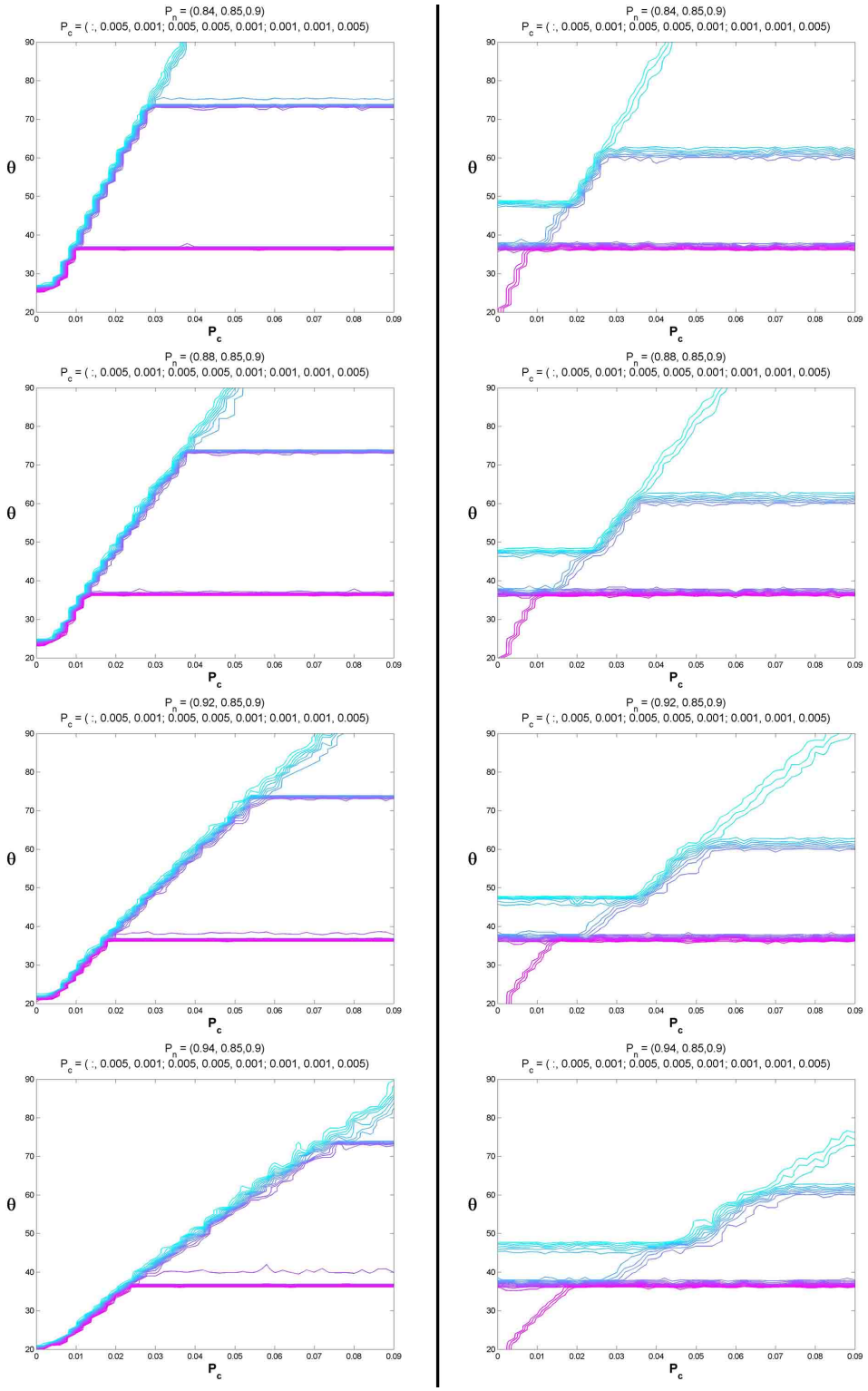

Figure 4.34: Contour plots of the results presented in figure 4.33.

CHAPTER 5

CONCLUSION

In this thesis we studied, with increasing complexity, the diffusion of behaviors in a random network using a generalization of the Bootstrap Percolation model that consists of the following two complications: λ societies each with different parameters, making the overall social network heterogeneous (if $\lambda \geq 2$); and arbitrarily weighted relations between those societies. The studies were divided into $\lambda = 1$, $\lambda = 2$ and $\lambda = 3$:

1. $\lambda = 1$: For one society we first visualized how the diffusion spreads for a case with a lattice network and then in the case of a general adjacency matrix. We also studied how the proportion of adoption Ω behaves as a function of θ and α and also observed how the connectivity P_c affects this behavior. Then we studied the critical threshold as a function of population size and concluded it was a linear relation. Finally we obtained 3-D plots of the process Ω *vs.* $(P_c,\ \theta)$ to summarize most of the previous results.

2. $\lambda = 2$: For the two society case we studied the canonical case $\mathbb{W} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and obtained several different plots to observe the behavior of the process. Zooms of some plots were also obtained to highlight the complexity in a smaller scale. Also in the two society case we analyzed multiple weight matrices and the effect they had on the diffusion. We did this by comparing all these different weights with the canonical weight previously mentioned.

3. $\lambda = 3$: Finally for a three society case we prove that the created tool is general. We simulated a three society case with an arbitrary weight matrix and compared it to the canonical weight matrix of dimension three.

We were successful in creating a program to analyze the diffusion of innovations in contexts with any number of societies and an arbitrary weight matrix to alter the relations between them.

We believe that this tool could prove to be very helpful with the aid of sociological statistics. Given enough information being fed to the program about real world scenarios, this code could be used to predict the behavior of the phenomenon in question but with different parameters than the ones provided. This simulation would obviously be more reliable if the simulated points are in close proximity to the ones empirically observed.

5.1 Possible Future Studies

Several small applications of the current program were left untested:

1. Cases where the cross connectivity between societies was higher than the internal connectivity. It was partially covered by making an analysis of the weight case $\mathbb{W} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ but further studies could have been done.

2. Proportion of adoption in general as a function of population size, specifically when studied in the context of different weight scenarios. For a given set of parameters, particularly if we are close to the critical threshold, increasing population size might make the society not trigger a global cascade, for example. Studying how sensitive different scenarios are to changes in population size might be interesting.

Other than this, possible improvements to the program could be made to study more complicated scenarios such as the ones mentioned in section 3.3: non-progressive cases, hybrid thresholds, percentage thresholds, more than two behaviors, individual randomized thresholds etc.

CHAPTER 6

REFERENCES

[1] D. Watts and P. Dodds, "Influentials, networks, and public opinion formation," *Journal of Consumer Research*, vol. 34, pp. 441–458, December 2007.

[2] E. Rogers, *Diffusion of Innovations*. New York, NY: The Free Press, 2003.

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

[4] W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. 452–473, 1977.

[5] J. Chalupa, G. Reich, and P. Leath, "Bootstrap percolation on a Bethe lattice," *Journal of Physics C*, vol. 12, pp. L31–L35, 1979.

[6] J. Adler and U. Lev, "Bootstrap percolation: Visualizations and applications," *Brazilian Journal of Physics*, vol. 33, no. 3, pp. 641–644, September 2003.

[7] H. Amini, "Bootstrap percolation and diffusion in random graphs with given vertex degrees," *Electron. J. Comb.*, vol. 17, Feb. 2010.

[8] H. Amini, M. Draief, and M. Lelarge, *Network Control and Optimization*. Springer Berlin Heidelberg, 2009, ch. "Marketing in a Random Network," pp. 17-25.

[9] M. Granovetter, "Threshold models of collective behavior," *The American Journal of Sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.

[10] J. Coleman, E. Katz, and H. Menzel, *Medical Innovation: A Diffusion Study*. New York, NY: Bobbs-Merril Co., 1966.

[11] D. Strang and S. Soule, "Diffusion in organizations and social movements: From hybrid corn to poison pills," *Annual Review of Sociology*, vol. 24, pp. 265–290, 1998.

[12] S. Morris, "Contagion," *Review of Economic Studies*, vol. 67, no. 1, pp. 57–78, 2000.

[13] J. Kleinberg, *Algorithmic Game Theory*. Cambridge, UK: Cambridge University Press, 2007, ch. "Cascading behaviour in networks: Algorithmic and economic issues."

[14] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," presented at the Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, 2006.

[15] M. Andre et al., "Transmission network analysis to complement routine tuberculosis contact investigations," *American Journal of Public Health*, vol. 97, no. 3, pp. 470–477, 2007.

[16] M. Lelarge, *Internet and Network Economics*. Springer: Berlin Heidelberg, 2008, ch. "Diffusion of Innovations on Random Networks: Understanding the Chasm," pp. 178–185 (from Lecture Notes in Computer Science, volume 5385).

APPENDIX A

MAIN MATLAB CODE

All the following files can be found at https://github.com/F-Dominguez/Diffusion-Matlab-Code.

A.1 Main File

```
%************************************************
%* Percolation simulation in heterogeneous media *
%* Final Version Programmed April 4, 2012        *
%* By Francisco Dominguez                        *
%************************************************

tic; close all; clc

%% Define Parameters

% n:            Dimension of Adjacency matrix.
% Thresh_Begin: Initial value of Threshold.
% Thresh_Intv:  Increment of Threshold for variation.
% Thresh_End:   Final Value of Threshold.
% Repeat:       Number of times percolation experiment is re-
%               peated to obtain the average of the proportion
%               of adoption.
% Pseed :       Probability of initial activity, or initial
%               adoption of new behaviour. Pn = 1-Pseed.
% Pconnect:  Probability of connectedness for the Adjacency
%            Matrix; this is the probability that connections
%            between members will arise.
% NumPop:    Number of societies.
% PopSize:   Vector stating the size of all societies.
```

```
% PopSame:   Vector stating if we need the submatrices (of the
%       Adjacency matrix) representing interaction between
%       societies to be equal; for example if we need the Adja-
%       cency matrix of society i interacting with j to be equal
%       to society j interacting with i. Anything other than
%       'Same' will make them be generated differently, i.e.
%       will get us non symmetric relations between
%       populations.
% PopSym:    Matrix stating if the submatrices of the Adjacency
%       matrix are generated symmetrically or not. Anything
%       other than Sym will yield non symmetric random matrices,
%       i.e. we won't have a symmetric relation inside
%       societies.
% Weight_Pop: Vector introducing weight to the utility function
%       of individuals in a particular society. This is to vary
%       the importance of adoption of a certain society in a
%       given individual. This is also a way to make a relative
%       variation of threshold amongst the societies.
%
% It is possible for this code to simulate percolation for any
% number of societies. One must define a Pn vector with dimen-
% sion equal to this number. Also one must define a Pconnect
% matrix with values for every interacting society; NumPop^2.
% Pn_Mat and Pconnect_Mat are 3xNumPop matrices. First row de-
% fines the initial values of Pn or Pconnect respectively, se-
% cond row gives the increment for simulation and third row
% gives the final value for simulation. The code takes all po-
% ssible permutations of values introduced in this matrices.
%  This last step is achieved in the function Variation_Perm.
%-----------------------------------------------------------------

% One Society

Thresh_Begin = 6;
Thresh_Intv = 1;
Thresh_End = 6;
Repeat = 500;

Pn_Mat = cat(3,.70,.001,.99);
Pconnect_Mat = cat(3,.002,.00005,.002);

NumPop = 1;
PopSize = [10000];
```

```
PopSame = {'Same'};
PopSym = {'Diff'};
Weight_Pop = [1];

%-------------------------------------------------------------
%% Two Societies

Thresh_Begin = 0;
Thresh_Intv = 1;
Thresh_End = 100;
Repeat = 3;

Pn_Mat = cat(3,[0.70 0.80],...
               [0.05 0.10],...
               [0.95 0.90]);
Pconnect_Mat = ...
cat(3,[0.000 0.001;0.001 0.010],...
      [0.005 0.001;0.001 0.010],...
      [0.300 0.001;0.001 0.010]);
NumPop     = 2;
PopSize    = [10000 10000];
PopSame    = {'Same','Same'};
PopSym     = {'Diff','Diff';'Diff','Diff'};
Weight_Pop = [1 -1;-1 1];

%-------------------------------------------------------------
% Three Societies

Thresh_Begin = 20;
Thresh_Intv = 1;
Thresh_End = 90;
Repeat =10;

Pn_Mat = cat(3,[0.84 0.85 0.90],...
               [0.02 0.05 0.05],...
               [0.94 0.85 0.90]);
Pconnect_Mat = ...
cat(3,...
[0.000 0.005 0.001;0.005 0.005 0.001;0.001 0.001 0.005],...
[0.002 0.001 0.001;0.001 0.005 0.001;0.001 0.001 0.005],...
[0.090 0.005 0.001;0.005 0.005 0.001;0.001 0.001 0.005]);
NumPop     = 3;
PopSize    = [10000 10000 10000];
```

```
PopSame    = {'Same','Same','Same'};
PopSym     = {'Diff','Diff','Diff';...
              'Diff','Diff','Diff';...
              'Diff','Diff','Diff'};
Weight_Pop = [1 1 1;1 1 1;1 1 1];

%-------------------------------------------------------------
%% Simulation

[Pconnect_Perm] = Variation_Perm(Pconnect_Mat);
[Pn_Perm]       = Variation_Perm(Pn_Mat);
[Average,MSE]   = Variation_MP(Thresh_Begin,Thresh_Intv,...
Thresh_End,Pconnect_Perm,Pn_Perm,Repeat,NumPop,PopSize,...
PopSame,PopSym,Weight_Pop);

elapsedtime = toc;

%-------------------------------------------------------------
%% GUI

% To execute the Graphical User Interface, either uncomment the
% following line or execute it in the command prompt. This will
% load the data into the interface and allow for dynamical ob-
% servation of the results.

% test1(Pconnect_Perm,Pn_Perm,Thresh_Begin:Thresh_End,...
Average,MSE,NumPop)
```

