DRAFT COPY

Printed April 6, 2018

DYNAMICS OF AN SIS-LIKE AUDIENCE APPLAUSE MODEL

Antonio Miguel V. Cruz

acruz@nip.upd.edu.ph

An Undergraduate Thesis Submitted to

National Institute of Physics

College of Science

University of the Philippines

Diliman, Quezon City

In Partial Fulfillment of the Requirements

for the Degree of

BACHELOR OF SCIENCE IN PHYSICS

April 2018

Table of Contents

List of Figures

Abstract

asdf asdf

Chapter 1

Applause Dynamics

1.1 Nature of Applause

The audience applause is one of the most ubiquitous and timeless social phenomenons observed in human culture. People would normally applaud to express their approval over a social event, and may even jeer, shout, snap, and boo, on top of clapping. Such behavior is also exhibited by animals such as gorillas and chimpanzees. People collectively seem to know when to start clapping in any occasion, whether it is during or after a speech, performance, sport event, etc. People also unconsciously know whether to continue clapping or to stop. Audience members extend their applause to musicians playing an outro or to being addressed, and stop immediately when the musician proceeds with the next song. Lastly, it is historically known that people who have been hired to willingly and consciously applaud are inserted in an audience in hopes to extend the applause for a social event. These observations of an audience applause being self-organized and somehow connected give rise to the question of what the nature of applause is and how people decide when and how long to clap.

1.2 Social Dynamics

Social economics is an interdisciplinary study of the interrelationship between group and individual behavior. Individual decisions made interactively with others have been modelled formally to be able to understand the dynamics. The underlying assumption is that individuals are influenced by the choices of others. From this assumption, it can be said that feedback loops exist since the past decisions of a certain individuals may influence future decisions of others. Examples of such social phenomena are crime, teenage pregnancy, and high school dropout rates.[1][2]

1.3 Background

1.3.1 Networks

A network is a collection of points,called vertices or nodes, connected by lines, called edges. Networks are normally used to represent systems that contain individual parts that are somehow connected, such as the internet, big data, or social interactions. These are used to study the nature of individual components and their dynamics. A few properties of networks are topology and homogeneity. Network topology refers to how the network is connected. The different topologies are named after the physical shape of the network, such as a ring or a line. A key topology is a fully-connected network wherein all nodes are completely connected to each other. Network homogeneity refers to the nodes of the network. A homogeneous network has individual nodes of the same characteristics and properties while a heterogeneous network contains different nodes.

1.3.2 Complex Systems

By investigating the mechanisms that determine the topology of the networks of aforementioned systems (internet, big data, etc.) properties previously not observed when studying the components individually emerge. This has lead to a new field of

study that focuses on how relationships between components give rise to its collective behaviors and how the system interacts and forms relationships with its environment, called complex systems. Intrinsic to complex systems is that they are hard to model due to the complexity of the interacting components. Over-simplifying the complexity may lead to the failure of the model. Included in as a property of complexity is the inclination of a large system to mutate after reaching a critical point or state. Methods on studying complex systems have been established and broken down to three parts, data acquisition, modelling, and measuring complexity. Data acquisition generally tends towards statistical learning and data mining. Modelling turns to mainly two methods, cellular automata and agent-based modelling. The cellular automata is a simple mathematical model used to investigate self-organization in statistical mechanics. This model represents spatial dynamics and highlights local interactions, spatial heterogeneity, and large-scale aggregate patterns. Measuring complexity has demanded an inclusion of new metrics. These metrics include average path length, clustering coefficient, degree distribution, and spectral properties.

1.4 The audience as a complex system

Human behavior is usually studied qualitatively (under psychology or sociology) and is notoriously hard to quantify due to all the possible parameters and the difficulty in creating a controlled environment. The audience applause is an example of human behavior that is a collective of interacting agents with underlying dynamics. This allows us to treat the audience as a complex system in order to study its complexity and dynamics.

1.5 Known Studies and Models

There have been very little studies made on applause, down to the sound, its rhythm and its dynamics. One particular study treats the applause as a contagion that

propagates through the audience, allowing it to be modelled using an SIR-model. The SIR-model allows each unit in the model (for this case, each person henceforth referred to as an agent) to have 3 states, susceptible, infected, and recovered. Each agent is initally silent before the applause (susceptible). After which they start to applaud (infected). Finally stop clapping (recovered). Such a model would be appropriate if the agents no longer clap again after stopping, but cases exist where the agent my stop clapping prematurely, and then feel obliged to clap again due to the fact that the rest still continue to do so. With that, an SIS-like compartmental model with only two states(susceptible or infected) is adapted to properly account for such cases.

1.6 Problem Statement

The dynamics of the applause of an audience with N agents using an SIS epidemic model is presented and analyzed. *The same is done with an agent-based model with spatial effects.* The agents are connected via two dimensional lattice network and are initially fully connected. Later on, the agents will observe a modified, extended Moore neighborhood in order to incorporate spatial effects. The network is assumed to be homogeneous for simplicity; all agents share the same parameters for a given simulation. Agents are assumed to clap immediately after a performance, and may continue to do so depending on their given parameters. The research seeks only to find a correlation between the applause duration and the audience size, ultimately showing interdependence among the agents of the system. Why people applaud and to what are not investigated due to its unquantifiable nature.

Chapter 2

Monte Carlo Compartmental Method

Modelling epidemics has been done in order to study the mechanisms by which diseases spread. This helps in predicting how fast and far the disease can spread in order to control and prevent future outbreaks. By treating the applause as a diseases that spreads in the audience, the same tools to model epidemics can be used to model the applause. (i think this should be in the intro)

2.1 Compartmental Model

2.1.1 States

The proposed compartmental model separates the agents into two states, silent (S) and clapping (C). The silent state replaces the susceptible state while clapping state replaces the infected state. Intuitively, agents in state S are audience members who are not clapping while agents in state C are audience members who are clapping. The number of agents in state S, given by n_s, and the number of agents in state C, given by n_c, dictate the state of the system, given by $\vec{n}$ where $\vec{n} \equiv (n_c, n_s)$. It is assumed that the total number of agents N is fixed, where $N = n_c + n_s$ and $\vec{n}$ is fully specified

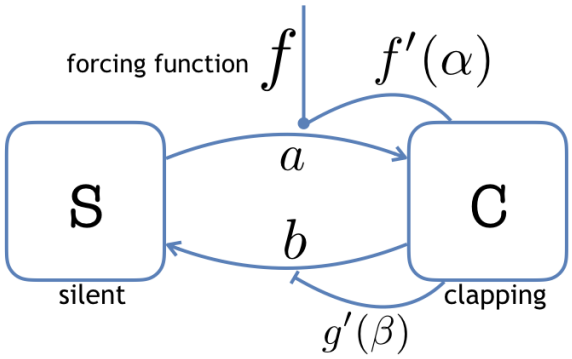


Figure 2.1: The compartmental model of audience applause based on the SIS epidemic model. Agents are either in state S or state C. Parameters a and b are the transitional probabilities. Functions f, f', and g' are further discussed.

by n_c alone.

2.1.2 Parameters

The parameters a and b are the transition parameters. They range from 0 to 1 to represent transitional probabilities. a controls the transition probability from S to C and b controls the transition probability from C to S. The respective probability transitions are given by R_1 and R_2 where

$$\mathrm{R}_1 \ : \ \mathrm{S} \xrightarrow{a} \mathrm{C} \tag{2.1}$$

$$\mathrm{R}_2 \ : \ \mathrm{C} \xrightarrow{b} \mathrm{S}. \tag{2.2}$$

2.1.3 Functions

An audience is initially silent until given a social cue to start clapping, such as the end of a performance. This corresponds in the model to all agents starting in state S, and then forcing the agents to undergo R_1. The function f forces the transition R_1 for an indicated time interval, τ, which is typically 2-3 seconds/iterations. Once f expires, the system behaves freely and its agents may undergo various R_1 and/or R_2 transitions depending on the given parameters.

Aside from social cues, audience members may start clapping simply because others are. The peer influence may cause agents unaffected by social cues to start clapping, or those who have already stopped clapping to clap again. This creates a feedback mechanism that initiates more people to clap if majority of the agents are clapping corresponding in the model to the function f'. The function f' incorporates this feedback mechanism and is parametrized by α:

$$f'(\alpha) = \alpha \frac{n_c}{N-1}, \tag{2.3}$$

where α ranges from 0 to 1 for probabilistic interpretations. The probability for a spontaneous R_1 transition is directly proportional to the fraction of the population in state C and α. The denominator is set to $N-1$ because an agent cannot spontaneously influence itself; it is only influenced by the rest of the population. The function is more effective when there are more agents in state C.

Finally, audience members may contain their own bias towards a social event and may applaud longer or shorter depending on the bias. Also, the continuous applause of the majority can inhibit those who are clapping to stop clapping. This corresponds in the model to the function g'. The factor g' incorporates the inhibition as a modulation function and is parametrized by β:

$$g'(\beta) = \frac{1}{1 + \beta\, n_C/(N-1)} \tag{2.4}$$

where $\beta \geq 0$, representing the bias. Higher β translates to agents less likely to undergo transition R_2 The equation is taken from the Michaelis-Menten equation, which aims to model enzyme kinetics[**?**].

This completes the differential equations for the reactions (2.1) and (2.2):

$$\frac{d}{dt} n_c = a(f + f' - f'f) n_s - b g' n_c \tag{2.5}$$

$$\frac{d}{dt} n_s = b g' n_c - a(f + f' - f'f) n_s \tag{2.6}$$

These equations are consistent with the assumption that the total audience size is fixed, that is $dn_c/dt = -dn_s/dt$. The derivation for the combined probability of the functions f and f' is shown in appendix (A.7.

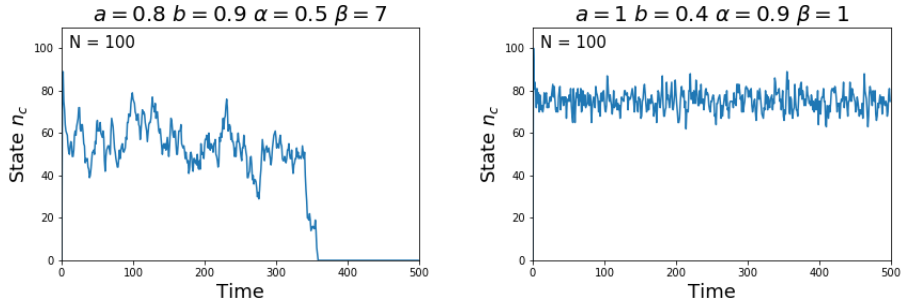
2.2 Simulation Alogrithm

The compartmental SIS-like model is confirmed by simulating the R_1 and R_2 process via an agent-based Monte Carlo method. For each iteration, every agent is assigned a random number that is compared to one of the transition probabilities:

$$P(R_1) = a(f + f' - f'f) \tag{2.7}$$

$$P(R_2) = bg' \tag{2.8}$$

Agents in state S undergo transition R_1 with probability shown in 2.7. Agents in state C undergo transition R_2 with probability shown in 2.8. A uniform random number, u, where $u \in [0, 1]$, is drawn from a random number generator and then compared to the corresponding probability P. If $u \leq P$, the appropriate transition is allowed to occur. The code developed to simulate the audience applause is provided in the appendix (A.2).



(a) A simulation with finite applause time.

(b) A simulation with a seemingly infinite and steady applause time.

Figure 2.2: Sample simulations given a fixed population $N = 100$. Simulation (a) is a typical, intuitive audience applause that has an end. Simulation (b) is non-trivial and counter-intuitive in that it seems to have no end.

The simulator outputs a graph of n_c versus time, where time is defined to be each iteration. Examples of such simulations are shown in figure 2.2. An emergent property of the system is that a certain set of parameters will simulate an audience applause

that does not end. Even if such cases do not exist in real life, further investigation of the steady-state applause may unravel the complexity of the system.

Chapter 3

Steady-state dynamics of the audience applause

A system is in steady-state when its behavior no longer changes in time. In the context of a real audience applause, the system rapidly spikes up in the number of people applauding, fluctuates, then slowly dies down till everyone stops clapping. This setup has the trivial steady-state of 0. However, simulations have shown that there are parameters where the steady-state is not zero, corresponding to n_c people clapping indefinitely. The value can be calculated by equating either (2.5) or (2.6) to zero, plugging in the given parameters a, b, α, β, and then solving for n_c.

3.1 Steady-state equation for different cases

The steady-state conditions are when the state $\vec{n}$ is fixed and when the forcing function expires. This is when $d\vec{n}/dt = 0$ and when $f = 0$. The derivation is shown in the appendix (A.3) Once these conditions are achieved, (2.5) and (2.6) are simplified to the steady-state equation for n_c:

$$\bar{a}(N - n_c)(N - 1 + \beta n_c) = b(N - 1)^2 \tag{3.1}$$

where $\bar{a} \equiv a\alpha$. The values of steady state n_c may be solved for any given $\bar{a}, b$, and β. Parametrizing (3.1) to time gives the phases space plot for the given parameters

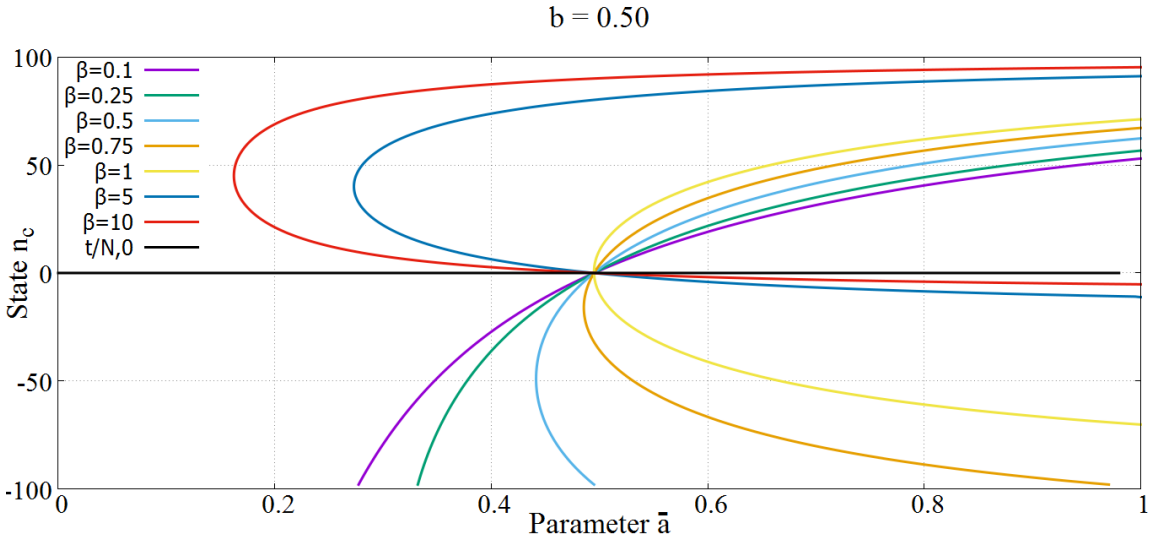


Figure 3.1: The phase space plot of the steady-state value n_c versus $\bar{a}$ for various β values and $b = 0.5$. Included is the trivial steaty-state solution $n_c = 0$

b and β, shown in figure 3.1. The phase space plots for various b values are shown in the appendix (A.4) Though the trivial steady-state solution is not included in (3.1), it has been included in the phase space. The coordinate ($\bar{a}$,n_c) on the curves represent the n_c value for a given $\bar{a}$ value at the given b and β value, where β is the color of the curve. Values below 0 are extraneous since n_c refers to the number of people clapping, which cannot be negative. This means that there are two possible steady-state values for $\beta \leq 1$, the trivial solution 0, and the non-trivial solution found on the appropriate curve. On the other hand, there are three possible steady-state values for $\beta > 1$, the trivial solution 0, and two non-trivial solutions found on the appropriate curve.