A.2 Nested Files

```
%************************************************
%* Variation_Perm                               *
%*                                              *
%* Calculation and reformatting of all possible *
%* permutations asked in the program.           *
%* By Francisco Dominguez                       *
%************************************************

%This code transforms the Pconnect_Mat and Pn_Mat into a useful
%form that encompasses all possible permutations of values of
%Pconnect or Pn for the program to execute the simulation.
%
%Mat_Intv : The input matrix in question; it can be Pconnect_Mat
%       or Pn_Mat.
%Intv_Nums: Calculates the total number of steps for each para-
%       meter, i.e. the total number of steps that the program
%       will do for a particular parameter of a population, for
%       example Pconnect11, Pconnect12 etc.
%Num_Cols : Is the total number of steps for all parameters, en-
%       compassing all possible Pconnect or Pn depending on the
%       input matrix.
%ForVar   : This is the output of the function, It will be
%       Pconnect_Perm or Pn_Perm depending on the input matrix.
%-------------------------------------------------------------

function [ForVar] = Variation_Perm(Mat_Intv)

%Calculation of number of increments.
Intv_Nums = (round((Mat_Intv(:,:,3)-Mat_Intv(:,:,1))./...
Mat_Intv(:,:,2))+1)';

%Total number of increments.
Num_Cols = prod(prod(Intv_Nums));

%Preallocation for speed.
ForVar = zeros(numel(Mat_Intv)/3,Num_Cols);

%Obtaining the permutations in form of a matrix.
for i = 1:size(Mat_Intv,1)
```

```
    for j = 1:size(Mat_Intv,2)
        RowNum        = j + (i-1) * size(Mat_Intv,1);
        Ent_Block_Len = prod(Intv_Nums(RowNum:end));
        Block_Num_Len  = Ent_Block_Len/Intv_Nums(j,i);
        Block = zeros(1,Ent_Block_Len);
        for z = 1:Ent_Block_Len/Block_Num_Len
            Block_Part_val = Mat_Intv(i,j,1) + (z-1)*...
            Mat_Intv(i,j,2);
            Block((1+(z-1)*Block_Num_Len):Block_Num_Len*z) = ...
                repmat(Block_Part_val,1,Block_Num_Len);
        end
        ForVar(RowNum,:) = ...
        repmat(Block,1,Num_Cols/Ent_Block_Len);
    end
end
```

```
%*****************************************************
%* Variation_MP                                      *
%* Nested 'for' loop for every combination of values *
%* This is fed to Init_Pop which creates the Nvect   *
%* and Adjacency Matrix, and to Evolve_MPopulation   *
%* which carries out the evolution.                  *
%* By Francisco Dominguez                            *
%*****************************************************

%This code is essentially a nested 'for' loop that executes the
%functions 'Init_Pop' and 'Evolve_MP' for all possible combina-
%tions of values input in Pn_Mat and Pconnect_Mat and then re-
%formatted into Pn_Perm and Pconnect_Perm.
%
%Pconnect_Len, Pn_Len and Thresh_Len are the number of values of
%        Pconnect, Pn and Threshold that are desired to be simu-
%        lated by the program.
%WEight_Pop_Len: In the current version of the code, it is not
%        possible to vary the desired values of the weight for
%        the utility functions related to threshold in each so-
%        ciety. This part is therefore not used but was included
%        for a future version of the program.
%Proportion: This value is the main result and object of study
%        in our simulations. It represent the proportion of non-
%        adopters (defined as the initial non adopting fraction
%        of the population) that ended up adoptting the beha-
%        viour. Therefore it will be a value of 1 if everyone
%        adopted at the end or 0 if no one did, with every va-
%        lue in between as a possible outcome.
%Regarding the counters, zz counts the total number of steps,
%this is used in the function 'timebar' which is a modified ver-
%sion of the Matlab function waitbar to also incorporate time
%remaining. The counter r is for the number of Repeat, c is for
%Pconnect, s is for Pseed or Pn and t is for Threshold.
%
%For each combination of numbers, the Adjacency Matrix using
%Pconnect and the Nvect (vector of states for each member of the
%society) is calculated using Init_Pop. Once those matrices are
%obtained, the function Evolve_MPopulation carries the evolution
%of the percolation through its final state and then the final
%Proportion is saved in the 4 dimensional array (For repeat,
%connect, seed and threshold). Finally the average is calculated
%over all repetitions.
```

```
%
%The counters in this code are:
%
%zz: Counter for total steps.
%c:  Counter for Pconnect steps.
%s:  Counter for Pseed or Pn steps.
%t:  Counter for Threshold steps.
%-------------------------------------------------------------

function [Average,MSE,AllPerms] = Variation_MP(Thresh_Begin,...
Thresh_Intv,Thresh_End,Pconnect_Perm,Pn_Perm,Repeat,NumPop,...
PopSize,PopSame,PopSym,Weight_Pop)

%Length calculation.
Pconnect_Len = size(Pconnect_Perm,2);
Pn_Len = size(Pn_Perm,2);
Thresh_Len = (Thresh_End-Thresh_Begin)/Thresh_Intv +1;

% Following line reserved for following version incorporating
% weight variation.
% Weight_Pop_Len = numel(Weight_Pop);
Weight_Pop_Len =1;

%Preallocation for speed.
Proportion = zeros(Thresh_Len,Pn_Len,Pconnect_Len,Repeat,...
      Weight_Pop_Len);
AllPerms   = zeros(1 + size(Pconnect_Perm,1) + ...
      size(Pn_Perm,1),Thresh_Len*Pn_Len*Pconnect_Len);
Par_Connect = cell(NumPop,NumPop);

h = timebar(0,sprintf('%3.1f Percent Done',0));
zz=0;
for r = 1:Repeat
    c=1;
    for Pconnect = Pconnect_Perm
        [Adj_Mat]  =  Init_Pop(NumPop,PopSize,...
                      reshape(Pconnect,NumPop,...
                      []),PopSame,PopSym);
        %Total number of connections.
        Tot_Connect = sum(cell2mat(Adj_Mat),2);
        for Active_Pop = 1:NumPop
            for Connected_Pop = 1:NumPop
                 Par_Connect{Active_Pop, Connected_Pop} =...
```

```
                    sum(Adj_Mat{Active_Pop, Connected_Pop},2);
            end
        end
        s=1;
        for Pn = Pn_Perm
            %Network vector. It has 1's for non adopters, 0's
            %for adopters.
            [NvectO]  =  Init_Pop(NumPop,PopSize,Pn);
            %Initial Number of non-adopters.
            IN = sum(cell2mat(NvectO));
            t = 1;
            for Threshold = Thresh_Begin:Thresh_Intv:Thresh_End
                Nvect = NvectO;
                [NAf] =  Evolve_MPopulation(Weight_Pop,NumPop,...
                         Adj_Mat,Nvect,Tot_Connect,...
                         Par_Connect,Threshold,PopSize);
                %Proportion of non-adopters that adopted the ...
                %behaviour.
                Proportion(t,s,c,r) = 1-(NAf/IN);
                t      = t + 1;
                zz = zz+1;
                steps = ...
                    (zz)/(Repeat*Thresh_Len*Pn_Len*Pconnect_Len);
                timebar(steps,h,...
                    sprintf('%3.1f Percent Done',steps*100));
            end
            s=s+1;
        end
        c=c+1;
    end

end
%Average of Proportion over r runs.
Average = mean(Proportion,4);
%MSE of each set of numbers.
MSE     = std(Proportion,1,4);
close(h);
```

```
%************************************************************
%* Init_Pop
%*
%* Generation of General Adjacency Matrix and Population Vector.
%* By Francisco Dominguez
%************************************************************

%This function generates the Adjacency Matrix if the Pval in
%question is Pconnect or the Nvect (vector of population con-
%taining 0 for new behaviour or 1 for original behaviour) if
%the Pval is the Pn.
%
%The matrix PopSym dictates wether a submatrix of the adjacency
%matrix (AdjM) is symmetrical or not. In this program, the AdjM
%for each society and the AdjM for the relationship between so-
%cieties are all included in the same general AdjM. The same is
%done for the Nvect which encompasses the Nvect of each indivi-
%dual society. Therefore, each component of the general AdjM
%dictates the relation of individual societies or between two
%particular societies.
%PopSym determines if those relations are symmetrical. If so
%then the matrix is generated using a random sparse symmetrical
%matrix command instead of just sparse symmetric matrix. Note
%that if symmetric matrix is desired, then it must be a square
%matrix i.e. PopSize(i) = PopSize(j). In other words PopSym dic-
%tates if we have symmetric adjacency matrices for societies;
%AdjM_11 is symmetric.
%
%The vector PopSame, on the other hand dictates whether or not
%interaction within societies are symmetrical between them; for
%example if we need the Adjacency matrix of society i inter-
%acting with j to be equal to society j interacting with i;
%AdjM_13 = AdjM_31. Anything other than 'Same' will  make them
%be generated differently. Note that if Same is desired, then
%the Pconnect values must be equal too.
%------------------------------------------------------------

%OutVar: This output will be either AdjM or Nvect, depending on
%         the input Pval.
%Pval  : Can be Pn or Pconnect, if Pn then OutVar will be Nvect,
%        if Pconnect then OutVar will be AdjM.
function [OutVar] = Init_Pop(NumPop,PopSize,Pval,PopSame,PopSym)
```

```
%In the following case the program is considering Pconnect.
if nargin > 3
    OutVar     = cell(NumPop);
    k          = 1;
    for i = 1:NumPop
        for j = 1:NumPop
            %If Same is reported on the vector PopSame and
            %the inserted values are equal.
            if i > j && strncmpi('Same',PopSame{k},4) && ...
                    Pval(i,j) == Pval(j,i)
                %Then the Adjacency matrix is the same.
                OutVar{i,j} = OutVar{j,i}';
                k          = k + 1;
            else
                if i > j
                    k = k + 1;
                end
                %If the entry for the (i.j) adjacency com-
                %ponent in the PopSym matrix starts with Sym
                %and the population size of those 2 societies
                %is equal then use symmetric.
                if strncmpi('Sym',PopSym{i,j},3) &&...
                    PopSize(i) == PopSize(j)
                    OutVar{i,j} = ...
                        spones(sprandsym(PopSize(i),Pval(i,j)));
                else
                    OutVar{i,j} = spones(sprand...
                        (PopSize(i),PopSize(j),Pval(i,j)));
                end
            end
        end
    end
%In the following case the program is considering Pn.
else
    OutVar = cell(NumPop,1);
    for i = 1:NumPop
        OutVar{i} = rand(PopSize(i),1)<Pval(i);
    end
end
```

```
%*********************************************************
%* Evolve_MPopulation                                    *
%* Core of the program, the percolation is simulated here. *
%* By Francisco Dominguez                                *
%*********************************************************

%This is the core of the Percolation or cascade simulation. It
%applies at each step, the rule that if a member has more or
%equal than Threshold neighbors with the new behaviour (which
%is represented by 0) then it adopts that behaviour from the
%original one (represented by 1). The programs stops if at any
%given cycle, no new individuals adopted the new behaviour, in
%which case all of them adopted it or there was a cluster that
%wasn't connected enough to the rest of the network.
%
%Active_Pop: Refers to the current population being evolved. The
%   Active_Pop refers to 'rows' in the Adjacency matrix's sub
%   matrix configuration. This means that if we have Active_Pop
%   = 1 in a 3 society configuration, we are taking into account
%   the submatrices referring to the interaction of society 1
%   with itself, with society 2 and society 3.
%
%Connected_Pop: Refers to 'columns' in the Adjacency matrix's
%   sub matrix configuration.
%
%With both of these numbers the program does the product of Ad-
%jacency matrix and Vector of Population by parts to then report
%the vector of population at the next step.
%
%The weight parameter is considered here. Although the threshold
%is the same throughout the societies, each assigns a weight on
%their links (relations) with each other society, and therefore
%we can manipulate how important relations are between interac-
%ting societies. If we set the weight to 0, no feedback will
%occur when behaviours change in the interacting societies.
%We can also set it as negative and therefore, it will have the
%effect of making the adoption of the new behaviour undesirable
%for a certain population if another population is adopting it.
%---------------------------------------------------------------

function [NAf] =  Evolve_MPopulation(Weight_Pop,NumPop,...
    Adj_Mat,Nvect,Tot_Connect,Threshold,PopSize)
%In the following code, Active_pop refers to 'rows' in the AdjM,
```

```
%while Connected_Pop is used to denote 'columns' in the AdjM.
%These rows and columns are not of scalars but of submatrices,
%referring to particular societies. Therefore 13 is the interac—
%tion if society 1 with society 3, and therefore an entire matrix
%itself.

dNA = 1;
Connect2one = cell(1,NumPop);
Connect2zero = cell(1,NumPop);
while dNA
   for Active_Pop = 1:NumPop
       for Connected_Pop = 1:NumPop
           if Weight_Pop == ones(NumPop)
                 if Connected_Pop == 1
                     %Vector stating the amount of connec—
                     %tions to 1's. This is basically a part
                     %of the utility function for members of
                     %Active_Pop.
                     Connect2one{Active_Pop} = ...
                       Adj_Mat{Active_Pop, Connected_Pop}*...
                       Nvect{Connected_Pop};
                 else
                     %Vector stating the amount of connec—
                     %tions to 1's.
                     Connect2one{Active_Pop} = ...
                       Connect2one{Active_Pop} + ...
                       Adj_Mat{Active_Pop, Connected_Pop}*...
                       Nvect{Connected_Pop};
                 end
           else
               switch Weight_Pop(Active_Pop, Connected_Pop)
                 case 0,
                    if Connected_Pop == 1
                       Connect2zero{Active_Pop} = ...
                        zeros(PopSize(Connected_Pop),1);
                     end
                 case 1,
                    if Connected_Pop == 1
                       Connect2zero{Active_Pop} = ...
                        Par_Connect{Active_Pop,Connected_Pop}...
                        — (Adj_Mat{Active_Pop,Connected_Pop}*...
                        Nvect{Connected_Pop});
                    else
```

```
                    Connect2zero{Active_Pop} = ...
                     Connect2zero{Active_Pop} + ...
                     Par_Connect{Active_Pop,Connected_Pop}...
                     - (Adj_Mat{Active_Pop,Connected_Pop}*...
                     Nvect{Connected_Pop});
                end
            otherwise,
                if Connected_Pop == 1
                    Connect2zero{Active_Pop} = ...
                     Weight_Pop(Active_Pop,Connected_Pop)*(...
                     Par_Connect{Active_Pop,Connected_Pop}...
                     - (Adj_Mat{Active_Pop,Connected_Pop}*...
                     Nvect{Connected_Pop}));
                else
                    Connect2zero{Active_Pop} = ...
                     Connect2zero{Active_Pop} + ...
                     Weight_Pop(Active_Pop,Connected_Pop)*(...
                     Par_Connect{Active_Pop,Connected_Pop}...
                     - (Adj_Mat{Active_Pop,Connected_Pop}*...
                     Nvect{Connected_Pop}));
                end
            end
        end
    end
end
if Weight_Pop == ones(NumPop)
    %If number of connections to 0's is less than m,then
    %the behaviour stays the same.
    Amask = ((Tot_Connect-cell2mat(Connect2one'))<Threshold);
    %Initial Number of nonadopters at each step.
    NAi = sum(cell2mat(Nvect));
    %Percolation step.
    Nvect = mat2cell(cell2mat(Nvect).*Amask,PopSize);
    %Final Number of nonadopters at each step.
    NAf = sum(cell2mat(Nvect));
    %Number of adopters gained.
    dNA = NAi-NAf;
else
    Amask = ((cell2mat(Connect2zero'))<Threshold);
    NAi = sum(cell2mat(Nvect));
    Nvect = mat2cell(cell2mat(Nvect).*Amask,PopSize);
    NAf = sum(cell2mat(Nvect));
    dNA = NAi-NAf;
```

```
    end
end
```

A.3 Remaining Time Program

```
%**************************************************
%* Timebar                                        *
%* Modified waitbar function of Matlab to estimate *
%* time remaining.                                *
%**************************************************

%This is a modified version of MATLAB's waitbar function with
%the added functionality of displaying time estimation. It was
%obtained from the following webpage:
%(www.mathworks.com/matlabcentral/
%fileexchange/22161-waitbar-with-time-estimation)
%after which it was very slightly modified to fit into the
%rest of the program.

function h = timebar(X,varargin)
%  TIMEBAR a modified version of MATLAB's waitbar function.
%-------------------------------------------------------------
%    H = WAITBAR(X,'message') creates and displays a waitbar of
%      fractional length X.  The handle to the waitbar
%      figure is returned in H. X should be between 0 and 1.
%
%    WAITBAR(X) will set the length of the bar in the most re-
%      cently created waitbar window to the fractional length X.
%
%    WAITBAR(X,H) will set the length of the bar in waitbar H
%      to the fractional length X.
%
%    WAITBAR(X,H,'message') will update the message text in
%      the waitbar figure, in addition to setting the fractional
%      length to X.
%
%    WAITBAR is typically used inside a FOR loop that performs a
%      lengthy computation.
%
%    Example:
%       h = waitbar(0,'Please wait...');
%       for i=1:1000,
%           % computation here %
%           waitbar(i/1000,h)
```

```
%       end
%
% NOTES:
% - This program produced with heavy modification of Chad
% English's timebar function.  The update was designed to
% receive input identically to MATLAB's waitbar function to
% allow for interchangeability.
%
% - This program does not apply the property values that the
%  traditional waitbar allows.
%
%--------------------------------------------------------------

% 1 - GATHER THE INPUT
    if nargin == 1;
        h = findobj(allchild(0),'flat','Tag','waitbar');
        message = '';
    elseif isnumeric(X) & ishandle(varargin{1}) & nargin == 2;
        h = varargin{1}; message = '';
    elseif isnumeric(X) & ischar(varargin{1}) & nargin == 2;
        h = []; message = varargin{1};
    elseif isnumeric(X) & ishandle(varargin{1}) & nargin == 3;
        h = varargin{1}; message = varargin{2};
    else
        disp('Error defnining waitbar'); return;
    end

% 2 - BUILD/UPDATE THE MESSAGE BAR
    if  isempty(h) || ~ishandle(h(1));
    h = buildwaitbar(X,message);
    else updatewaitbar(h,X,message); end

%-------------------------------------------------------------
% SUBFUNCTION: buildwaitbar
function h = buildwaitbar(X,message)
% BUILDWAITBAR constructs the figure containing the waitbar

 % 1 - SET WINDOW SIZE AND POSITION
    % 1.1 - Gather screen information
        screensize = get(0,'screensize');  % User's screen size
        screenwidth = screensize(3);       % User's screen width
        screenheight = screensize(4);      % User's screen height
```

```
    % 1.2 - Define the waitbar position
        winwidth = 300;           % Width of timebar window
        winheight = 85;           % Height of timebar window
        winpos = [0.5*(screenwidth-winwidth), ...
            % Position
            0.5*(screenheight-winheight), winwidth, winheight];

% 2 - OPEN FIGURE AND SET PROPERTIES
    wincolor = 0.85*[1 1 1]; % Define window color

    % 2.1 - Define the main waitbar figure
    h = figure('menubar','none','numbertitle','off',...
        'name','0% Complete','position',winpos,'color',...
        wincolor,'tag','waitbar','IntegerHandle','off');

    % 2.2 - Define the message textbox
    userdata.text(1) = uicontrol(h,'style','text','hor',...
        'left','pos',[10 winheight-30 winwidth-20 20],...
        'string',message,'backgroundcolor',wincolor,...
        'tag','message');

    % 2.3 - Build estimated remaining static text textbox
    est_text = 'Estimated time remaining: ';
    userdata.text(2) = uicontrol(h,'style','text',...
        'string',est_text,'pos',[10 15 winwidth/2 20],...
        'FontSize',7,'backgroundcolor',wincolor,...
        'HorizontalAlignment','right');

    % 2.4 - Build estimated time textbox
    userdata.remain = uicontrol(h,'style','text','string','',...
        'FontSize',7,'HorizontalAlignment','left',...
        'pos',[winwidth/2+10 14.5 winwidth-25 20], ...
        'backgroundcolor',wincolor);

    % 2.5 - Build elapsed static text textbox
    est_text = 'Total elapsed time: ';
    userdata.text(3) = uicontrol(h,'style','text','string',...
        est_text,'pos',[10 3 winwidth/2 20],'FontSize',7,...
        'backgroundcolor',wincolor,'HorizontalAlignment',...
        'right');

    % 2.6 - Build elapsed time textbox
    userdata.elapse = uicontrol(h,'style','text','string','',...
```

```
        'pos',[winwidth/2+10 3.5 winwidth-25 20],...
        'FontSize',7,'backgroundcolor',wincolor,...
        'HorizontalAlignment','left');

    % 2.7 - Build percent progress textbox
    userdata.percent = uicontrol(h,'style','text','hor',...
        'right','pos',[winwidth-35 winheight-52 28 20],...
        'string','','backgroundcolor',wincolor);

    % 2.8 - Build progress bar axis
    userdata.axes = axes('parent',h,'units','pixels',...
        'xlim',[0 1],'pos',[10 winheight-45 winwidth-50 15],...
        'box','on','color',[1 1 1],'xtick',[],'ytick',[]);

% 3 - INITILIZE THE PROGESS BAR
 userdata.bar = ...
   patch([0 0 0 0 0],[0 1 1 0 0],'r'); % Bar to zero area
 userdata.time = clock;                % Record the current time
 userdata.inc = clock;                 % Set incremental clock
 set(h,'userdata',userdata)            % Store data in the figure
 updatewaitbar(h,X,message);           % Updates waitbar if X~=0

%-------------------------------------------------------------------
% SUBFUNCTION: updatewaitbar
function updatewaitbar(h,progress,message)
% UPDATEWAITBAR changes the status of the waitbar progress

% 1 - GATHER WAITBAR INFORMATION
 drawnow;                       % Needed for window to appear
 h = h(1);                      % Only allow newest bar to update
 userdata = get(h,'userdata'); % Get userdata from bar figure

    % Check object tag to see if it is a timebar
    if ~strcmp(get(h,'tag'), 'waitbar')
        error('Handle is not for a waitbar window')
    end

    % Update the message
        if ~isempty(message);
            hh = guihandles(h);
            set(hh.message,'String',message);
        end
```

```
% 2 - UPDATE THE GUI (only update if more than 1 sec has passed)
    if etime(clock,userdata.inc) > 1 || progress == 1

    % 2.1 - Compute the elapsed time and incremental time
      % the total elapsed time
      elap = etime(clock,userdata.time);
      % store current
      userdata.inc = clock; set(h,'Userdata',userdata);

    % 2.2 - Calculate the estimated time remaining
        sec_remain = elap*(1/progress-1);
        e_mes = datestr(elap/86400,'HH:MM:SS');
        r_mes = datestr(sec_remain/86400,'HH:MM:SS');

    % 2.3 - Produce error if progress is > 1
        if progress > 1; r_mes = 'Error, progress > 1'; end

    % 2.4 - Update information
     % Update bar
        set(userdata.bar,'xdata',[0 0 progress progress 0])
     % Update remaining time string
        set(userdata.remain,'string',r_mes);
     % Update elapsed time string
        set(userdata.elapse,'string',e_mes);
     % Update progress %
        set(userdata.percent,'string',...
            strcat(num2str(floor(100*progress)),'%'));
        set(h,'Name',[num2str(floor(100*progress)),...
            '% Complete']); % Update figure name

    end
```