3.1.1 Critical Points

From observing all phase space plots of various b values, all β curves intersect, along with the trivial solution, at a specific point. Setting $n_c = 0$ in (3.1) gives us this critical point of intersection, $\bar{a}_1$:

$$\bar{a}_1 = \frac{b(N-1)}{N} \tag{3.2}$$

which for $N \gg 1$ results to the observed $\bar{a}_1 \approx b$. Solution for the critical point is shown in the appendix (A.5

The non-trivial solution to the steady-state equation (3.1) is quadratic, which does not include the trivial solution. This results to two non-trivial n_c solutions for a given $\bar{a}$. For $\beta \leq 1$, the non-trivial steady-state solutions appear uniquely for $\bar{a} > \bar{a}_1$ since the second n_c value is negative and therefore, extraneous. However, when $\beta > 1$, two non-trivial solutions appear between a new critical value of $\bar{a} = \bar{a}_2$, corresponding to the vertex of (3.1), and $\bar{a}_1$. For $\bar{a} > \bar{a}_1$, the non-trivial steady-state solution appears uniquely once again, similar to $\beta \leq 1$. The $\bar{a}_2$ is given by

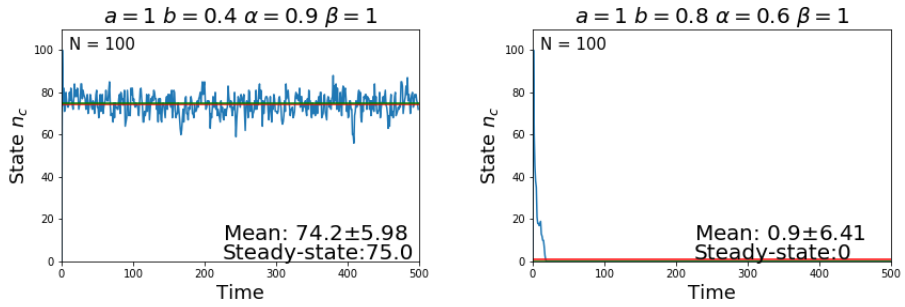
$$\bar{a}_2 = \frac{b(N-1)^2}{(N-n_c^*)(N-1+\beta n_c^*)} \tag{3.3}$$

where $n_c^* \equiv [1 + (\beta - 1)N]/2\beta$. Derivation is shown in the appendix (A.5 This non-trivial solution however, is unstable upon substitution with $\ddot{\vec{n}}$ resulting to $\ddot{\vec{n}} < 0$. Thus, the middle branch in the range $\bar{a} \in (\bar{a}_2, \bar{a}_1)$ is an unstable steady-state.

3.2 Simulation experiments

Analytical results are confirmed by simulation. The number of iterations, t, is set to be much greater than the duration of the forcing function,τ. Figure 3.4 shows sample simulations along with the analytical steady-state and the average of the system. The analytical steady-state is plotted in green while the mean is plotted in red. As observed, the simulation closely resembles the analytical value. The system settles to its steady-state very early in the simulation. It can be safely assumed that the n_c values at $t \gg \tau$ are equivalent to n_c for $t \to \infty$.

10 sample runs were performed with different initial random number generator



(a) A simulation with a non-trivial steady state of 75. The average n_c is close to the analytic value.

(b) A simulation with a trivial steady state of 75. The average n_c value is approximately 0.

Figure 3.2: Sample simulations that compare the analytical steady-state of the given parameters and the mean n_c value of the system.

states and the final n_c value for each iteration were recorded. The mean and standard deviation of the simulated steady-state values are then computed for each parameter space $(\bar{a}, b, \beta)$. The data points were then plotted with the corresponding parametrized differential curve shown in Fig. 3.3. All code used is provided in the appendix (A.6), as well as more simulation experiments for various b values (A.7

For $\beta \leq 1$, the simulation is consistent with the trivial steady state solution until it reaches the critical point $\bar{a} = b$, after which follows non-trivial, non-extraneous solution consistent with (3.1). For $\beta > 1$, the simulation follows the trivial steady-state and then breaks away and approaches the vertex of (3.1) at $\bar{a}_2$, after which, continues to follow the upper branch of (3.1). The analytical solutions fail to predict the bifurcation from the trivial solution to the vertex of the curve. Also, the significance of the lower branch for $\beta > 1$ curves is unknown and warrants further investigation. The vertex is said to be unstable due to the fact that it unreliably and unpredictably settles to either trivial or non-trivial solution. The lower branch may share this instability.

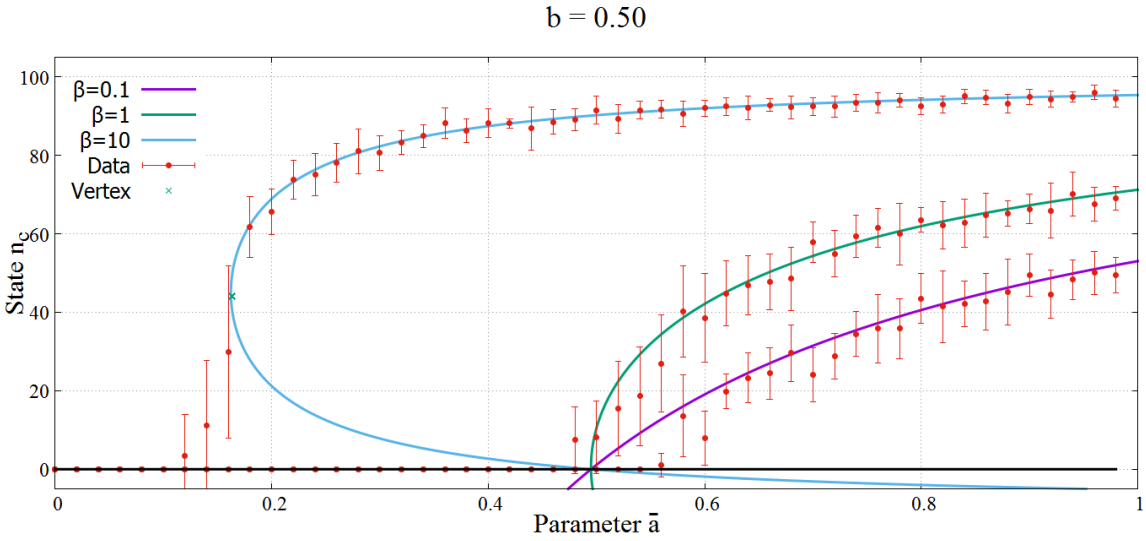


Figure 3.3: The simulated steady-state values plotted against the analytic steady-state curves. The error bars provide the extent of deviation in simulations. The vertex is represented with the cross.

3.2.1 Unstable points

For the previous simulation experiments, all agents start at state S and then forced to transition to state C. To investigate the properties of the lower branch on the non-trivial solution, we vary the number of agents that start the simulation in state C and remove the forcing function. This allows us to determine where the system will settle given the specific starting n_c value, more particularly for n_c values near the lower branch of the non-trivial solution. The code that generate the vector graphs is provided in the appendix (A.8)

Shown in figure 3.4 are the new phase space graphs that include arrows that point to where that n_c value for the given $\bar{a}$ settles. More vector graphs are available in the appendix (A.9 The color of the arrows represents the probability of going towards that direction. Intuitively for $\beta \leq 1$, all n_c values point towards the trivial solution for all $\bar{a}$ values less than $\bar{a}_1$. All n_c with $\bar{a}$ values greater than $\bar{a}_1$ point towards the non-trival solution. For $\beta > 1$, what differs is the behavior at the vertex and the lower branch of the non-trivial solution. n_c values above the vertex have an estimated 50%

to either settle at the vertex or zero, while n_c values below the vertex absolutely settle to zero. Points slightly above the lower branch tend more towards settling to the upper branch but still have a chance to settle at 0. Likewise for points below the lower branch. Points within the lower branch have a 50% of settling to either trivial or non trivial solution.

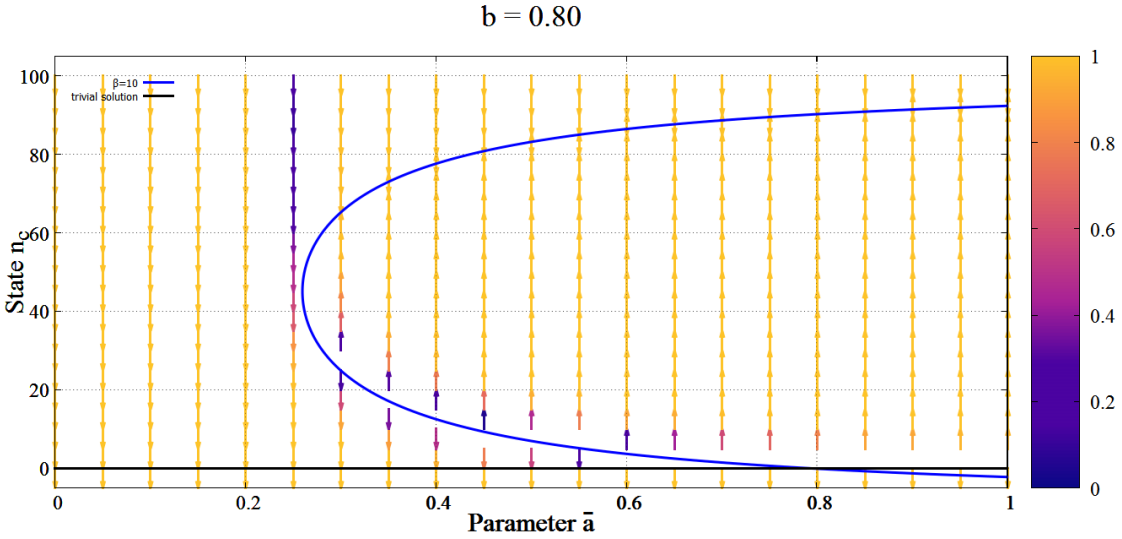


Figure 3.4: Vector graph for parameters $b = 0.8, \beta = 10$. The origin of the vector represents the initial n_c and corresponding $\bar{a}$ value. The direction it points reveals whether it settles to the trivial or non-trivial solution. The color represents the probability of settling towards the directed steady-state.

Chapter 4

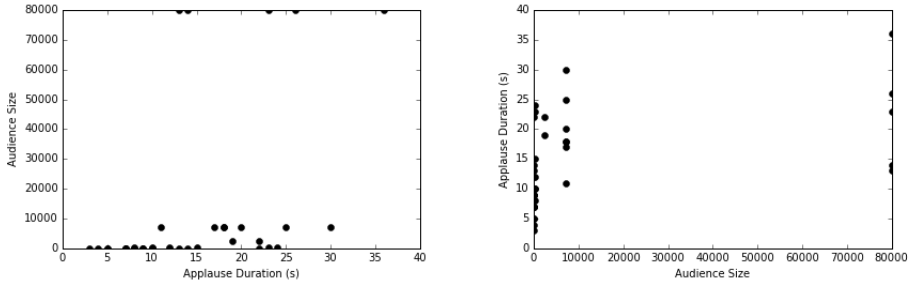
Incorporating Spatial Effects

After analysing the model, it was then compared to real-life applause. The applause duration of varying audience sizes was recorded and graphed. Selection criteria for the data included a definite applause time that was fully-recorded and uncut, and a known population. Because of this, most of the videos are live concerts and musical performances. The data points were plotted in different ways in order to find an appropriate linear trend in figure 4.1.

Figure 4.1 (d) makes the most sense; the applause duration of the audience is dependent on the size of the audience. The question now is whether or not the compartmental model can recreate 4.1 (d). Sadly, the current model is scale-free; changing the population does not affect the applause duration for a given set of parameters. This is because of the assumption that the network is fully connected, meaning that an audience member in front is influenced the exact same way by the whole audience as an audience member at the back. As this does not seem to be the case in real-life applause, spatial effects must be incorporated by modifying the feedback function $f'(\alpha)$.

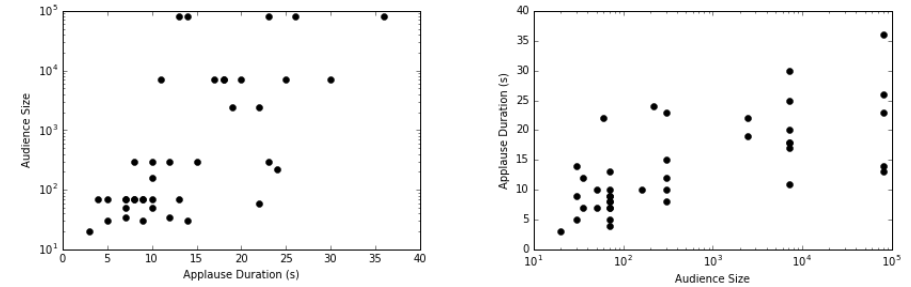
4.1 Different configurations for field of vision

The field of view configurations represent the audience members that may influence the reference agent. In real-life, the reference agent is only influenced by the



(a) Graphing size versus applause duration.

(b) Graphing applause duration versus size.

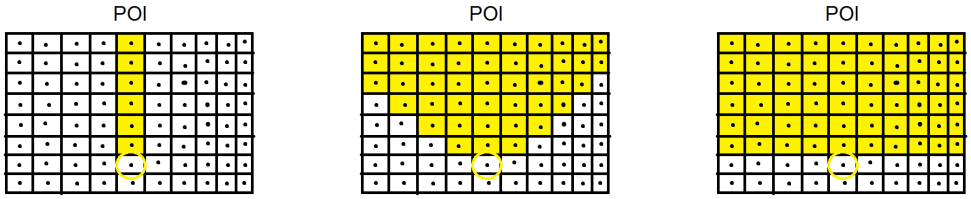


(c) Graphing size (in the logarithmic scale) versus applause duration

(d) Graphing applause duration versus size (in the logarithmic scale).

Figure 4.1: Different graphs of the data points of real-life applause.

audience members it can see. The network is no longer fully-connected, observing a modified, extended Moore neighborhood. It is extended since it considers all agents till the front row of the system, and not just those 1 unit away. Is is modified since it does not consider all directions; the directions considered are affected by the configuration. '0 degrees' represents all audience members only directly in front of the reference agent until the very front. '180 degrees' represents all rows ahead of the reference agent. '90 degrees' represents a spatial configuration somewhere in between '0 degrees' and '180 degrees'; considering all agents from the north-east direction to the north-west. The configurations are shown in figure 4.2.