APPENDIX B

AUXILIARY PROGRAMS

The following files are not the core of the program but several auxiliary scripts to aid in plotting 2D and 3D graphs, a Graphical User Interface and several other tools.

```
%*****************************************************
%* Visualization                                     *
%* Visual aid for diffusion progress in networks     *
%* Lattice networks and Random networks              *
%* By Francisco Dominguez                            *
%*****************************************************

%This program does the diffusion process in 2 different sce-
%narios. The first scenario is for Lattice Networks where the
%threshold is 4 throughout the population. Relations are with
%first neighbors only.
%
%The second does the same for an arbitrary adjacency matrix.
%In both cases it captures snapshots of the diffusion and then
%produces a video of the contagion.
%------------------------------------------------------------

%% Lattice

L = 1000;
p = 0.95;
m = 3;

dNA = 1;
A = int8(rand(L)<p);
B = (zeros(L,L));
i = 1;
index = 0;
```

```
while dNA
    Aright = circshift(A, [0, 1]);
    Aleft = circshift(A, [0, -1]);
    Adown = circshift(A, [1, 0]);
    Aup = circshift(A, [-1, 0]);
    Amask = int8((Aright + Aleft + Adown + Aup) >= m);
    NAi=sum(sum(A));
    imagesc((1-A)), title({ 'Frame Number: ' num2str(i) }),...
        colormap(gray), axis square;
    set(gca, 'FontSize',12)
    pause(0.04)
    B=B+double(A);
    A=A.*Amask;
    NAf=sum(sum(A));
    dNA=NAi-NAf;
    Speed(i)=dNA;
    if (i >= index)
     filename = strcat('Animation2D_', num2str(index), '.jpg');
     print (filename , '-djpeg', '-r250')
     index = index + 30;
    end
    i=i+1;
end

%% Adjacency Matrix

clear all
clc

while (i <= 70)

    L = 100;
    Pob = L^2;
    Pn = 0.90;
    Pc = 0.00058;
    Threshold = 3;

    dNA = 1;
    i = 1;
    Connect2zero=zeros(Pob,1);
    A = double(rand(L)<Pn);
    Nvect = reshape(A,Pob,1);
```

```
    Adj_Mat=spones(sprand(Pob,Pob,Pc));
    Par_Connect=sum(Adj_Mat,2);
    index = 0;


    while dNA
        Connect2zero = Par_Connect - Adj_Mat * Nvect;
        Amask = (Connect2zero'<Threshold);
        NAi = sum(Nvect);
        Nvect = Nvect.*Amask';
        NAf = sum(Nvect);
        A=reshape(Nvect,[L L]);
        imagesc((1-A)), title({ 'Step Number: ' num2str(i) }),...
            colormap(gray),axis square
        set(gca, 'FontSize',15)
        pause(0.1)
        dNA=NAi-NAf;
        Speed(i)=dNA;
        if (i >= index)
        filename = strcat('Animation2D_',...
                          num2str(index), '.jpg');
        print (filename , '-djpeg', '-r250')
        index = index + 2;
        end
        i=i+1;
    end
end
```

```
%*****************************************************
%* Graph3d                                           *
%* Graphing script for single 3D graphs of Pconnect, *
%* Pn and Threshold versus Proportion of Adoption    *
%* By Francisco Dominguez                            *
%*****************************************************

%This Script is to facilitate the plotting of 3d graphs using
%the results of the simulation. One selects all but one value
%of Pconnect and Pn to be constant and plots the Threshold and
%Proportion of Adoption with this variable Pconnect or Pn. In
%the case of two societies one must select the value amongst 4
%Pconnects and two Pns.
%This program uses Find_Perm_Data to obtain all the permutations
%of the desired values to plot and to arrange them in a format
%that Matlab will use.
%In this program the Pc represents a different Pconnect value
%for a different submatrix while Pn represents a Pn value for a
%different subvector. It is worthwhile to notice that Matlab
%numbers cells in a matrix ordered by column; in a 3x3 matrix the
%numbering will be #1 for the first cell, then below that cell
%#2 and to the right of #1 will be #4.
%-------------------------------------------------------------


%% Data Loading

% clear all
% load('C:\Users\...\Percolation\Results\Run_1.mat')

%-------------------------------------------------------------
%% 1 Society

Pn = 0.95;
conditions = {'Pn'       [1 Pn]};

Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
[PconCols,PnCols,ThreshCols] = ...
                    Find_Perm_Data(conditions,Pconnect_Perm,...
                    Pn_Perm,Thresh_Perm);
Result = Average(ThreshCols,PnCols,PconCols);
ResResult = reshape(Result,[size(Result,1) size(Result,3)]);
```

```
X = Thresh_Begin:Thresh_Intv:Thresh_End;
Y = Pconnect_Mat(1,1,1):Pconnect_Mat(1,1,2):Pconnect_Mat(1,1,3);

[XX,YY] = meshgrid(Y,X);
%(cool, winter, summer, spring, map, autumn(5),etc)
colormap cool(7);
surf(XX,YY,ResResult);
set(gca, 'FontSize',12)
xlim([Pconnect_Mat(1,1,1) Pconnect_Mat(1,1,3)]);
ylim([Thresh_Begin Thresh_End]);
zlim([0 1]);
% view([-127.5 30]);
xlabel('P_c','fontweight','bold','FontSize',16);
ylabel('\theta','fontweight','bold','FontSize',22);
set(get(gca,'ZLabel'),'Rotation',0);
zlabel('\Omega   ','fontweight','bold','FontSize',18);
title({['P_n = (' num2str(Pn) '),   P_c = ( : ),   W = ('...
    num2str(Weight_Pop(1,1)) ') ']},'FontSize',18);
print -djpeg  1-socPn0.95.jpg -r250;
saveas(gcf,'1-socPn0.75.fig');

%-------------------------------------------------------------
%% 2 Societies

% Define Parameters

Pc2 = 0.001;
Pc3 = 0.001;
Pc4 = 0.010;

Pn1 = 0.80;
Pn2 = 0.90;

% Formatting

conditions = {    'Pconnect' [2 Pc2];
                  'Pconnect' [3 Pc3];
                  'Pconnect' [4 Pc4];
                  'Pn' [1 Pn1];
                  'Pn' [2 Pn2]};

Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
[PconCols,PnCols,ThreshCols] = ...
```

```
                    Find_Perm_Data(conditions,Pconnect_Perm,...
                    Pn_Perm,Thresh_Perm);
Result = Average(ThreshCols,PnCols,PconCols);
ResResult = reshape(Result,[size(Result,1) size(Result,3)]);

% Plotting

X = Thresh_Begin:Thresh_Intv:Thresh_End;
Y = Pconnect_Mat(1,1,1):Pconnect_Mat(1,1,2):Pconnect_Mat(1,1,3);

[XX,YY] = meshgrid(Y,X);
%(cool, winter, summer, spring, map, autumn(5),etc)
colormap cool(3);
surf(XX,YY,ResResult);
xlim([Pconnect_Mat(1,1,1) Pconnect_Mat(1,1,3)]);
ylim([Thresh_Begin Thresh_End]);
zlim([0 1]);
view([-127.5 30]);
xlabel('P_c','fontweight','bold','FontSize',16);
ylabel('\theta','fontweight','bold','FontSize',22);
set(get(gca,'ZLabel'),'Rotation',0);
zlabel('\Omega    ','fontweight','bold','FontSize',18);
title({['P_n = (' num2str(Pn1) ', ' num2str(Pn2) '), W = ('...
   num2str(Weight_Pop(1,1)) ',' num2str(Weight_Pop(1,2)) ','...
   num2str(Weight_Pop(2,1)) ',' num2str(Weight_Pop(2,2)) ')'],...
   ['P_c = ( :, ' num2str(Pc2) ', ' num2str(Pc3) ', '...
   num2str(Pc4) ')']},'FontSize',16);
print -djpeg  W1Pn(.80,.90).jpg -r250
saveas(gcf,'W1Pn(.80,.90).fig');

%-------------------------------------------------------------
%% 3 Societies

%Define Parameters

Pc2 = 0.005;
Pc3 = 0.001;
Pc4 = 0.005;
Pc5 = 0.005;
Pc6 = 0.001;
Pc7 = 0.001;
Pc8 = 0.001;
Pc9 = 0.005;
```

```
Pn1 = 0.90;
Pn2 = 0.85;
Pn3 = 0.90;

% Formatting

conditions = {    'Pconnect' [2 Pc2];
                  'Pconnect' [3 Pc3];
                  'Pconnect' [4 Pc4];
                  'Pconnect' [5 Pc5];
                  'Pconnect' [6 Pc6];
                  'Pconnect' [7 Pc7];
                  'Pconnect' [8 Pc8];
                  'Pconnect' [9 Pc9];
                  'Pn' [1 Pn1];
                  'Pn' [2 Pn2];
                  'Pn' [3 Pn3]};

Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
[PconCols,PnCols,ThreshCols] = ...
                    Find_Perm_Data(conditions,Pconnect_Perm,...
                    Pn_Perm,Thresh_Perm);
Result = Average(ThreshCols,PnCols,PconCols);
ResResult = reshape(Result,[size(Average,1) size(Average,3)]);

% Plotting

X = Thresh_Begin:Thresh_Intv:Thresh_End;
Y = 0.000:0.001:0.09;

[XX,YY] = meshgrid(Y,X);
%(winter, summer, spring, parula(5), map, autumn(5),etc)
colormap cool;
surf(XX,YY,ResResult);
view([-127.5 30]);
xlabel('P_c','fontweight','bold','FontSize',16);
ylabel('\theta','fontweight','bold','FontSize',22);
set(get(gca,'ZLabel'),'Rotation',0);
zlabel('\Omega','fontweight','bold','FontSize',18);
title({['P_n = (' num2str(Pn1) ', ' num2str(Pn2)...
    ',' num2str(Pn3) ')'], ['P_c = ( :, ' num2str(Pc2)...
    ', ' num2str(Pc3) '; ' num2str(Pc4) ', ' num2str(Pc5)...
```

```
    ', ' num2str(Pc6) '; ' num2str(Pc7) ', ' num2str(Pc8)...