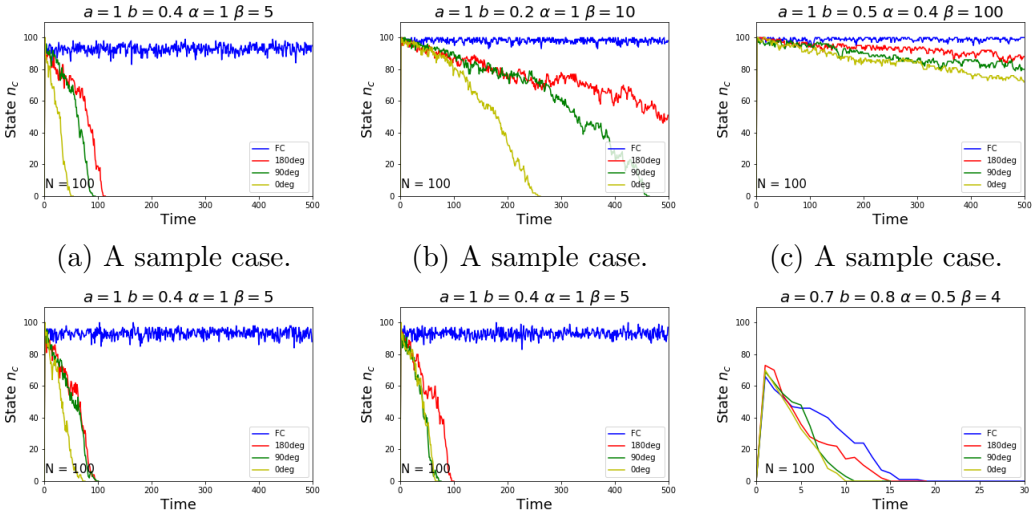
(a) '0 degrees' (b) '90 degrees' (c) '180 degrees'

Figure 4.2: Different field-of-view configurations. The areas highlighted in yellow are what can influence the encircled reference agent.

4.2 Simulating the different spatial configurations

Initially, $f'(\alpha)$ bases the fraction of n_c throughout the whole population. After incorporating spatial effects, that fraction is only within the field-of-view of the reference agent. Even after spatial effects, the feedback function is still parametrized by α. The code developed is provided in the appendix (A.10).

Shown in figure 4.3 are simulations with different parameters (a, b, α, β) comparing the different feedback functions, the original fully-connected network, '180 degrees', '90 degrees', and '0 degrees'. Graphs (a), (b), and (c) show that in our system feedback functions are less effective in less connected networks. There is clearly a downward trend (starting from FC, '180', '90', then '0') when it comes to applause duration. Graphs (d) and (e) show that '90 degrees' can be similar to either remaining configurations, as well as be somewhere in between. Generally, simulations incorporating any form of field-of-view feedback do not have a non-trivial steady-state. Graphs (b) and (c) show cases where the simulations have not ended, but exhibit an obvious downward trend. Given more iterations, the simulations would end, unlike the fully-connected ones. Graph (g) shows a special case where all simulations have a trivial steady-state of zero. One thing to note is that the analysis done for the fully-connected system (phase space graphs, critical and unstable points) cannot be applied to the systems with spatial effects. A different steady-state equation must be setup to reflect how each agent is influenced differently. Such analysis will no longer



(a) A sample case.

(b) A sample case.

(c) A sample case.

(d) A case where '90 degrees' is similar to '0 degrees'.

(e) A case where '90 degrees' is similar to '180 degrees'.

(f) A case where all configurations have a trivial steady-state 0.

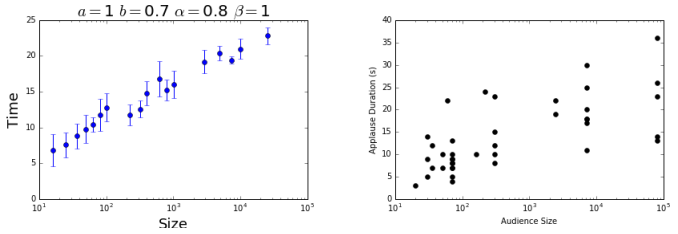
Figure 4.3: Different graphs comparing effect of different feedback functions under different parameters.

be pursued as it is outside the skill set of the researcher.

4.3 Finding the parameters (a, b, α, β) for real-life applause

Recalling, incorporating spatial effects to the feedback function effectively reduces the number of agents influencing a reference agent. Increasing the population size should increase the feedback since the agents in the back row of a 100x100 grid should 'see' more people than those in the back row of a 10x10 grid. This effectively eliminates the original problem of the fully-connected network being scale free. It is now possible for the applause duration to be dependent on the population. This phenomena is investigated by simulating a set of parameters (a, b, α, β) with varying populations. For simplicity, the system grid will always be a 2-dimensional lattice of equal lengths. Also, the spatial configuration used will be '180 degrees'. The total

population sample will be perfect squares ranging from 10^1 to 10^4. The code developed is provided in the appendix (A.12). For this case, each iteration is considered 1 second. Five trials were performed per sample population. Due to insufferable run times, if a parameter set (a, b, α, β, N) returned a trial with applause duration greater than 60 seconds, it is marked as 0 seconds and no further trials are performed. This means that graphs with data points of 0 seconds are to be disregarded.



(a) Simulated points showing population size dependence.

(b) Real-life data points on population size dependence.

Figure 4.4: Comparing the simulated and actual graph of applause duration versus population size. Both increase linearly in the logarithmic scale.

Shown in figure 4.4 is the parameter set $(a = 1, b - 0.7, \alpha = 0.8, \beta = 1)$ that best recreates the real-life applause. Similar parameter sets are shown in the appendix (**??**) Though this is the best fit parameter set, it still does not fully emulate the real-life applause. As seen, the data points for the real-life applause is more deviant compared to the simulation. The model may still be incomplete, or the data points for real-life applause is not enough. Further investigation is warranted, as well as an in-depth anaylsis on the new model.

Chapter 5

Conclusions and Recommendations

A compartmental model SIS-like model was successfully constructed to study the complexity of the audience applause, as well as simulate it with an agent-based Monte Carlo method. The original, fully-connected system exhibited parameter sets that express a non-trivial steady-state solution. The dynamics of the given system were studied, uncovering critical points in the phase space plot, as well as unstable points. Further modifications were made to the compartmental model in order to incorporate spatial effects, as such effects more closely resembled a real-life applause. Doing so removed the scale-free property of the original fully-connected system. This allowed simulations to show a correlation between applause duration and audience size. The simulations were backed by data points of real-life applause. Analysis of the model with incorporated spatial effects is necessary in order to understand the dynamics of the audience applause. This will allow further development of the model, which in turn, will more closely resemble the real-life applause.

Appendix A

Appendix

A.1 Combined probability of functions f and f'

The forcing function f permits agents to undergo R_1 and is represented with a probability p_1. The probability that R_1 does not occur is $q_1 = 1 - p_1$. Likewise for the function f' to undergo R_1 is represented with a probability p_2, and non-occurring probability q_2. Combining the probabilities p and q results to

$$p_1 + q_1 = 1 \tag{A.1}$$

$$p_2 + q_2 = 1 \tag{A.2}$$

Multiplying the equations (A.1) and (A.2) and simplifying

$$(p_1 + q_1)(p_2 + q_2) = 1 \tag{A.3}$$

$$p_1p_2 + p_1q_2 + p_2q_1 + q_1q_2 = 1 \tag{A.4}$$

q_1q_2 represents neither event occuring thus

$$p_{net} = 1 - q_1q_2 = 1 - (1 - p_1)(1 - p_2) \tag{A.5}$$

$$= 1 - (1 - p_1 - p_2 + p_1p_2) \tag{A.6}$$

$$= p_1 + p_2 - p_1p_2 \tag{A.7}$$

Replacing (**??**) with the functions gives

$$p_{net} = f + f' - f'f \tag{A.8}$$

which is used in (2.7)

A.2 Library of applause functions

```
from random import random
from numpy import zeros, sqrt, sum, nan_to_num
from math import isnan

#function that forces agents to go to state C
def force_func(t, end):
    if t < end:
        return 1
    else:
        return 0

#f'(alpha); feedback based on the number of ppl already in
   state C; the more ppl in state C, the more ppl will clap
def feedback_alpha(alpha, nC, population):
    return alpha * nC / (population - 1)

#g'(beta)
def feedback_beta(beta, nC, population):
    return 1 / (1 + beta * nC / (population - 1))

#spatial-dependent feedack, you can control the 'radius' of
   the reference agent
#taper refers to how many to add at the ends;radius = 0 taper
    = 1 is '90deg'
def feedback_space(alpha,system,system_row, system_column, N,
    M, radius, taper):
    applause_state = []
```

```
    for i in range(system_row):
        radius_mech = 0
        applause_state.append(system[i,system_column])
        while radius_mech != radius + taper*(system_row - i):
            radius_mech += 1
            if system_column + radius_mech < M:
                applause_state.append(system[i,system_column
                    + radius_mech])
            if system_column - radius_mech > -1:
                applause_state.append(system[i,system_column
                    - radius_mech])
    return alpha*nan_to_num(sum(applause_state)/len(
        applause_state))

def feedback_180deg(alpha,system,system_row, M): #reverted to
    simpler feedback space functions since runs took too long
    applause_state = [0]
    for i in range(system_row):
        applause_state.append(sum(system[i]))
    return alpha*nan_to_num(sum(applause_state)/(system_row*M
        ))

#quadratic equation
def quad_eq(x,y,z,sign):
    return (-y + sign * (sqrt((y ** 2) - 4 * x * z))) / 2 * x

#do i still need this??!?!
def frange(start, stop, step):
    i = start
```

```
    while i < stop:
        yield i
        i += step

#creates 'network' of audience
def audience(N,M,C):
    agents = zeros((N,M))

    for i in range(N):
        for j in range(M):
            if sum(agents) == C:
                break
            else:
                agents[i][j]=1

    return agents

#the actual simulator
def app_sim(aStoC, bCtoS, alpha, beta, N, M, C, t, t_1):
    population = N * M
    AGENT = audience(N, M, C)
    graph = []

    for k in range(t):
        nC = sum(AGENT) #number of people clapping
        graph.append(nC)
        for i in range(N):
            for j in range(M):
                if AGENT[i,j] == 0:
```

```
                    if random() <= aStoC * (1 - (1-force_func
                        (k, t_1)) * (1 - feedback_alpha(alpha,
                         nC, population))):
                         AGENT[i,j] += 1
                else:
                    if random() <= bCtoS * feedback_beta(beta
                        , nC, population):
                         AGENT[i,j] -= 1
    return graph

#sim with spatial dependence specific to 180 deg
def sim_space(aStoC, bCtoS, alpha, beta, N, M, C, t, t_1):
    population = N * M
    AGENT = audience(N, M, C)
    graph = []
    zeroCount = 0

    for k in range(t):
        nC = sum(AGENT) #number of people clapping
        if nC == 0:
            zeroCount += 1
        graph.append(nC)
        for i in range(N):
            for j in range(M):
                if AGENT[i,j] == 0:
                    if random() <= aStoC * (1 - (1-force_func
                        (k, t_1)) * (1 - feedback_180deg(alpha
                        ,AGENT,i, M))):
                         AGENT[i,j] += 1
```

```
                else:
                    if random() <= bCtoS * feedback_beta(beta
                        , nC, population):
                        AGENT[i,j] -= 1
        if zeroCount == 6:
            break
            return graph
    return graph

#graphs theoretical steady_state based on parameters
def steady_nC(aStoC, bCtoS, alpha, beta, population, sign):
    if beta == 0:
        if alpha == 0:
            return (aStoC * population) / (aStoC+bCtoS)
        else:
            if sign == 1:
                return population - (bCtoS * (population - 1)
                    ) / (aStoC * alpha)
            else:
                return 0
    else:
        if alpha == 0:
            return 0
        else:
            if sign == 0:
                return 0
            else:
```

```
    if isnan(quad_eq(1, ((population*aStoC*alpha*
        beta)-(population*aStoC*alpha)+(aStoC*
        alpha))/(-aStoC*alpha*beta), (((population
        **2)*aStoC*alpha) - (population*aStoC*
        alpha) - ((population**2)*bCtoS) + (2*
        population*bCtoS) - (bCtoS))/(-aStoC*alpha
        *beta), sign)) == True:
        return 0
    else:
        return quad_eq(1, ((population*aStoC*
            alpha*beta)-(population*aStoC*alpha)+(
            aStoC*alpha))/(-aStoC*alpha*beta), (((
            population**2)*aStoC*alpha) - (
            population*aStoC*alpha) - ((population
            **2)*bCtoS) + (2*population*bCtoS) - (
            bCtoS))/(-aStoC*alpha*beta), sign)
```

A.3 Deriving the steady-state equation

Setting $\frac{d\vec{n}}{dt} = 0$,

$$\frac{d}{dt}n_c = 0 \tag{A.9}$$

$$a(f + f' - f'f)n_s - bg'n_c = 0 \tag{A.10}$$

$$a(f + f' - f'f)n_s = bg'n_c \tag{A.11}$$

$$\tag{A.12}$$

Note: $N = n_c + n_s, n_s = N - nc$, and $f = 0$.

$$af'(N - nc) = bg'n_c \tag{A.13}$$

Plugging in (2.3) and (2.4) to (A.13):

$$a(\alpha\frac{n_c}{N-1})(N - n_c) = b(\frac{1}{1 + \frac{\beta n_c}{N-1}})(n_c) \tag{A.14}$$

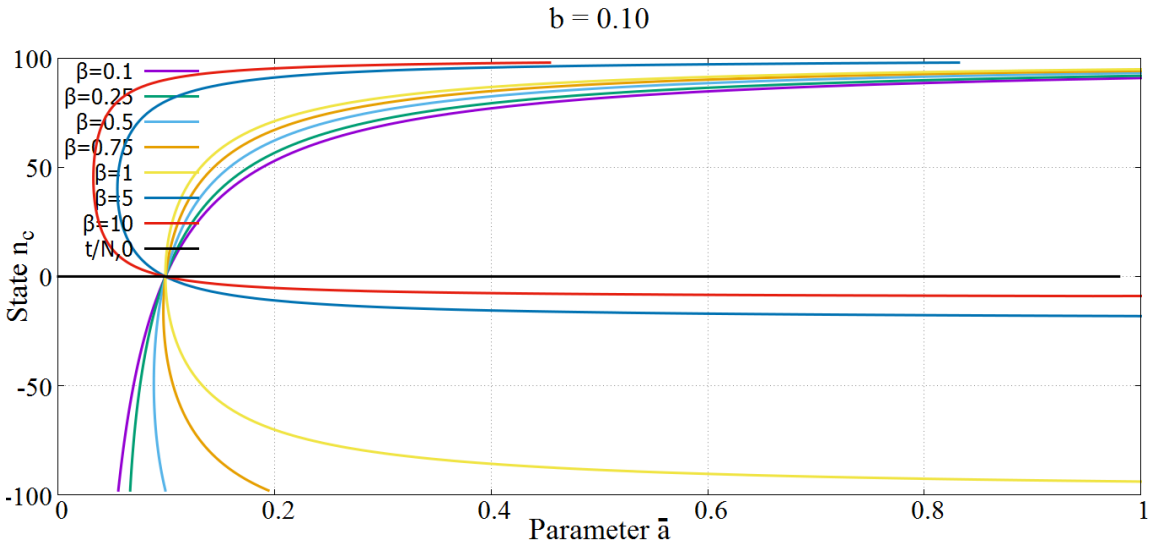
$$\frac{a\alpha(N - n_c)}{N-1}n_c = b\frac{N-1}{N-1+\beta n_c}n_c \tag{A.15}$$

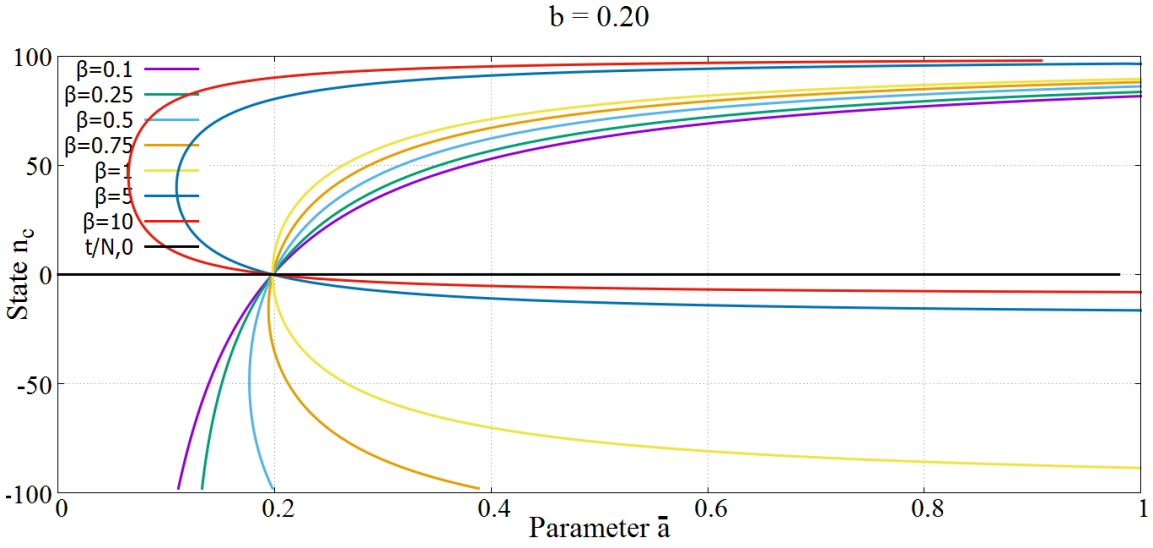
$$a\alpha(N - n_c) = b\frac{(N-1)^2}{N-1+\beta n_c} \tag{A.16}$$

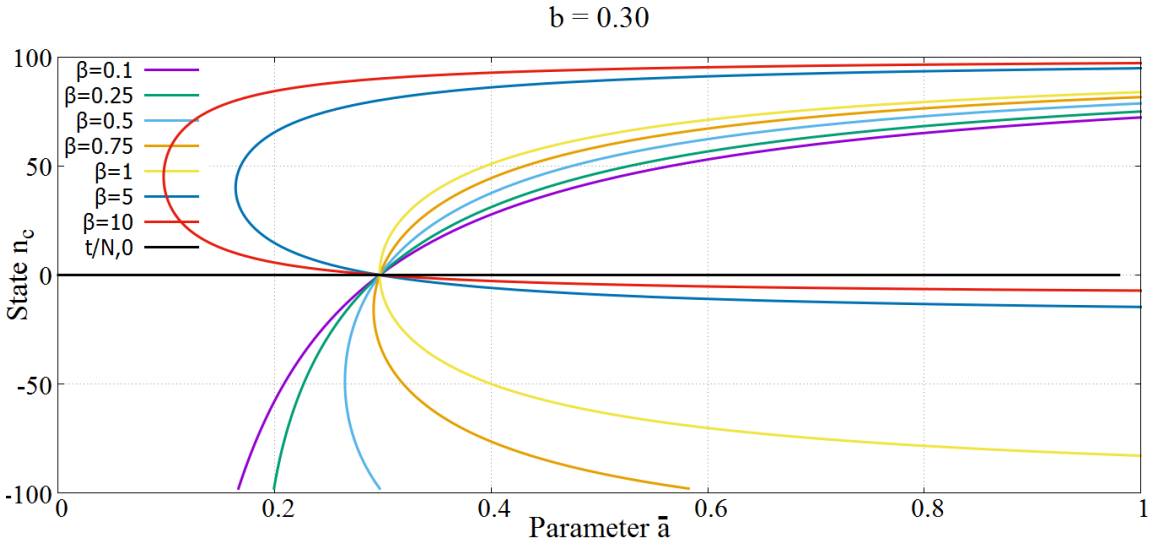
$$a\alpha(N - n_c)(N - 1 + \beta n_c) = b(N-1)^2 \tag{A.17}$$

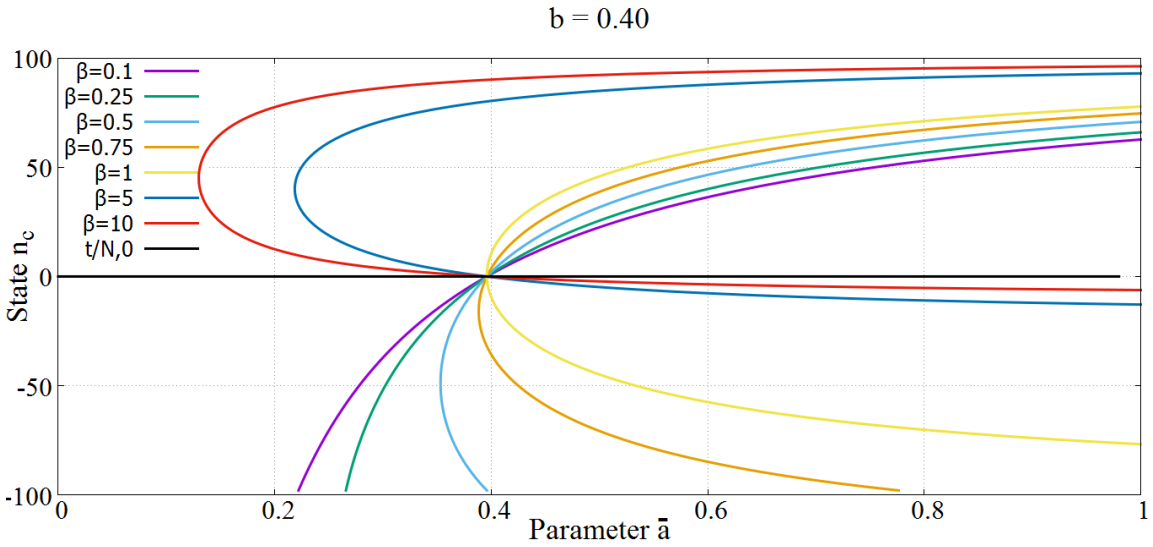
A.4 Steady-state Phase Space

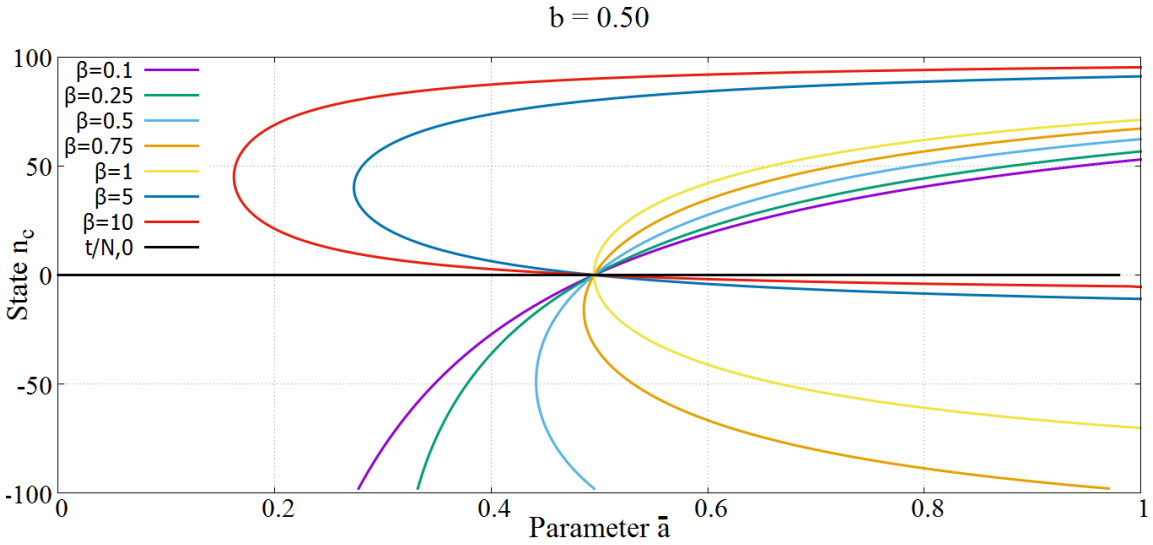
Shown are the phase space plots for various b values.