    ', ' num2str(Pc9) ')']},'FontSize',18);
print -djpeg  run2_3Pn(.90,.85,.90).jpg -r250 ;


figure (2);
contour(XX,YY,ResResult);
```

```
%*****************************************************
%* Movievolution                                     *
%* Graphing script for 3D graphs videos of Pconnect, *
%* Pn and Threshold versus Proportion of Adoption    *
%* By Francisco Dominguez                            *
%*****************************************************

%Similar to Graph3d, this script uses Find_Perm_Data to generate
%multiple graphs of the simulation data. This program then cre-
%ates a video in .avi format to show the evolution of the beha-
%viour of the model when one parameter is varied. Therefore in
%this case two parameters remain variable.
%Each frame in the video is a graph plotted in a similar way to
%Graph3d. This script is particularly useful for analysis of the
%behaviour of the model.
%As in Graph3d Pc represents a different Pconnect value for a
%different submatrix while Pn represents a Pn value for a dif-
%ferent subvector. It is worthwhile to notice that Matlab num-
%bers cells in a matrix ordered by column; in a 3x3 matrix the
%numbering will be #1 for the first cell, then below that cell
%#2 and to the right of #1 will be #4.
%-------------------------------------------------------------


%% 1-Society

numframes = 1+(Pn_Mat(1,1,3)-Pn_Mat(1,1,1))/Pn_Mat(1,1,2);
clear i;

fig1 = figure;
set(fig1,'render','painters')
for i = 1:numframes;
    Pn = Pn_Mat(1,1,1) + (i-1)*Pn_Mat(1,1,2);
    conditions = {'Pn'       [1 Pn]};
    Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
    [PconCols,PnCols,ThreshCols] = Find_Perm_Data(conditions,...
        Pconnect_Perm,Pn_Perm,Thresh_Perm);
    Result = Average(ThreshCols,PnCols,PconCols);
    ResResult = reshape(...
                Result,[size(Average,1) size(Average,3)]);

    X = Thresh_Begin:Thresh_Intv:Thresh_End;
    Y = Pconnect_Mat(1,1,1):Pconnect_Mat(1,1,2):...
```

```
                                Pconnect_Mat(1,1,3);

    [XX,YY] = meshgrid(Y,X);
    %(cool, winter, summer, spring, map, autumn(5),etc)
    colormap cool(7);
    surf(XX,YY,ResResult);
    % zlim([0 1]);
    % ylim([0 10]);
    view([-127.5 30]);
    xlabel('P_c','fontweight','bold','FontSize',16);
    ylabel('\theta','fontweight','bold','FontSize',22);
    set(get(gca,'ZLabel'),'Rotation',0);
    zlabel('\Omega','fontweight','bold','FontSize',18);
    title({['P_n = (' num2str(Pn) '),   P_c = ( : ),   W = ('...
    num2str(Weight_Pop(1,1)) ') ']},'FontSize',16);
    pause(.03);
    A(i) = getframe(fig1);
    cla;
end

close(fig1);
movie2avi(A,'1_Society.avi','compression','Cinepak','fps',3)

%% 2 Societies

% Define Parameters

%'numframes' is the number of graphs (and therefore frames)
%to be made for the video.
numframes = 1+(Pn_Mat(1,1,3)-Pn_Mat(1,1,1))/Pn_Mat(1,1,2);
clear i;

Pn2 = 0.80;
Pc2 = 0.001;
Pc3 = 0.001;
Pc4 = 0.01;

% Frame Generation

%In this example Pn1 is the parameter changing every frame while
%Pc1 is the variable being plotted along with Threshold and
%Proportion.
```

```
fig1 = figure;
set(fig1,'render','painters')
for i = 1:numframes;
    Pn1 = Pn_Mat(1,1,1) + (i-1)*Pn_Mat(1,1,2);
    conditions = {    'Pconnect' [2 Pc2];
                      'Pconnect' [3 Pc3];
                      'Pconnect' [4 Pc4];
                      'Pn'       [1 Pn1];
                      'Pn'       [2 Pn2]};

    Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
    [PconCols,PnCols,ThreshCols] = Find_Perm_Data(conditions,...
        Pconnect_Perm,Pn_Perm,Thresh_Perm);
    Result = Average(ThreshCols,PnCols,PconCols);
    ResResult = reshape(...
                Result,[size(Average,1) size(Average,3)]);

    X = Thresh_Begin:Thresh_Intv:Thresh_End;
    Y = 0.001:0.001:0.02;

    [XX,YY] = meshgrid(Y,X);
    colormap cool;
    surf(XX,YY,ResResult);
    view([-127.5 30]);
    xlabel('Pconnect');
    ylabel('Threshold');
    zlabel('Proportion of Adoption');
    title(['Adoption for Pn = (' num2str(Pn1) ', ' num2str...
        (Pn2)'), Pconnect = ( : , ' num2str(Pc2) ', ' ...
        num2str(Pc3) ', 'num2str(Pc4) '), Weight = (' ...
        num2str(Weight_Pop(1,1)) ', 'num2str(Weight_Pop...
        (1,2)) ', ' num2str(Weight_Pop(2,1)) ', ' ...
        num2str(Weight_Pop(2,2)) ') ']);
    pause(.05);
    A(i) = getframe(fig1);
    cla;
end

% Movie making

close(fig1);
movie2avi(A,'testsss.avi','compression','Cinepak','fps',3)
```

```
%% 3-Societies

%Define Parameters (3-D)

%'numframes' is the number of graphs (and therefore frames)
%to be made for the video.
numframes = 1+(Pn_Mat(1,1,3)-Pn_Mat(1,1,1))/Pn_Mat(1,1,2);
clear i;

Pc2 = 0.005;
Pc3 = 0.001;
Pc4 = 0.005;
Pc5 = 0.01;
Pc6 = 0.001;
Pc7 = 0.001;
Pc8 = 0.001;
Pc9 = 0.01;

Pn1 = 0.75;
Pn3 = 0.90;

% Frame Generation (3-D)

% In this example Pn1 is the parameter changing every frame while
% Pc1 is the variable being plotted along with Threshold and
% Proportion.

fig1 = figure;
set(fig1,'render','painters')
for i = 1:numframes;
    Pn2 = Pn_Mat(1,1,1) + (i-1)*Pn_Mat(1,1,2);
    conditions = {    'Pconnect' [2 Pc2];
                      'Pconnect' [3 Pc3];
                      'Pconnect' [4 Pc4];
                      'Pconnect' [5 Pc5];
                      'Pconnect' [6 Pc6];
                      'Pconnect' [7 Pc7];
                      'Pconnect' [8 Pc8];
                      'Pconnect' [9 Pc9];
                      'Pn' [1 Pn1];
                      'Pn' [2 Pn2];
                      'Pn' [3 Pn3]};
```

```
    Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
    [PconCols,PnCols,ThreshCols] = Find_Perm_Data(conditions,...
                        Pconnect_Perm,Pn_Perm,Thresh_Perm);
    Result = Average(ThreshCols,PnCols,PconCols);
    ResResult = reshape(...
                Result,[size(Average,1) size(Average,3)]);

    X = Thresh_Begin:Thresh_Intv:Thresh_End;
    Y = 0.001:0.0005:0.02;

    [XX,YY] = meshgrid(Y,X);
    colormap cool;
    surf(XX,YY,ResResult);
    view([-127.5 30]);
    xlabel('Pconnect');
    ylabel('Threshold');
    zlabel('Proportion of Adoption');
    title(['Pn = (' num2str(Pn1) ', ' num2str(Pn2)...
    ',' num2str(Pn3) '), Pc = ( :, ' num2str(Pc2)...
    ', ' num2str(Pc3) '; ' num2str(Pc4) ', ' num2str(Pc5)...
    ', ' num2str(Pc6) '; ' num2str(Pc7) ', ' num2str(Pc8)...
    ', ' num2str(Pc9) ')']);
    pause(.03);
    A(i) = getframe(fig1);
    cla;
end

% Movie making (3-D)

close(fig1);
movie2avi(A,'run2Pn1_0.75Pn3_0.90.avi','compression','Cinepak','fps',3)
```

```
%*****************************************************
%* Find_Perm_Data                                    *
%* Data Ordering and formatting for plot generation. *
%* By Francisco Dominguez                            *
%*****************************************************

%In order to easily access any values of the entire data array
%this script was created. With the use of this code we can ac-
%cess data with ease even from an array of N dimensions in the
%case that we have N different parameters of M societies. This
%code is used in Graph3d and in Movievolution. It essentially
%obtains the required data from the data array and the output
%is synthetized into a simpler matrix. The code takes the va-
%lues input in 'conditions' as constant values and searches for
%that location in the data array. Whatever is not input in
%'conditions' the code considers variable and so that entire
%dimension in the data array is what becomes the output of the
%code.
%-------------------------------------------------------------

function [PconCols,PnCols,ThreshCols] = Find_Perm_Data(...
          conditions,Pconnect_Perm,Pn_Perm,Thresh_Perm)


CheckPcon = [];CheckPn = [];CheckThresh = [];

%Checks for the variable parameter.

for i = 1 : size(conditions,1)

    if strncmpi(conditions{i,1},'Pconnect',7)
        CheckPcon = strcat(CheckPcon,[' abs(Pconnect_Perm(' ...
            num2str(conditions{i,2}(1)) ',:)-' ...
            num2str(conditions{i,2}(2)) ')<1e-10 & ']);
    elseif strncmpi(conditions{i,1},'Pn',7)
        CheckPn = strcat(CheckPn,[' abs(Pn_Perm(' ...
            num2str(conditions{i,2}(1)) ',:)-' ...
            num2str(conditions{i,2}(2)) ')<1e-10 & ']);
    else
        CheckThresh = strcat(CheckThresh,[' abs(Thresh_Perm-' ...
            num2str(conditions{i,2}(1)) ')<1e-10 & ']);
    end
end
```