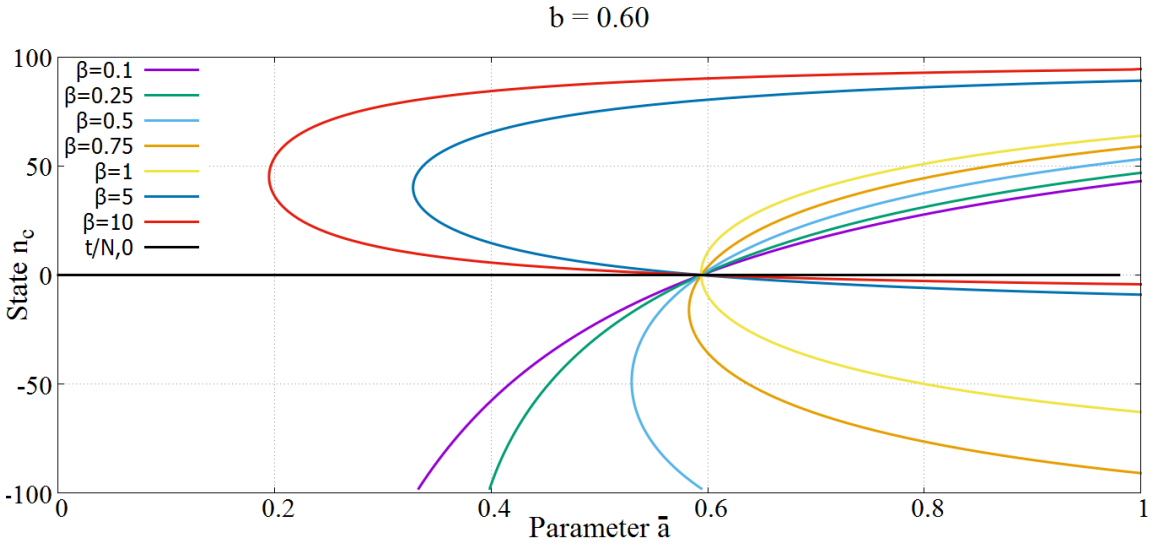


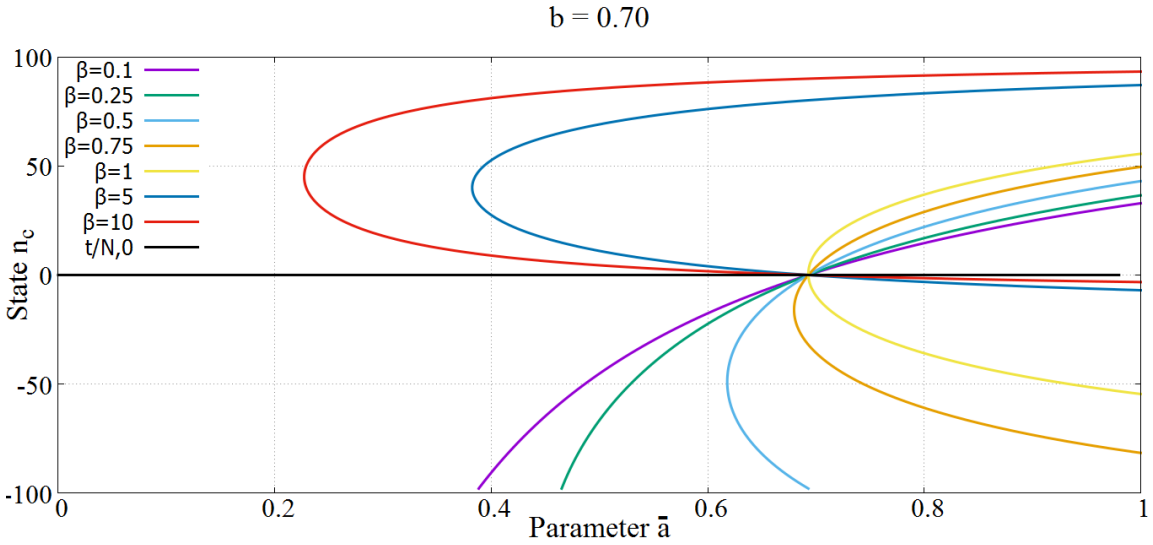


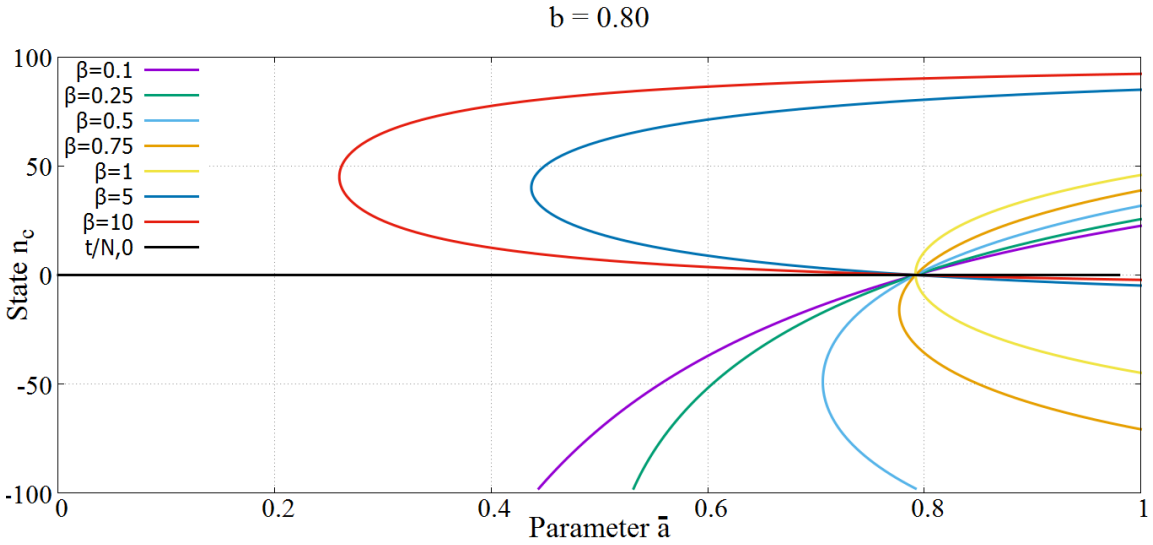

b = 0.30
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1

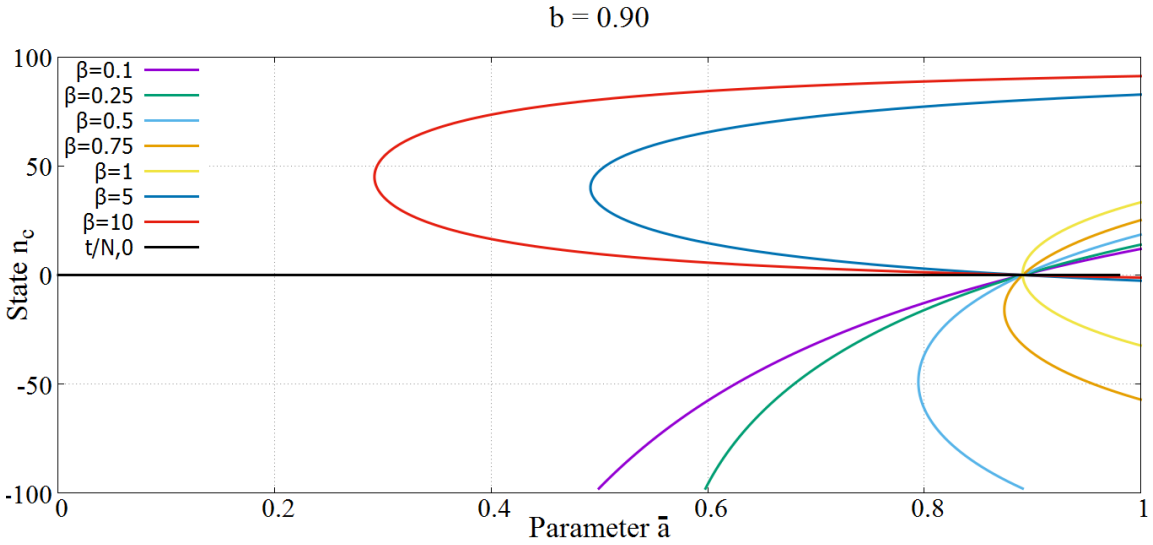

b = 0.40
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1

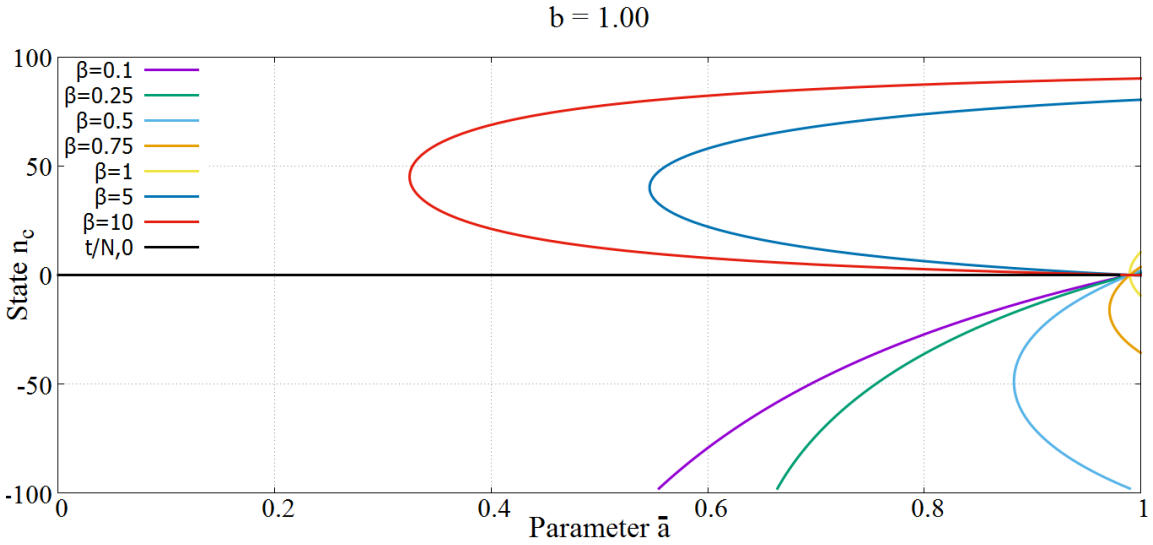

b = 0.50
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1


b = 0.60
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1


b = 0.70
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1


b = 0.90
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1


b = 1.00
β=0.1
β=0.25
β=0.5
β=0.75
β=1
β=5
β=10
t/N,0
State n_c
Parameter ā
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1

A.5 Critical points

Setting $n_c = 0$ in the steady-state equation (3.1) gives us this critical point of intersection

$$\bar{a}(N - n_c)(N - 1 + \beta n_c) = b(N-1)^2 \tag{A.18}$$

$$\bar{a}N(N-1) = b(N-1)^2 \tag{A.19}$$

$$\bar{a} = \frac{b(N-1)^2}{N(N-1)} \tag{A.20}$$

$$\bar{a}_1 = \frac{b(N-1)}{N} \tag{A.21}$$

$$\tag{A.22}$$

On the other hand, $\bar{a_2}$ is arrived by plugging in a special n_c value, $n_c^* \equiv [1 + (\beta - 1)N]/2\beta$.

$$\bar{a}(N - n_c^*)(N - 1 + \beta n_c^*) = b(N-1)^2 \tag{A.23}$$

$$\bar{a}_2 = \frac{b(N-1)^2}{(N - n_c^*)(N - 1 + \beta n_c^*)} \tag{A.24}$$

A.6 Steady-state Simulations

Shown is the code that outputs the simulations with the analytical steady-state as well as the mean of the system for a single set of parameters.

```
############################################################
#Simulates applause for a specific set of parameters#
#Displays the appropriate steady state value        #
#Parameters can be adjusted accordingly below       #
############################################################


import applause_functions as app
import matplotlib.pyplot as plt
import numpy as np

N = 10
M = 10
population = N * M
time = 500
t_1 = 2
C = 0
aStoC = 1
bCtoS = 0.8
alpha = 0.6
beta = 1

fig = plt.figure()
ax = fig.add_subplot(111)
```

```
sim = app.app_sim(aStoC, bCtoS, alpha, beta, N, M, C, time,
    t_1)
plt.plot(sim)
ax.text(0.45*time,1,'Steady-state:'+str(round(app.steady_nC(
    aStoC, bCtoS, alpha, beta, population, 1),0)), fontsize
    =20)
ax.text(0.45*time,10,'Mean: '+str(round(np.mean(sim),1))+'$\
    pm$'+str(round(np.std(sim),2)), fontsize=20)
ax.text(10,100,'N = '+str(population), fontsize=15)
plt.axhline(np.mean(sim), color = 'r')
plt.axhline(app.steady_nC(aStoC, bCtoS, alpha, beta,
    population, 1), color = 'g')
plt.title('$a = $'+str(aStoC)+' $b = $'+str(bCtoS)+r' $\alpha
    =$'+str(alpha)+r' $\beta=$'+str(beta),fontsize=20)
plt.xlabel('Time',fontsize=18)
plt.ylabel('State'+r' $n_{c}$',fontsize=18)
plt.xlim(0,time)
plt.ylim(0,110)
plt.savefig('ssB.png')
plt.show()
```

Following is the code used to generate the data points of the simulated steady-state values to be plotted against the analytic steady-state solutions.

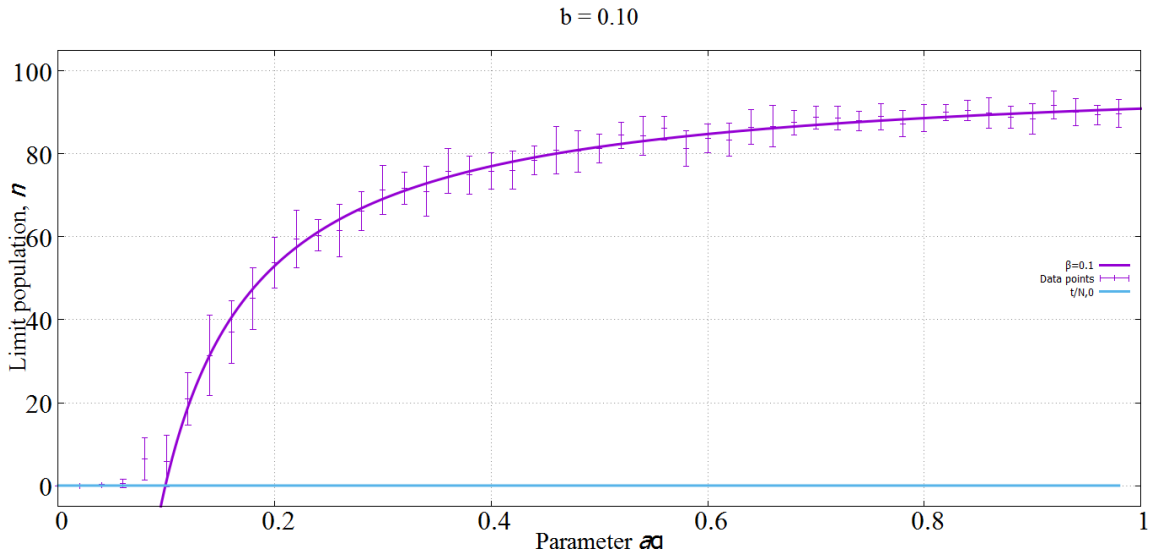
```
#Takes the last point of the simulation (Trials) times then
    calculates mean and std#
#Does so for varying a-bar values
                                                        #
#Creates .csv file with columns: a-bar, mean, std
                                        #
```

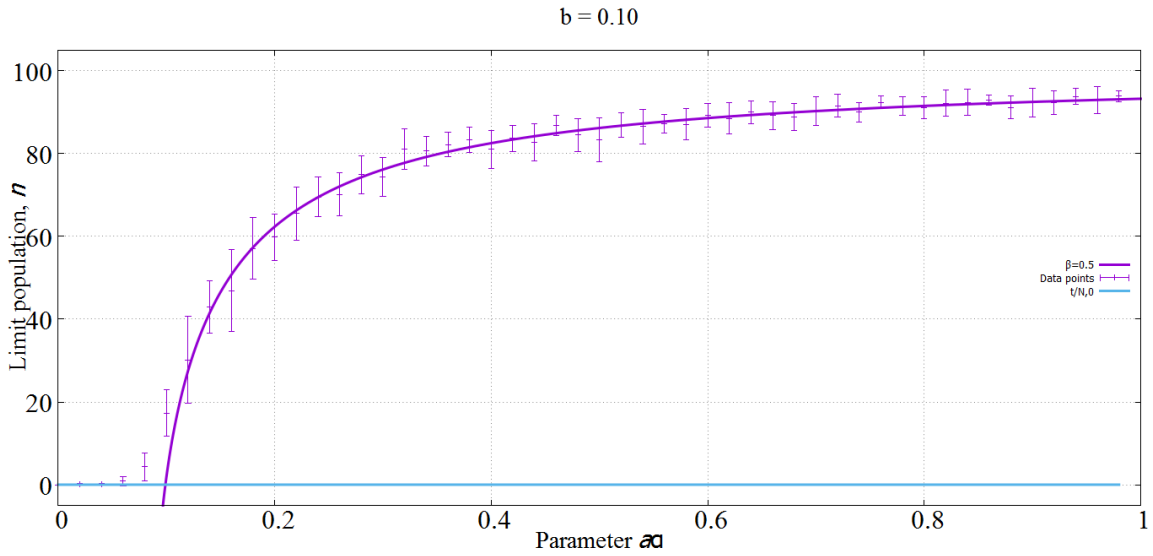
```
#Plots the data points

   #
#Parameters can be adjusted accordingly below
                                                #


import applause_functions as app
import matplotlib.pyplot as plt
import numpy as np

#####Simulation Parameters#####
N = 10
M = 10
startPop = 20
time = 100
t_1 = 0
aStoC = 1.0
bCtoS = 0.8
alpha = 1
beta = 10
###Point Generator Parameters###
Trials = 50
interval = 0.02
###################################
def pointGen(aStoC, bCtoS, beta, N, M, startPop, time, t_1,
    Trials, interval):

     meanlist = []
```

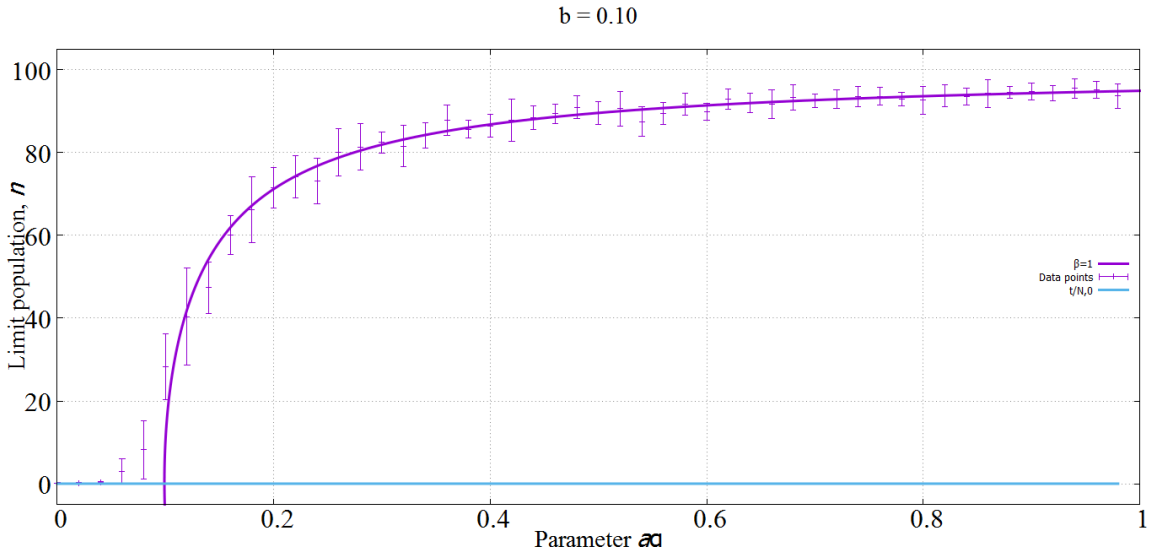
```
    stdlist = []
    ilist = []
    for i in np.arange(0,1.0,interval):
        list_a = []
        for j in range(Trials):
            list_a.append(app.app_sim(aStoC, bCtoS, i, beta,
                N, M, startPop, time, t_1)[time-1]) #takes the
                 last number from the appsim vector
        mean = np.mean(np.array(list_a))
        std = np.std(np.array(list_a))
        meanlist.append(mean)
        stdlist.append(std)
        ilist.append(i)

    ilist = np.array(ilist)
    meanlist = np.array(meanlist)
    stdlist = np.array(stdlist)

    #writes ilist,meanlist, and stdlist into a txt file
    filename = 'b = ' + str(bCtoS) + ', beta = ' + str(beta)
      + ', startPop = ' + str(startPop) + '.txt'
    with open(filename,'w') as f:
        lis=[ilist,meanlist,stdlist]
        for x in zip(*lis):
            f.write("{0}\t{1}\t{2}\n".format(*x))

    return (ilist,meanlist,stdlist)
```

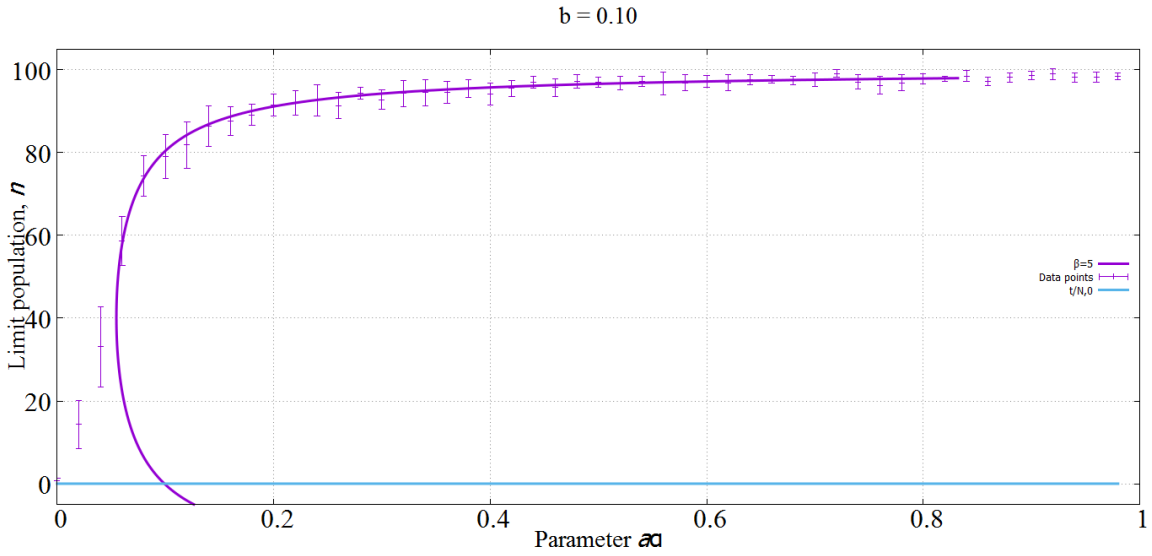
A.7 Steady-state Simulations Results

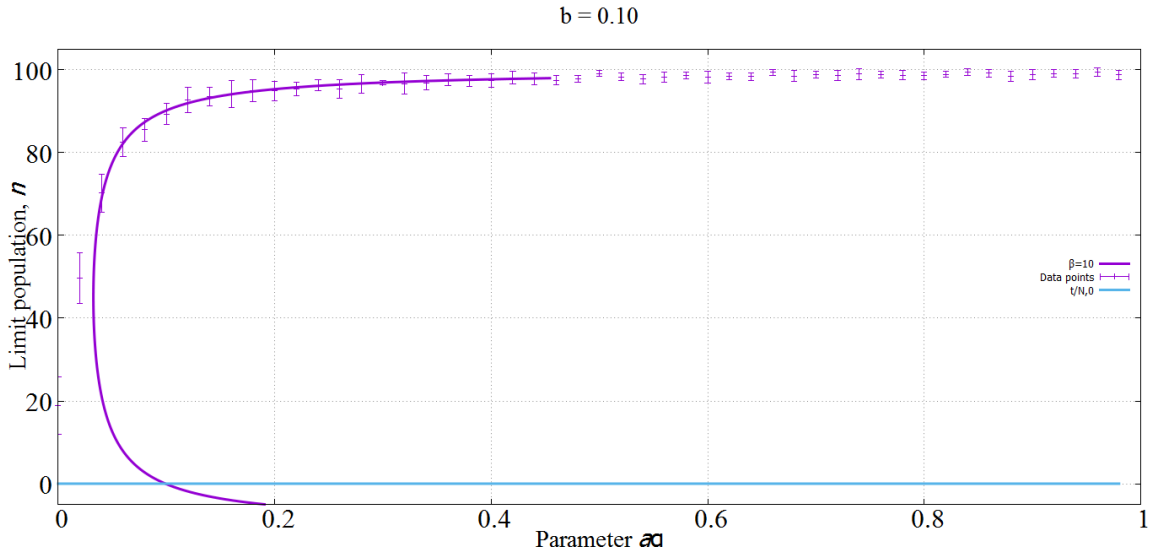
Shown are the phase space plots along with the simulated data points using the same parameters for various b and β values.

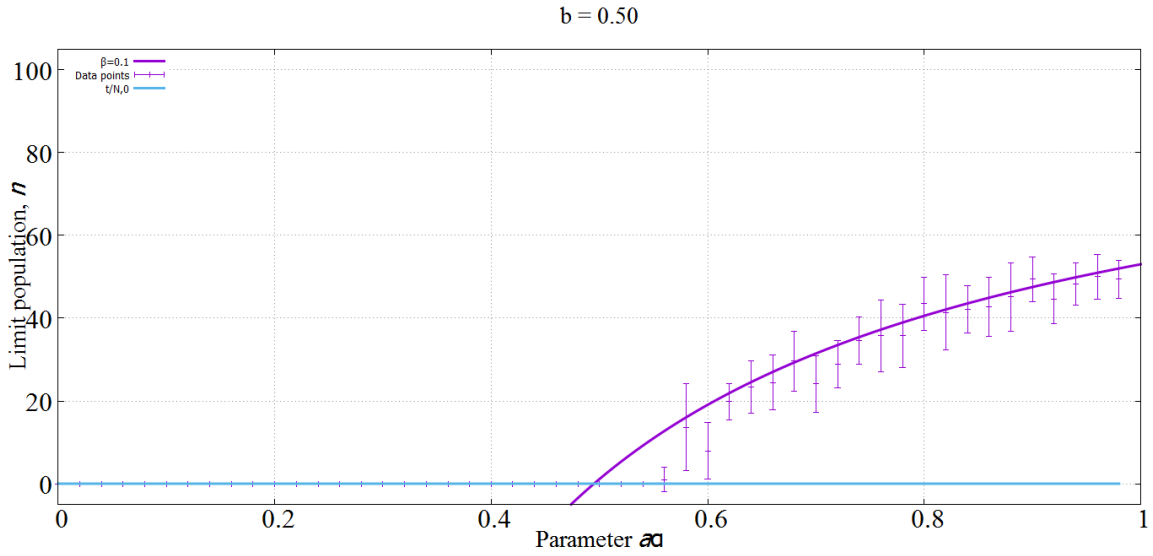


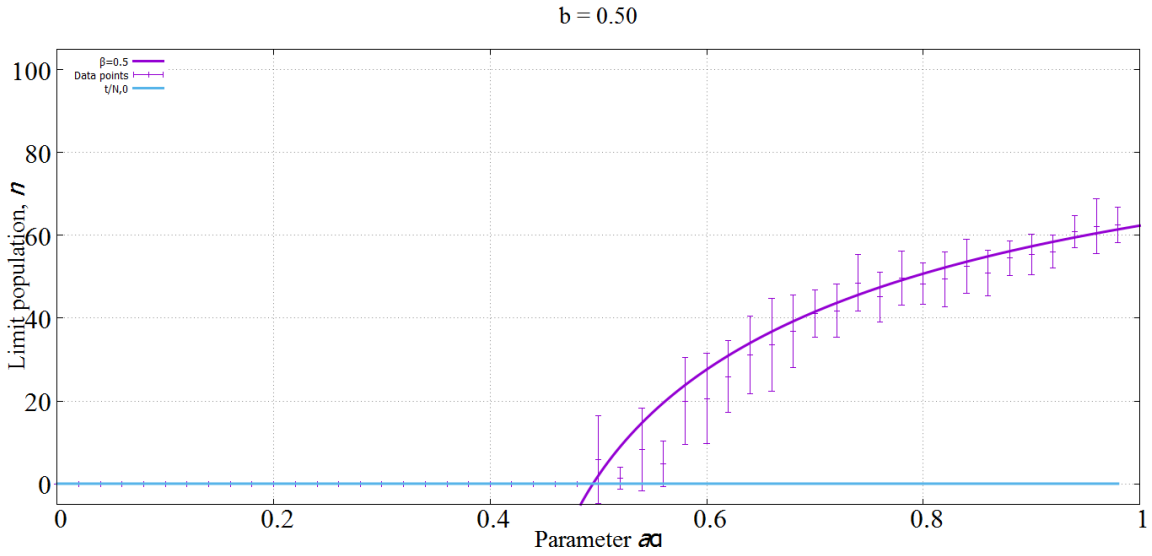


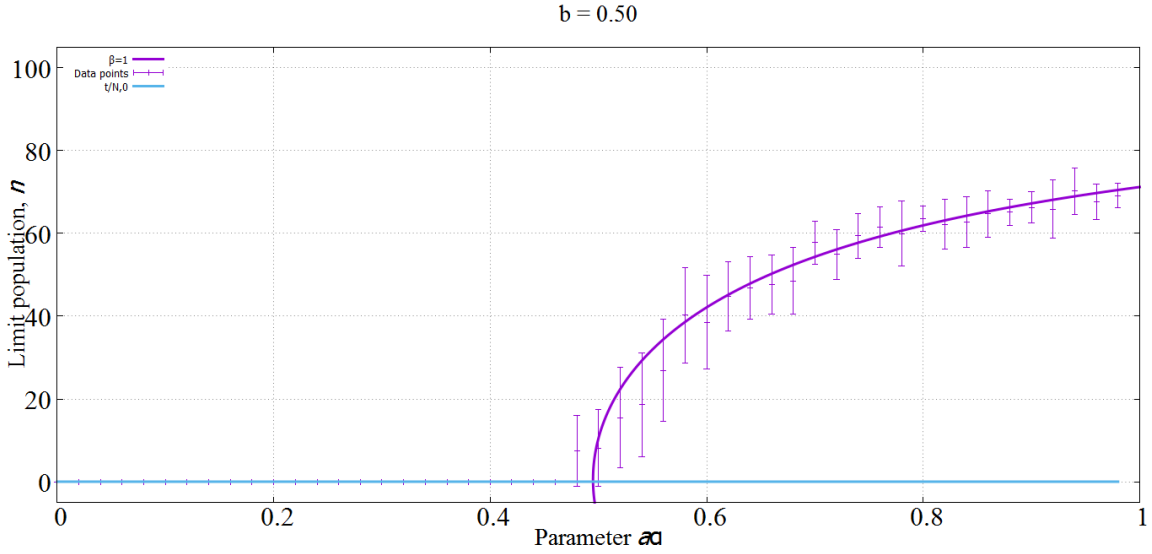

b = 0.10
Limit population, n
Parameter a
β=1
Data points
t/N,0
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

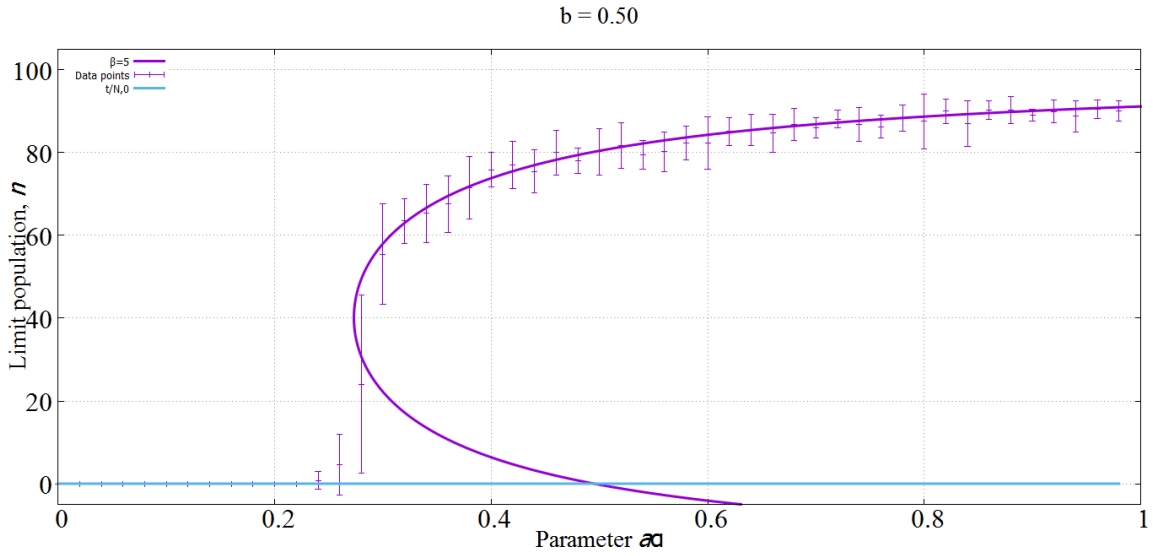

b = 0.10
Limit population, n
Parameter a
β=5
Data points
t/N,0
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

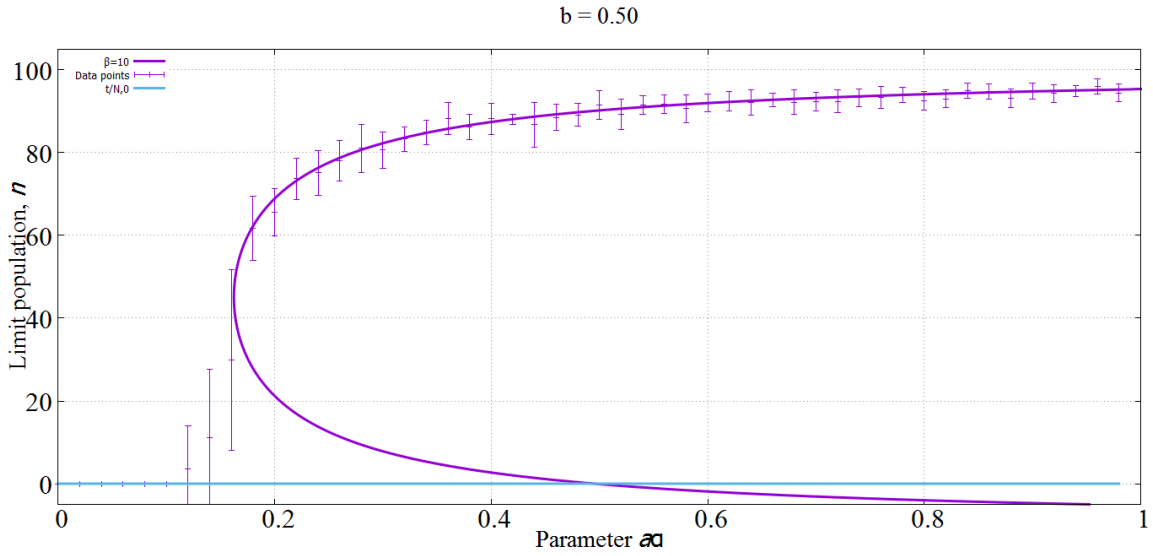

b = 0.10
Limit population, n
Parameter a
β=10
Data points
t/N,0
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