```
%Outputs Variable parameter information according to
%fixed parameters input.

if isempty(CheckPcon)
    PconCols = 1:size(Pconnect_Perm,2);
else
    PconCols = eval(CheckPcon(1:end-2));
end
if isempty(CheckPn)
    PnCols = 1:size(Pn_Perm,2);
else
    PnCols = eval(CheckPn(1:end-2));
end
if isempty(CheckThresh)
    ThreshCols = 1:size(Thresh_Perm,2);
else
    ThreshCols = eval(CheckThresh(1:end-2));
end
```

```
%*********************************************************
%*Graph2D                                                 *
%*Template for generating 2D plot.                        *
%* By Francisco Dominguez                                 *
%*********************************************************

%This is basically a template for plotting 2D plots of data
%from the simulation. It must be edited based on individual
%plot requirements.
%---------------------------------------------------------------

%%One Society_2-D Plot

Pc = 0.012;
Pn = 0.7;
conditions = {    'Pconnect' [1 Pc];
                  'Pn' [1 Pn]};
Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
[PconCols,PnCols,ThreshCols] = Find_Perm_Data(conditions,...
    Pconnect_Perm,Pn_Perm,Thresh_Perm);
Result = Average(ThreshCols,PnCols,PconCols);
Error = MSE(ThreshCols,PnCols,PconCols);

figure (1);
h = errorbar(Thresh_Begin:Thresh_Intv:Thresh_End,Result,Error);
% h = plot(Thresh_Begin:Thresh_Intv:Thresh_End,Result);
set(h,'color',[0, 0.5078, 0])                  %// data
set(h,'LineWidth',1.4,{'LineStyle'},{'-'});    %'--'; ':'; '-.'
set(h,{'Marker'},{'o'});                       %'none';'o';'x'
set(gca, 'FontSize',14)
grid on;
xlim([44 58]);
ylim([-0.1 1.1]);
xlabel('\theta','fontweight','bold','FontSize',18);
ylabel('\Omega      ','fontweight','bold','FontSize',16);
set(get(gca,'YLabel'),'Rotation',0);
title({['P_n = (' num2str(Pn) '),   P_c = (' num2str(Pc) '),...
    W = (' num2str(Weight_Pop(1,1)) ') ']},'FontSize',16);
print -djpeg  Pn0.7Pc0.012error.jpg -r250;
saveas(gcf,'Pn0.7Pc0.012error.fig');

%% Multiple 2-D Plots
```

```
Pn = 0.7;
z = 1;
Result = zeros(1+(Thresh_End - Thresh_Begin)/Thresh_Intv,...
         (Pconnect_Mat(1,1,3)-Pconnect_Mat(1,1,1))/...
         Pconnect_Mat(1,1,2));
for i= Pconnect_Mat(1,1,1):Pconnect_Mat(1,1,2):...
                           Pconnect_Mat(1,1,3);

conditions = {    'Pconnect' [1 i];
                  'Pn' [1 Pn]};
Thresh_Perm = Thresh_Begin:Thresh_Intv:Thresh_End;
[PconCols,PnCols,ThreshCols] = Find_Perm_Data(conditions,...
    Pconnect_Perm,Pn_Perm,Thresh_Perm);
Result(:,z) = Average(ThreshCols,PnCols,PconCols);
Error(:,z) = MSE(ThreshCols,PnCols,PconCols);
z = z + 1;
end

k1 = 0.01;
k2 = 0.012;
k3 = 0.014;

x1 = 1 + (k1 - Pconnect_Mat(1,1,1))/Pconnect_Mat(1,1,2);
x2 = 1 + (k2 - Pconnect_Mat(1,1,1))/Pconnect_Mat(1,1,2);
x3 = 1 + (k3 - Pconnect_Mat(1,1,1))/Pconnect_Mat(1,1,2);

figure (1);
h = plot(Thresh_Begin:Thresh_Intv:Thresh_End,Result(:,x1),...
         Thresh_Begin:Thresh_Intv:Thresh_End,Result(:,x2),...
         Thresh_Begin:Thresh_Intv:Thresh_End,Result(:,x3));
set(h,'LineWidth',1.4,{'LineStyle'},{'-'});  % '--' ; ':' ; '-.'
set(h,{'Marker'},{'o'});                     % 'none';'o';'x'
set(gca, 'FontSize',14)
axis tight;
grid on;
xlim([36 66]);
ylim([-0.1 1.1]);
legend('P_c = 0.01','P_c = 0.012','P_c = 0.014');
xlabel('\theta','fontweight','bold','FontSize',18);
ylabel('\Omega      ','fontweight','bold','FontSize',16);
set(get(gca,'YLabel'),'Rotation',0);
title({['P_n = (' num2str(Pn) '),    W = ('...
    num2str(Weight_Pop(1,1)) ') ']},'FontSize',16);
```

```
print -djpeg  Pn0.7triplePc.jpg -r250;
saveas(gcf,'Pn0.7triplePc.fig');

%% Critical Threshold


Pc = Pconnect_Begin;
Pn = Pn_Begin;
Weight_Pop = 1;

n =2;
x = Critical(2, 1 : n :end);
y = Critical(1, 1 : n :end);

h = plot( x ,y,'MarkerSize', 4);
set(h,'LineWidth',1.4,{'LineStyle'},{'-'});    % '--' ; ':' ; '-.'
set(h,{'Marker'},{'o'});                       % 'none';'o';'x'
set(h,{'Color'},{'b'});                        % 'r';'g';'b'
set(gca, 'FontSize',14)
pbaspect([1.8 1 1]);

axis tight;
grid on;
xlabel('m','fontweight','bold','FontSize',18);
ylabel('\theta_c      ','fontweight','bold','FontSize',16);
set(get(gca,'YLabel'),'Rotation',0);
title({['P_n = (' num2str(Pn) '),   P_c = (' num2str(Pc) '),...
   W = (' num2str(Weight_Pop) ') ']},'FontSize',16);

%Obtain a Fit.

coeffs = polyfit(x, y, 1);

% Get fitted values

b = 1 + (Pop_End - Pop_Begin) / (Pop_Intv*n);
fittedX = linspace(min(x), max(x), b);
fittedY = polyval(coeffs, fittedX);
R = corrcoef([y' fittedY'])

% Plot the fitted line

hold on;
```

```
plot(fittedX, fittedY, 'r-', 'LineWidth', 2);

print -djpeg  Critical.jpg -r250;
saveas(gcf,'Critical.fig');
```

```
%*********************************************************
%* Reshape_Variable                                      *
%* Data Reshaping for 2d plots.                          *
%* By Francisco Dominguez                                *
%*********************************************************

%This script was made to quickly access data for 2D plotting.
%Later Find_Perm_Data was made to be more general but this
%script is included for completeness and for the fact that
%for 2D plots it is still more efficient.
%If the data array is too big, getting data in a form that
%Matlab understands for plotting 2D graphs can be cumbersome.
%This script was made to aid in that task.
%The code will take and entry data matrix (Average or MSE for
%example) then a number or vector determining the constant
%dimension or dimensions, and finally a number or vector
%determining the constant entry in those dimensions.
%
%As an example:
%
% AvgThresh = Reshape_Variable(Average,2,1);
% MSEThresh = Reshape_Variable(MSE,2,1);
% Graph_Thresh(Thresh_Begin,Thresh_Intv,Thresh_End,...
%              AvgThresh,MSEThresh);
%
%In here, the code will take the second dimension of the Average
%and MSE arrays and of that dimension the first entry and it
%will report the result in a vector form. The code will not work
%properly if the desired output is of a higher dimension than 1,
%i.e. it only works well to output vectors, therefore one must
%be careful to determine the correct number of constants.
%
%Another example:
%
% AvgThresh = Reshape_Variable(Average,[2 1],[1 3]);
% MSEThresh = Reshape_Variable(MSE[2 1],[1 3]);
% Graph_Thresh(Thresh_Begin,Thresh_Intv,Thresh_End,...
%              AvgThresh,MSEThresh);
%---------------------------------------------------------------


function [z] = Reshape_Variable(x,Const_Dim,entry_num)
```

```
Dimen = size(x);

[Const_Dim,IX] = unique(Const_Dim);
NumConsts = length(Const_Dim);
entry_num = entry_num(IX);

if NumConsts == 1
    if Const_Dim == 1
        y = x(entry_num,:,:);
        rows = Dimen(2);
    elseif Const_Dim == 2
        y = x(:,entry_num,:);
        rows = Dimen(1);
    else
        y = x(:,:,entry_num);
        rows = Dimen(1);
    end
    z = reshape(y,rows,[]);
else
    [~,IX] = sort(Const_Dim,'descend');
    for i = 1:NumConsts
        if Const_Dim(IX(i)) == 1
            x = x(entry_num(IX(i)),:,:);
        elseif Const_Dim(IX(i)) == 2
            x = x(:,entry_num(IX(i)),:);
        else
            x = x(:,:,entry_num(IX(i)));
        end
    end
    z=x(:);
end
```

```
%*********************************************************
%* Critical_Test                                         *
%* Templates to perform critical Threshold tests.        *
%* By Francisco Dominguez                                *
%*********************************************************

%This is only a template for a very specific kind of task. We
%want to find the first value of Threshold that gives us a
%Proportion of adoption greater than X%. This can be then
%plotted as a function of population size, Pconnect or Pn.
%Both of the following templates were used as a basis when this
%kind of graph was being studied.
%-----------------------------------------------------------

%% Template 1

loop_rows = find(Pconnect_Perm(1,:)==.01007);
g = zeros(1,length(loop_rows));
for i = 1:length(loop_rows)
    f = find(Average(:,loop_rows(i))>2e-5,1,'last');
    if ~isempty(f)
        g(i) = Thresh_Begin + Thresh_Intv*(f-1);
    end
end
plot(Pconnect_Perm(4,loop_rows),g)

%% Template 2

g = zeros(1,size(Average,3));
for i = 1 :size(Average,3)
    f = find(Average(:,i)>4e-3,1,'last');
    if ~isempty(f)
        g(i) = f;
    end
end
```