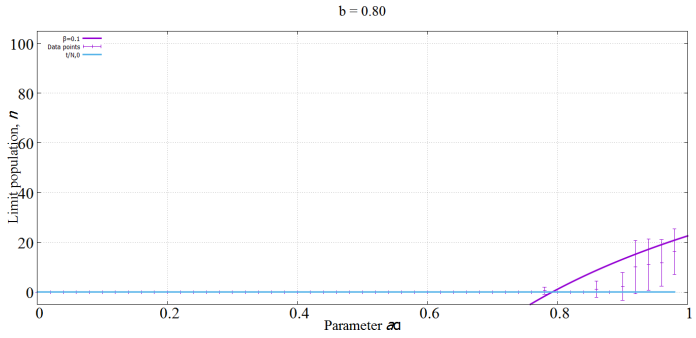

b = 0.50
Limit population, n
Parameter a
β=0.1
Data points
t/N,0
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

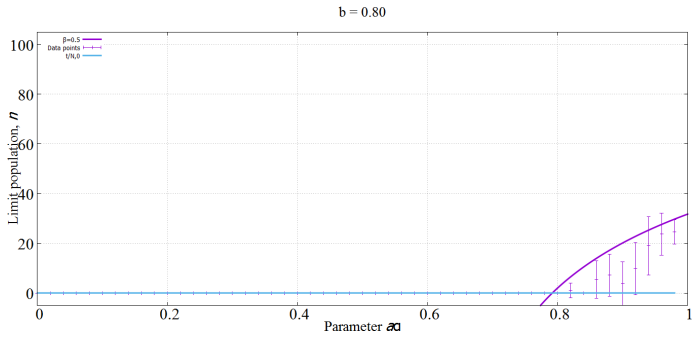

b = 0.50
β=0.5
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

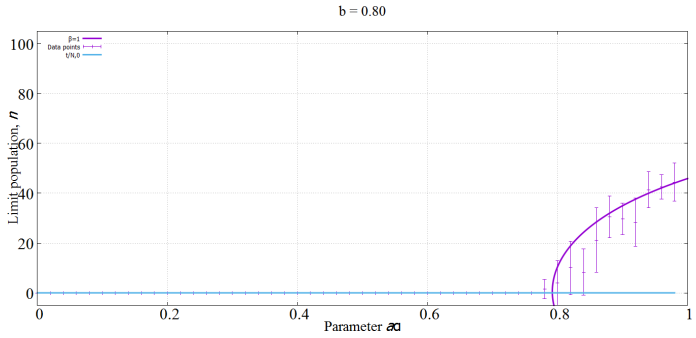

b = 0.50
β=1
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

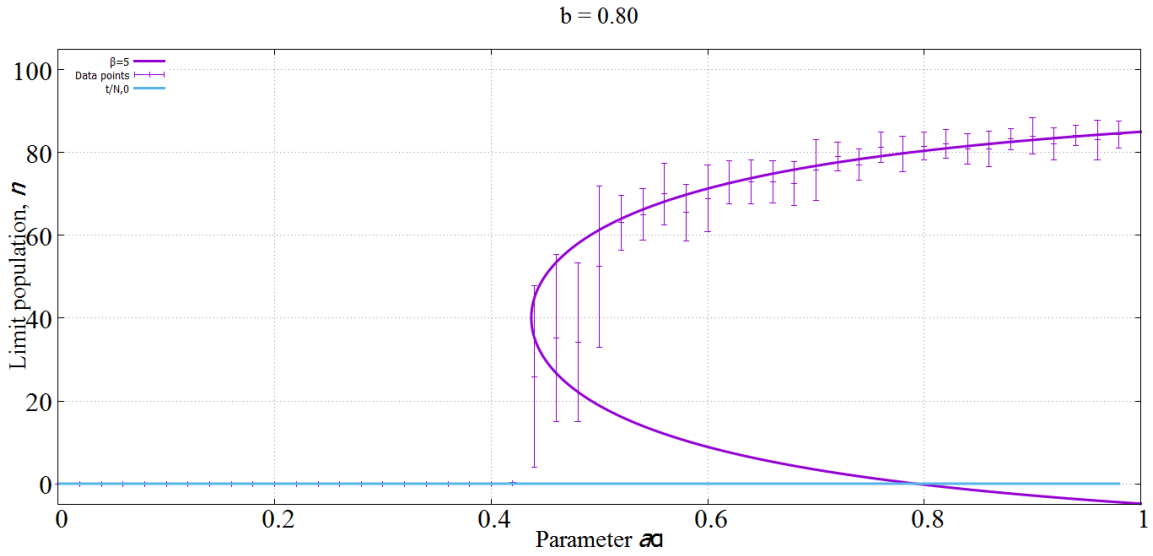

b = 0.50
β=5
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

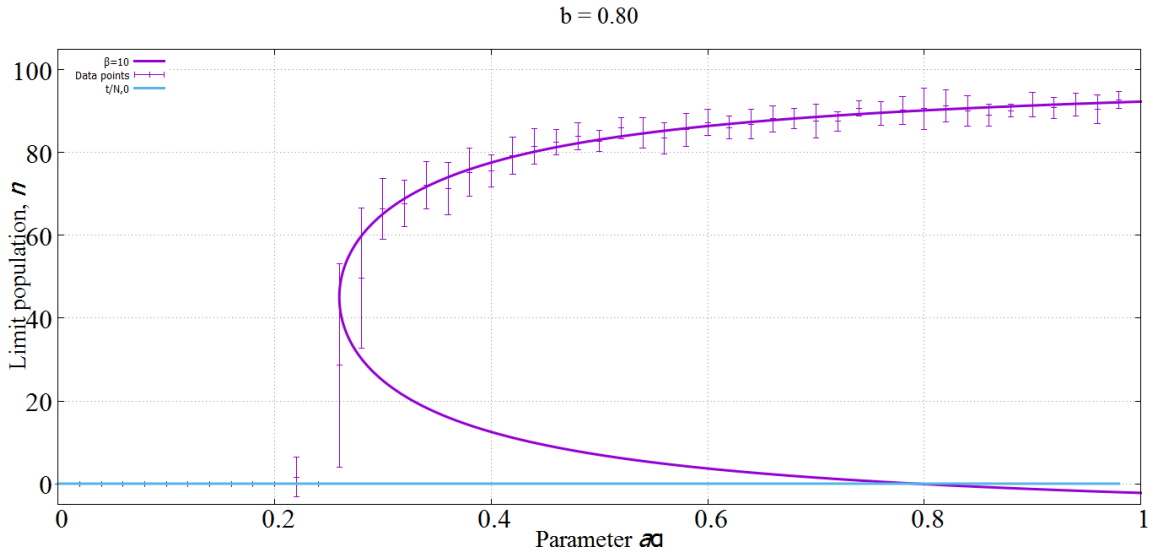

b = 0.50
β=10
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=0.1
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=0.5
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=1
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=5
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1


b = 0.80
β=10
Data points
t/N,0
Limit population, n
Parameter a
0
20
40
60
80
100
0
0.2
0.4
0.6
0.8
1

A.8 Vector Generator

Following is the code used to generate the vectors pointing to the steady-state given the initial to be plotted against the analytic steady-state solutions.

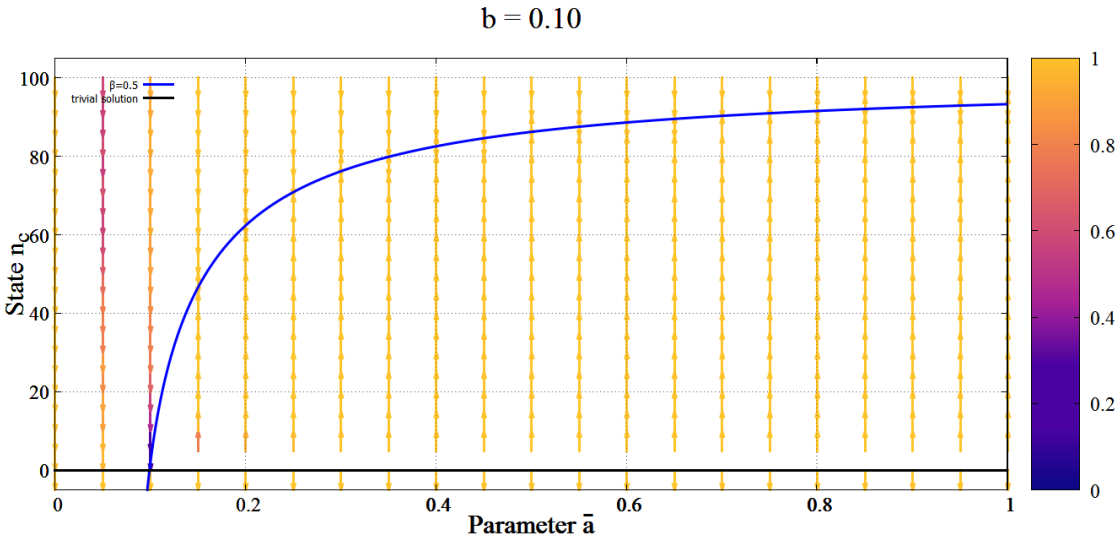
```
import applause_functions as app
import matplotlib.pyplot as plt
import numpy as np
import time as t

speed = t.clock()

N = 10
M = 10
startPop = 1
time = 100
t_1 = 0
aStoC = 1.0
bCtoS = 0.8
alpha = 1
beta = 10

x = 11
y = 11
z = 100

#a = app.app_sim(aStoC, bCtoS, alpha, beta, N, M, startPop,
    time, t_1)

def probFilter(a): #converts all non-zero values in a list to
    1
```

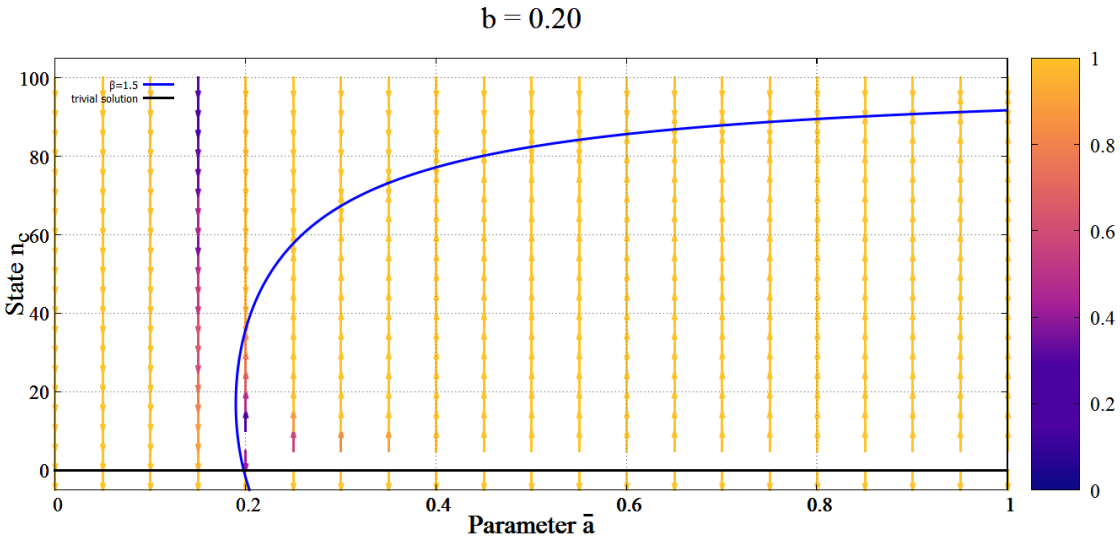
```
    for q in range(len(a)):
        if a[q] != 0:
            a[q] = 1

def lastList(abar, bCtoS, alpha, beta, N, M, start, time, t_1
    , iter):
    #runs app_sim 'iter' times and lists the last element per
        iteration
    finalVal = []
    probVal = []
    for i in range(iter):
        x = app.app_sim(abar, bCtoS, alpha, beta, N, M, start
            , time, t_1)[-1]
        finalVal.append(x)
        if x==0:
            probVal.append(0.0)
        else:
            probVal.append(1.0)
    return finalVal, probVal



def heatData(bCtoS, beta, x, y, z):
    abarRange = np.linspace(0,1,x)
    startRange = np.linspace(0,100,y)

    n = 0

    abar = np.zeros(len(abarRange)*len(startRange))
    start = np.zeros(len(abarRange)*len(startRange))
```

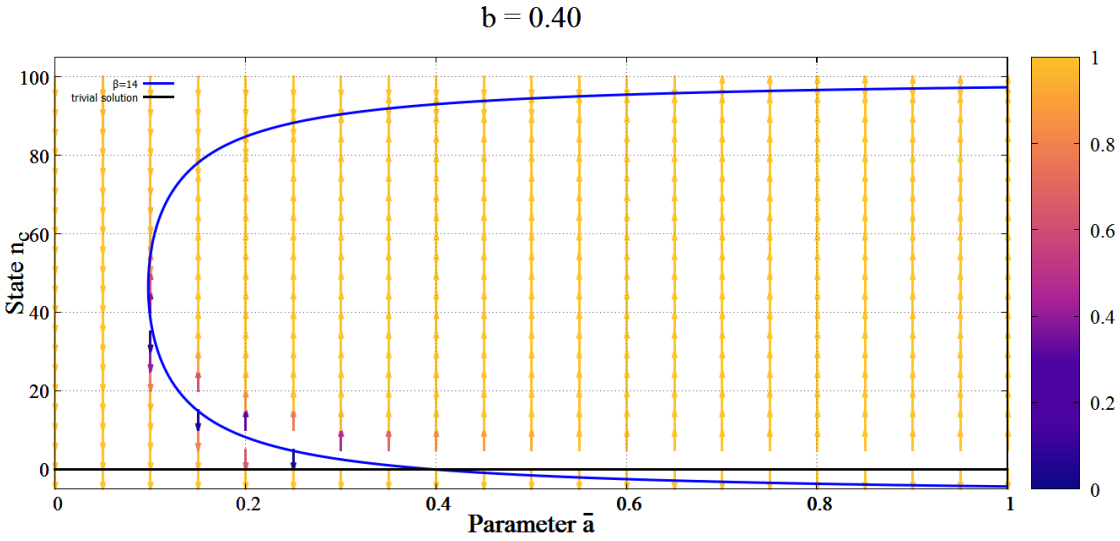
```
    rawHeat = np.zeros(len(abarRange)*len(startRange))

    for i in range(len(abarRange)):
        for j in range(len(startRange)):
            k = lastList(abarRange[i], bCtoS, alpha, beta, N,
                M, startRange[j], time, t_1, z)
            abar[n] = (abarRange[i])
            start[n] = (startRange[j])
            rawHeat[n] =(sum(k[1])/len(k[1]))
            n += 1
            print(t.clock() - speed)


    filename = 'b = ' + str(bCtoS) + ', beta = ' + str(beta)
      + ', x = ' + str(x) + ', y = ' + str(y) + ', z = ' +
      str(z) + '.txt'
    with open(filename,'w') as f:
        lis=[abar,start,rawHeat]
        for x in zip(*lis):
            f.write("{0}\t{1}\t{2}\n".format(*x))

    return abar,start,rawHeat
```