B.1 Graphical User Interface

```
%*********************************************************
%* GraphUserInt                                          *
%* Graphical User Interface for dynamic plotting of data. *
%* By Francisco Dominguez                                *
%*********************************************************

%This is the Graphical User Interface for the dynamic plots of
%data. The GUI was designed as an upgraded tool of videomaking
%for the purpose of data analysis. In the GUI one can load the
%data of a simulation for 2 populations, then pick from Thres-
%hold, Pconnect and Pn the values that will vary (picking 1 will
%net us a 2D graph and picking 2 a 3D graph), set the specific
%value of everything that is constant and then plot the data.
%
%One must fill the values of everything that is to be constant.
%This means the appropriate Pn and Pconnect for both populations
%and for the interaction between them. To do this, when one
%clicks on the space provided to input the values, a slider with
%all the possibilities within that data set will appear; one can
%then choose the desired value. To obtain the graph one simply
%clicks on the "Generate Graph" button.
%
%Another important feature activated by checking the box
%'Activate Slider' gives us the possibility to change one of the
%fixed values and obtain a new graph with those values. The graph
%is actually updated with the simple move of the slider represen-
%ting the vales of a given parameter. The 'Activate Slider'
%checkbox must be checked before generating a graph.
%
%-------------------------------------------------------------

%NOTE: eventdata  reserved - to be defined in a future version...
%of MATLAB

function varargout = GraphUserInt(varargin)
% GraphUserInt(Pconnect_Perm,Pn_Perm,Thresh_Begin:...
%              Thresh_End,Average,MSE,NumPop)
% GraphUserInt MATLAB code for GraphUserInt.fig
% GraphUserInt, by itself, creates a new GraphUserInt or raises
```

```
% the existing singleton*.
%
% H = GraphUserInt returns the handle to a new GraphUserInt or
% the handle to the existing singleton*.
%
% GraphUserInt('CALLBACK',hObject,eventData,handles,...) calls
% the local function named CALLBACK in GraphUserInt.M with the
% given input arguments.
%
% GraphUserInt('Property','Value',...) creates a new GraphUserInt
% or raises the existing singleton*.  Starting from the left,
% property value pairs are applied to the GUI before
% GraphUserInt_OpeningFcn gets called. An unrecognized property
% name or invalid value makes property application stop.  All
% inputs are passed to GraphUserInt_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
% only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GraphUserInt

% Last Modified by GUIDE v2.5 16-May-2012 16:24:00

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @GraphUserInt_OpeningFcn, ...
    'gui_OutputFcn',  @GraphUserInt_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before GraphUserInt is made visible.
function GraphUserInt_OpeningFcn(hObject, eventdata, handles,...
 varargin   %#ok<*INUSL>
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% handles    structure with handles and user data (see GUIDATA)
% Choose default command line output for GraphUserInt
handles.output = hObject;
if isempty(varargin)
    handles.PopNums     = 2;
%     handles.Pcon_Perm   = {0:.4:3;1:2:10;1:1;1.2:.13:5};
%     handles.Pn_Perm     = {1:1;1.2:.13:5};
%     handles.Thresh_Perm = 17:2:25;
else
    handles.Pcon_Perm   = varargin{1};
    handles.Pn_Perm     = varargin{2};
    handles.Thresh_Perm = varargin{3};
    handles.Average     = varargin{4};
    handles.MSE         = varargin{5};
    handles.PopNums     = size(handles.Pcon_Perm,1);
end
handles.Cond{1} = [0 0];
handles.Cond{2} = 0;
handles.Cond{3} = 0;
handles.Cond{4} = [0 0];
handles.Thresh  = 0;
handles.activebutton = 0;

% Update handles structure
guidata(hObject, handles);
setappdata(handles.figure1,'fhUpdateGraph',...
           @Generate_Graph_Callback)
setappdata(0,'hMainGui',handles.figure1);
setappdata(handles.figure1,'handles',handles);
% UIWAIT makes GraphUserInt wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
%     command line.
function varargout = GraphUserInt_OutputFcn(hObject,...
```

```
                    eventdata, handles)
% varargout  cell array for returning output args
% (see VARARGOUT);
% hObject    handle to figure
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)...
    %#ok<*DEFNU,*INUSD>
% hObject    handle to popupmenu1 (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String'))
% returns popupmenu1 contents as cell array
% contents{get(hObject,'Value')} returns selected item
% from popupmenu1

% get(hObject,

% --- Executes during object creation, after setting
%     all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: popupmenu controls usually have a white background
% on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Thresh_Box.
function Thresh_Box_Callback(hObject, eventdata, handles)
% hObject    handle to Thresh_Box (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
CheckBox_Change(hObject,handles)
```

```
% Hint: get(hObject,'Value') returns toggle state of Thresh_Box


% --- Executes on button press in Pcon_Box.
function Pcon_Box_Callback(hObject, eventdata, handles)
% hObject    handle to Pcon_Box (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
CheckBox_Change(hObject,handles)
% Hint: get(hObject,'Value') returns toggle state of Pcon_Box


% --- Executes on button press in Pseed_Box.
function Pseed_Box_Callback(hObject, eventdata, handles)
% hObject    handle to Pseed_Box (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
CheckBox_Change(hObject,handles)
% Hint: get(hObject,'Value') returns toggle state of Pseed_Box

function CheckBox_Change(hObject,handles)

Pcon_Check   = get(handles.Pcon_Box,'value');
Pseed_Check  = get(handles.Pseed_Box,'value');
Thresh_Check = get(handles.Thresh_Box,'value');
NumChecks    = Pcon_Check + Thresh_Check + Pseed_Check;
if NumChecks>2
    set(hObject,'Value',0)
    warndlg('You can only have a max of 2 boxes checked...
             for graphing.')
end


% --- Executes on selection change in Condition_Pop.
function Condition_Pop_Callback(hObject, eventdata, handles)
% hObject    handle to Condition_Pop (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String'))
% returns Condition_Pop
% contents as cell array
% contents{get(hObject,'Value')} returns selected item
% from Condition_Pop
PopNum = get(hObject,'value');
if handles.PopNums<2
```

```
    set(hObject,'value',1);
    warndlg('There is only one population for this data set!')
else
    if sqrt(PopNum)==round(sqrt(PopNum))
        set(handles.Pseed_Edit,'string',num2str(...
            handles.Cond{PopNum}(2)),'enable','inactive')
    else
        set(handles.Pseed_Edit,'string',num2str(0),...
        'enable','off')
    end
    set(handles.Pcon_Edit,'string',num2str(handles....
    Cond{PopNum}(1)))
end


% --- Executes during object creation, after setting all
% properties.
function Condition_Pop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Condition_Pop (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: popupmenu controls usually have a white background
% on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Pcon_Edit_Callback(hObject, eventdata, handles)
% hObject    handle to Pcon_Edit (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Pcon_Edit as
% text
% str2double(get(hObject,'String')) returns contents of Pcon_Edit
% as a double
PopNum = get(handles.Condition_Pop,'value');
handles.Cond{PopNum}(1) = str2double(get(hObject,'string'));
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all
% properties.
function Pcon_Edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pcon_Edit (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Pseed_Edit_Callback(hObject, eventdata, handles)
% hObject    handle to Pseed_Edit (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Pseed_Edit
% as text
%        str2double(get(hObject,'String')) returns contents of
% Pseed_Edit
% as a double
PopNum = get(handles.Condition_Pop,'value');
handles.Cond{PopNum}(2) = str2double(get(hObject,'string'));
guidata(hObject, handles);


% --- Executes during object creation, after setting all
% properties.
function Pseed_Edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pseed_Edit (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function Thresh_Edit_Start_Callback(hObject, eventdata, handles)
% hObject    handle to Thresh_Edit_Start (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of
% Thresh_Edit_Start as
% text str2double(get(hObject,'String')) returns contents of
% Thresh_Edit_Start as a double
handles.Thresh(1) = str2double(get(hObject,'String'));
guidata(hObject, handles);

% --- Executes during object creation, after setting all
% properties.
function Thresh_Edit_Start_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Thresh_Edit_Start (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

function Thresh_Edit_End_Callback(hObject, eventdata, handles)
% hObject    handle to Thresh_Edit_End (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of
% Thresh_Edit_End as text
% str2double(get(hObject,'String')) returns contents of
% Thresh_Edit_End as a double
handles.Thresh(1) = str2double(get(hObject,'String'));
guidata(hObject, handles);

% --- Executes during object creation, after setting all
% properties.
function Thresh_Edit_End_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Thresh_Edit_End (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Generate_Graph.
function Generate_Graph_Callback(hObject, eventdata, handles)
% hObject    handle to Generate_Graph (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
hMainGui = getappdata(0,'hMainGui');
handles  = getappdata(hMainGui,'handles');

Pcon_Check   = get(handles.Pcon_Box,'value');
Pseed_Check  = get(handles.Pseed_Box,'value');
Thresh_Check = get(handles.Thresh_Box,'value');
NumChecks = Pcon_Check + Pseed_Check + Thresh_Check;
NumPop = length(handles.Cond);
PconCond = 1:NumPop;
PnCond = (1:sqrt(NumPop)).^2;

if NumChecks
    conditions = cell(0);
    for i = 1:NumPop
        if sqrt(i)==round(sqrt(i))
            if handles.Cond{i}(2)~=0
                conditions(end+1,:) = {'Pn' [sqrt(i)...
                 handles.Cond{i}(2)]}; %#ok<*AGROW>
                PnCond(PnCond==i) = [];
            end
        end
        if handles.Cond{i}(1)~=0
            conditions(end+1,:) = {'Pconnect'...
             [i handles.Cond{i}(1)]};
            PconCond(PconCond==i) = [];
        end
    end
    if handles.Thresh(1)~=0
        conditions(end+1,:) = {'Thresh' handles.Thresh};
    end
%     conditions(end+1,:) = {'Pn' [2 .8]};
%     conditions(end+1,:) = {'Pn' [3 .8]};
```

```
    [PconCols,PnCols,ThreshCols] = Find_Perm_Data...
        (conditions,handles.Pcon_Perm,handles.Pn_Perm,...
         handles.Thresh_Perm);
    Average      = handles.Average(ThreshCols,PnCols,PconCols);
    MSE          = handles.MSE(ThreshCols,PnCols,PconCols);

    if NumChecks>1
        if Thresh_Check && Pcon_Check
            ResResult = reshape(Average,[size(Average,1) ...
                        size(Average,3)]);

            X = handles.Thresh_Perm;
            Y = unique(handles.Pcon_Perm(PconCond,PconCols));
            [XX,YY] = meshgrid(Y,X);
            colormap cool;
            surf(handles.axes1,XX,YY,ResResult);
            view(handles.axes1,[-127.5 30]);
            xlabel(handles.axes1,'Pconnect');
            ylabel(handles.axes1,'Threshold');
            zlabel(handles.axes1,'Proportion of Adoption');
% title(['Adoption for Pn = (' num2str(handles.Cond{) ',...
% ' num2str(Pn2)'), Pconnect = ( : , ' num2str(Pc2) ',...
% ' num2str(Pc3) ', 'num2str(Pc4) '), Weight = (5,0,5,1)']);
        elseif Thresh_Check && Pseed_Check
            ResResult = reshape(Average,[size(Average,1)...
             size(Average,2)]);

            X = handles.Thresh_Perm;
            Y = unique(handles.Pn_Perm(PnCond,PnCols));
            [XX,YY] = meshgrid(Y,X);
            colormap cool;
            surf(handles.axes1,XX,YY,ResResult);
            view(handles.axes1,[-127.5 30]);
            xlabel(handles.axes1,'Pn');
            ylabel(handles.axes1,'Threshold');
            zlabel(handles.axes1,'Proportion of Adoption');
        elseif Pcon_Check && Pseed_Check
            ResResult = reshape(Average,[size(Average,2)...
             size(Average,3)]);

            X = unique(handles.Pn_Perm(PnCond,PnCols));
            Y = unique(handles.Pcon_Perm(PconCond,PconCols));
            [XX,YY] = meshgrid(Y,X);
```

```
            colormap cool;
            surf(handles.axes1,XX,YY,ResResult);
            view(handles.axes1,[-127.5 30]);
            xlabel(handles.axes1,'Pconnect');
            ylabel(handles.axes1,'Pn');
            zlabel(handles.axes1,'Proportion of Adoption');
        end
    else
        if Thresh_Check
            errorbar(handles.axes1,...
               handles.Thresh_Perm(ThreshCols),Average,MSE);
            title(handles.axes1,'Proportion vs. Thresh')
            xlabel(handles.axes1,'Threshold')
            ylabel(handles.axes1,...
            'Proportion of Adoption P=1-(NAf/IN)');
        elseif Pcon_Check
            errorbar(handles.axes1,...
            handles.Pcon_Perm(PconCols),Average,MSE);
            title(handles.axes1,'Proportion vs. Pconnect')
            xlabel(handles.axes1,'Pconnect')
            ylabel(handles.axes1,...
            'Proportion of Adoption P=1-(NAf/IN)');
        else
            errorbar(handles.axes1,...
               handles.Pn_Perm(1,PnCols),Average,MSE);
            title(handles.axes1,'Proportion vs. Pseed')
            xlabel(handles.axes1,'Pseed')
            ylabel(handles.axes1,...
            'Proportion of Adoption P=1-(NAf/IN)');
        end
    end
    axis(handles.axes1,'tight')
    elseif ~NumChecks || NumChecks>2
    warndlg('Must have at least one graph type checked!')
end


% --- If Enable == 'on', executes on mouse press in
% --- 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixelr
% --- border or over Pcon_Edit.
function Pcon_Edit_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to Pcon_Edit (see GCBO)
```

```
% handles    structure with handles and user data (see GUIDATA)
PopNum  = get(handles.Condition_Pop,'value');
PconVal = str2double(get(hObject,'string'));
PconStr = get(handles.Condition_Pop,'string');

pUniq   = unique(handles.Pcon_Perm(PopNum,:));
pmin    = min(pUniq);
pmax    = max(pUniq);
if (pmax~=pmin) %#ok<*BDLOG>
    pint    = abs(pUniq(2)-pUniq(1))/(pmax-pmin);
    pintL   = .25;
    if pintL < pint
        pintL = pint;
    end
else
    pint  = 1;
    pintL = pint;
end
if PconVal<pmin || PconVal>pmax
    PconVal = pmin;
end

newPconVal = ex_guide_timergui(pmin,pmax,PconVal,[pint pintL],...
    [PconStr{PopNum} ' Pconnect'],handles.activebutton);

if ~isempty(newPconVal)
    handles.Cond{PopNum}(1) = newPconVal;
    set(hObject,'string',num2str(newPconVal))
    hMainGui = getappdata(0,'hMainGui');
    setappdata(hMainGui,'handles',handles);
    guidata(hObject, handles);
end


% --- If Enable == 'on', executes on mouse press in
% --- 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixelr
% --- border or over Pseed_Edit.
function Pseed_Edit_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to Pseed_Edit (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
if strncmp(get(hObject,'enable'),'inactive',5)
    PopNum  = get(handles.Condition_Pop,'value');
```

```
    PnVal = str2double(get(hObject,'string'));
    PnStr = get(handles.Condition_Pop,'string');

    pUniq   = unique(handles.Pn_Perm(sqrt(PopNum),:));
    pmin    = min(pUniq);
    pmax    = max(pUniq);
    if (pmax-pmin)
        pint    = abs(pUniq(2)-pUniq(1))/(pmax-pmin);
        pintL   = .25;
        if pintL < pint
            pintL = pint;
        end
    else
        pint  = 1;
        pintL = pint;
    end
    if PnVal<pmin || PnVal>pmax
        PnVal = pmin;
    end

    newPnVal = ex_guide_timergui(pmin,pmax,PnVal,[pint pintL],...
        [PnStr{PopNum} ' Pn'],handles.activebutton);

    if ~isempty(newPnVal)
        handles.Cond{PopNum}(2) = newPnVal;
        set(hObject,'string',num2str(newPnVal))
        hMainGui = getappdata(0,'hMainGui');
        setappdata(hMainGui,'handles',handles);
        guidata(hObject, handles);
    end
end


% --- If Enable == 'on', executes on mouse press in
% --- 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixelr
% --- border or over Thresh_Edit_Start.
function Thresh_Edit_Start_ButtonDownFcn(hObject,...
         eventdata, handles)
% hObject    handle to Thresh_Edit_Start (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
if strncmp(get(hObject,'enable'),'inactive',5)
    ThVal = str2double(get(hObject,'string'));
```

```
    ThStr = 'Threshold Value';

    pUniq   = unique(handles.Thresh_Perm);
    pmin    = min(pUniq);
    pmax    = max(pUniq);
    if (pmax-pmin)
        pint    = abs(pUniq(2)-pUniq(1))/(pmax-pmin);
        pintL   = .25;
        if pintL < pint
            pintL = pint;
        end
    else
        pint  = 1;
        pintL = pint;
    end
    if ThVal<pmin || ThVal>pmax
        ThVal = pmin;
    end

    newThVal = ex_guide_timergui(pmin,pmax,ThVal,[pint pintL],...
        ThStr,handles.activebutton);

    if ~isempty(newThVal)
        handles.Thresh(1) = newThVal;
        set(hObject,'string',num2str(newThVal))
        hMainGui = getappdata(0,'hMainGui');
        setappdata(hMainGui,'handles',handles);
        guidata(hObject, handles);
    end
end


% --- Executes on button press in Active_Slider.
function Active_Slider_Callback(hObject, eventdata, handles)
% hObject    handle to Active_Slider (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of...
% Active_Slider
handles.activebutton = get(hObject,'Value');
guidata(hObject, handles);
```

```
%*********************************************************
%* ex_guide_timergui                                     *
%* Matlab generated code for the GUI.                    *
%* By Francisco Dominguez                                *
%*********************************************************

%This is part of the GUI. It is essentially a Matlab generated
%code for the designed GUI to obtain the graphs.
%-----------------------------------------------------------

% NOTE: eventdata  reserved - to be defined in a future
%       version of MATLAB

function varargout = ex_guide_timergui(varargin)
% EX_GUIDE_TIMERGUI - Execute graphic updates at
% regular intervals
%   MATLAB code for ex_guide_timergui.fig
%      EX_GUIDE_TIMERGUI, by itself, creates a new
%      EX_GUIDE_TIMERGUI or raises the existing singleton*.
%
%      H = EX_GUIDE_TIMERGUI returns the handle to a new
%      EX_GUIDE_TIMERGUI or the handle to the existing
%      singleton*.
%
%      EX_GUIDE_TIMERGUI('CALLBACK',hObject,eventData,...
%      handles,...)
%      calls the local function named CALLBACK in
%      EX_GUIDE_TIMERGUI.M with the given input arguments.
%
%      EX_GUIDE_TIMERGUI('Property','Value',...) creates a new
%      EX_GUIDE_TIMERGUI or raises the existing singleton*.
%      Starting from the left, property value pairs are applied
%      to the GUI before ex_guide_timergui_OpeningFcn gets
%      called. An unrecognized property name or invalid value
%      makes property application stop.  All inputs are passed
%      to ex_guide_timergui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
%       allows only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES, TIMER

% Last Modified by GUIDE v2.5 16-May-2012 13:24:20
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
           'gui_Singleton',  gui_Singleton, ...
           'gui_OpeningFcn', @ex_guide_timergui_OpeningFcn, ...
           'gui_OutputFcn',  @ex_guide_timergui_OutputFcn, ...
           'gui_LayoutFcn',  [] , ...
           'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ex_guide_timergui is made visible.
function ex_guide_timergui_OpeningFcn(hObject, eventdata,...
    handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ex_guide_timergui
% (see VARARGIN)

% Choose default command line output for ex_guide_timergui
handles.output = [];

% START USER CODE
% Create a timer object to fire at 1/10 sec intervals
% Specify function handles for its start and run callbacks
% handles.timer = timer(...
%  'ExecutionMode', 'fixedRate', ...       % Run timer repeatedly
%  Initial period is 1 sec.
%  'Period', 1, ...
%  Specify callback function
%  'TimerFcn', {@update_display,hObject});
% Initialize slider and its readout text field
```

```
if isempty(varargin)
    handles.pmin  = 0;
    handles.pmax  = 100;
    pVal  = 50;
    handles.pStep = [.01 .1];
    handles.Activ = 0;
else
    handles.pmin  = varargin{1};
    handles.pmax  = varargin{2};
    pVal  = varargin{3};
    handles.pStep = varargin{4};
    handles.GUItitle = varargin{5};
    set(handles.guilabel,'String',handles.GUItitle)
    handles.Activ = varargin{6};
end
set(handles.periodsldr,'Min',handles.pmin,'Max',handles.pmax,...
                'Value',pVal,'SliderStep',handles.pStep)
% set(handles.periodsldr,'Value',get(handles.timer,'Period'))
set(handles.slidervalue,'String',...
    num2str(get(handles.periodsldr,'Value')))
set(handles.text4,'String',num2str(handles.pmin))
set(handles.text5,'String',num2str(handles.pmax))
% END USER CODE

guidata(hObject,handles);
% Update handles structure
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command
% --- line.
function varargout = ex_guide_timergui_OutputFcn(hObject,...
     eventdata,handles)
% varargout  cell array for returning output args
% (see VARARGOUT);
% hObject    handle to figure
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
delete(hObject);
```

```
% --- Executes on slider movement.
function periodsldr_Callback(hObject, eventdata, handles)
% hObject    handle to periodsldr (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine
% range of slider

% START USER CODE
% Read the slider value
period = get(handles.periodsldr,'Value');
% Timers need the precision of periods to be greater than about
% 1 millisecond, so truncate the value returned by the slider
% period = period - mod(period,.01);
pmax = handles.pmax;
pmin = handles.pmin;
pStep = handles.pStep;
if abs(round(period/(pmax-pmin)/pStep(1))-(period/(pmax-pmin)////
    pStep(1)))>eps
    period = round(period/(pmax-pmin)/...
             pStep(1))*(pmax-pmin)*pStep(1);
end
if period>pmax
    period = pmax;
elseif period<pmin
    period = pmin;
end
% Set slider readout to show its value
set(handles.slidervalue,'String',num2str(period))
set(handles.periodsldr,'Value',period)
if handles.Activ
    hMainGui = getappdata(0,'hMainGui');
%     setappdata(hMainGui,'Slider',period)
    fhUpdateGraph = getappdata(hMainGui,'fhUpdateGraph');
    handles2 = getappdata(hMainGui,'handles');
    if strncmp(handles.GUItitle(end),'e',1)
        handles2.Thresh(1) = period;
        set(handles2.Thresh_Edit_Start,'string',num2str(period))
    else
        if strncmp(handles.GUItitle(end),'t',1)
            set(handles2.Pcon_Edit,'string',num2str(period))
            handles2.Cond{get(handles2.Condition_Pop,...
```

```
                    'value')}(1) = period;
        else
            set(handles2.Pseed_Edit,'string',num2str(period))
            handles2.Cond{get(handles2.Condition_Pop,...
                    'value')}(2) = period;
        end
    end
    setappdata(hMainGui,'handles',handles2);
    feval(fhUpdateGraph,[],[],[])
end
% If timer is on, stop it, reset the period, and start it again.
% if strcmp(get(handles.timer, 'Running'), 'on')
%     stop(handles.timer);
%     set(handles.timer,'Period',period)
%     start(handles.timer)
% else          % If timer is stopped, reset its period only.
%     set(handles.timer,'Period',period)
% end
% END USER CODE


% --- Executes during object creation, after setting
% --- all properties.
function periodsldr_CreateFcn(hObject, eventdata,handles)
% hObject    handle to periodsldr (see GCBO)
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% handles    structure with handles and user data (see GUIDATA)

% START USER CODE
% Necessary to provide this function to prevent timer callback
% from causing an error after GUI code stops executing.
% Before exiting, if the timer is running, stop it.
```

```
% if strcmp(get(handles.timer, 'Running'), 'on')
%     stop(handles.timer);
% end
% % Destroy timer
% delete(handles.timer)
% END USER CODE

% Hint: delete(hObject) closes the figure
uiresume(handles.figure1);


% --- Executes on button press in OK_Button.
function OK_Button_Callback(hObject, eventdata, handles)
% hObject    handle to OK_Button (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
period = get(handles.periodsldr,'Value');
% Timers need the precision of periods to be greater than about
% 1 millisecond, so truncate the value returned by the slider
% period = period - mod(period,.01);
handles.output = period;
% Update handles structure
guidata(hObject,handles);
uiresume(handles.figure1);

% --- Executes on button press in Cancel_Button.
function Cancel_Button_Callback(hObject, eventdata, handles)
% hObject    handle to Cancel_Button (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
uiresume(handles.figure1);


% --- Executes on button press in Clear.
function Clear_Callback(hObject, eventdata, handles)
% hObject    handle to Clear (see GCBO)
% handles    structure with handles and user data (see GUIDATA)
handles.output = 0;
% Update handles structure
guidata(hObject,handles);
uiresume(handles.figure1);
```