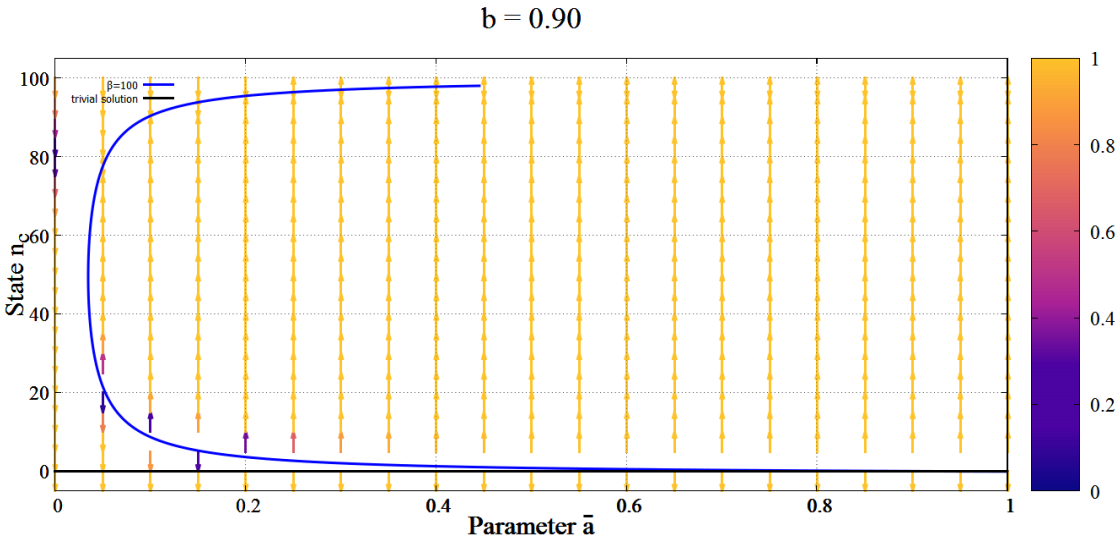
A.9 Vector Graphs

Shown are the phase space plots along with probabilistic vectors pointing towards the steady-state for various b and β values.






b = 0.40
β=14
trivial solution
State n_C
Parameter ā
0
20
40
60
80
100
0.2
0.4
0.6
0.8
1


b = 0.90
β=100
trivial solution
State n_C
Parameter ā
0
20
40
60
80
100
0.2
0.4
0.6
0.8
1

A.10 Simulations Spatial Effects

Shown is the code from the library that incorporates spatial effects to the simulations.

```
#spatial-dependent feedack, you can control the 'radius' of
    the reference agent
#taper refers to how many to add at the ends;radius = 0 taper
    = 1 is '90deg'
def feedback_space(alpha,system,system_row, system_column, N,
    M, radius, taper):
    applause_state = []
    for i in range(system_row):
        radius_mech = 0
        applause_state.append(system[i,system_column])
        while radius_mech != radius + taper*(system_row - i):
            radius_mech += 1
            if system_column + radius_mech < M:
                applause_state.append(system[i,system_column
                    + radius_mech])
            if system_column - radius_mech > -1:
                applause_state.append(system[i,system_column
                    - radius_mech])
    return alpha*nan_to_num(sum(applause_state)/len(
        applause_state))


def feedback_180deg(alpha,system,system_row, M): #reverted to
    simpler feedback space functions since runs took too long
    applause_state = [0]
    for i in range(system_row):
        applause_state.append(sum(system[i]))
```

```
    return alpha*nan_to_num(sum(applause_state)/(system_row*M
        ))

def feed_space(aStoC, bCtoS, alpha, beta, N, M, C, t, t_1,
    radius,taper):
    population = N * M
    AGENT = audience(N, M, C)
    graph = []
    zeroCount = 0

    for k in range(t):
        nC = sum(AGENT) #number of people clapping
        if nC == 0:
            zeroCount += 1
        graph.append(nC)
        for i in range(N):
            for j in range(M):
                if AGENT[i,j] == 0:
                    if random() <= aStoC * (1 - (1-force_func
                        (k, t_1)) * (1 - feedback_space(alpha,
                        AGENT,i, j, N, M, radius, taper))):
                            AGENT[i,j] += 1
                else:
                    if random() <= bCtoS * feedback_beta(beta
                        , nC, population):
                            AGENT[i,j] -= 1
        if zeroCount == 6:
            break
            return graph
```

```
    return graph

#sim with spatial dependence specific to 180 deg
def sim_space(aStoC, bCtoS, alpha, beta, N, M, C, t, t_1):
    population = N * M
    AGENT = audience(N, M, C)
    graph = []
    zeroCount = 0

    for k in range(t):
        nC = sum(AGENT) #number of people clapping
        if nC == 0:
            zeroCount += 1
        graph.append(nC)
        for i in range(N):
            for j in range(M):
                if AGENT[i,j] == 0:
                    if random() <= aStoC * (1 - (1-force_func
                        (k, t_1)) * (1 - feedback_180deg(alpha
                        ,AGENT,i, M))):
                            AGENT[i,j] += 1
                else:
                    if random() <= bCtoS * feedback_beta(beta
                        , nC, population):
                            AGENT[i,j] -= 1
        if zeroCount == 6:
            break
            return graph
    return graph
```

Shown is the code used to compare the different feedback functions.

```
import applause_functions as app
import matplotlib.pyplot as plt
import numpy as np
import time

N = 10
M = 10
population = N * M
t = 500
t_1 = 1
C = 0
aStoC = 0.7
bCtoS = 0.8
alpha = 0.5
beta = 4

start_time = time.time()

fig = plt.figure()
ax = fig.add_subplot(111)
sim1 = app.app_sim(aStoC, bCtoS, alpha, beta, N, M, C, t, t_1
    )
sim2 = app.feed_space(aStoC, bCtoS, alpha, beta, N, M, C, t,
    t_1,M,0)
sim3 = app.feed_space(aStoC, bCtoS, alpha, beta, N, M, C, t,
    t_1,0,1)
sim4 = app.feed_space(aStoC, bCtoS, alpha, beta, N, M, C, t,
    t_1,0,0)
```

```
plt.plot(sim1, color='b', label="FC")
plt.plot(sim2, color='r', label="180deg")
plt.plot(sim3, color='g', label="90deg")
plt.plot(sim4, color='y', label="0deg")
plt.legend(loc=4)

ax.text(1,5,'N = '+str(population), fontsize=15)

plt.title('$a = $'+str(aStoC)+' $b = $'+str(bCtoS)+r' $\alpha
    =$'+str(alpha)+r' $\beta=$'+str(beta),fontsize=20)
plt.xlabel('Time',fontsize=18)
plt.ylabel('State'+r' $n_{c}$',fontsize=18)
plt.xlim(0,30)
plt.ylim(0,110)
plt.savefig('feedback_sim7.png')
plt.show()
#print(sim2)
#print(str(time.time() - start_time) + ' seconds')
```

A.11 Investigating Population Dependence

Shown is the code used to investigate if the applause duration is dependent on population size.

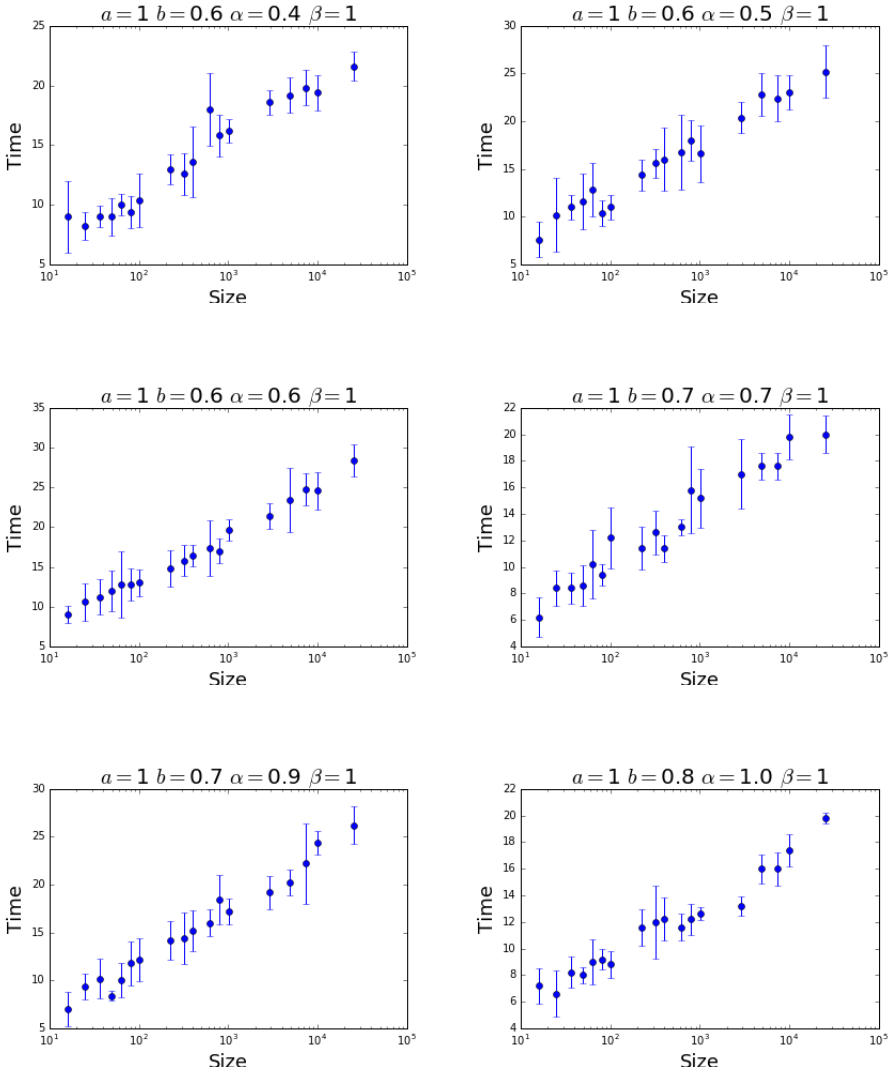
```
import applause_functions as app
import matplotlib.pyplot as plt
import numpy as np
import time


N = 10
M = 10
population = N * M
t = 60
t_1 = 2
C = 0
aStoC = 1
bCtoS = 0.8
alpha = 0.2
beta = 1


start_time = time.time()
trials = 5
popSet =
    [4,5,6,7,8,9,10,15,18,20,25,28,32,54,70,86,100,159]#,223,274,315]
     #set sqrt numbers that encompass the log scale


#used to take the index for which SIM becomes 0; gets the
    applause duration
def indexFilter(value,qlist,r1,r2):
     try:
```

```
        return qlist.index(value,r1,r2)
    except ValueError:
        return 0

for alpha_iter in range(8,11,1):
    for bCtoS_iter in range(1,11,1):

        #lists to be graphed
        duration = []
        durationSTD = []
        population = []

        for pop in popSet: #goes thru popSet
            appDurationTrials = [] #temporary holder of
               trials to be averaged
            for j in range(trials): #simulates parameters and
                population TRAIL times
                N = pop
                sim = app.sim_space(aStoC, bCtoS_iter*0.1,
                   alpha_iter*0.1, beta, N, N, C, t, t_1)
                if indexFilter(0,sim,1,t) != 0:
                    appDurationTrials.append(indexFilter(0,
                       sim,1,t))
                    print('Trial ' + str(j) + ' complete ' +
                        str(time.strftime("%Y-%m-%d %H:%M:%S
                       ")))
                else:
                    appDurationTrials.append(indexFilter(0,
                       sim,1,t))
```

```
                print('Trial ' + str(j) + ' reached non-
                    zero steady state ' + str(time.
                    strftime("%Y-%m-%d %H:%M:%S")))
                break


        duration.append(np.mean(appDurationTrials))
        durationSTD.append(np.std(appDurationTrials))
        population.append(N*N)
        print("Population " + str(pop) + " complete " +
            str(time.strftime("%Y-%m-%d %H:%M:%S")))
#scatter plot with error bars
#plt.subplot(111, xscale="log")
fig = plt.figure()
ax = fig.add_subplot(111,xscale="log")
x = population
y = duration
plt.errorbar(x, y, xerr=0, yerr=durationSTD, fmt='o')
plt.title('$a = $'+str(aStoC)+' $b = $'+str(round(
    bCtoS_iter*0.1,1))+r' $\alpha=$'+str(round(
    alpha_iter*0.1,1))+r' $\beta=$'+str(beta),fontsize
    =20)
plt.xlabel('Size',fontsize=18)
plt.ylabel('Time',fontsize=18)
plt.savefig('a = '+str(aStoC)+' b = '+str(round(
    bCtoS_iter*0.1,1))+r' alpha='+str(round(alpha_iter
    *0.1,1))+r' beta='+str(beta)+'.png')
plt.show()
print(str(time.time() - start_time) + ' seconds; ' +
    str(time.strftime("%Y-%m-%d %H:%M:%S")))
```

A.12 Best fit parameter sets

Shown are the graphs of applause duration versus population size of parameter sets that closely resemble the real-life data points.



Bibliography

[1] H. Peyton Young S.N. Durlauf. *Social Dynamics (Economic Learning and Social Evolution)*, volume 1. Academic Press, 2001.

[2] B.V. Krauth. Simulation-based estimation of peer effects. *Journal of Econometrics*, 133(0304-4076):243–271, 2006.

[3] J. Callaway. *Quantum theory of the solid state. Student edition*, volume 1 of *Quantum Theory of the Solid State. Student Edition*. Academic Press, 1976.

[4] A.K. Das G.A. Vugalter and V.A. Sorokin. Revivals in an infinite square well in the presence of a δ well. *Phys. Rev. A*, 66(012104):1–7, 2002.

[5] D.J. Griffiths. *Introduction to quantum mechanics*. Pearson Prentice Hall, second edition, 2005.

[6] W. Heitler and F. London. Wechselwirkung neutraler atome und homopolare bindung nach der quantenmechanik. *ZP*, 44:455–472, 1927.

[7] M.P. Marder. *Condensed matter physics*. Wiley-Interscience. John Wiley, 2000.

[8] R.W. Robinett. Quantum wave packet revivals. *Physics Reports*, 392:1–119, 2004.

[9] R. Schinke S.Yu. Grebenshchikov, C. Beck and S.C. Farantos. Three-dimensional molecular wave packets: Calculation of revival times from periodic orbits. *Physics Letters A*, (243):208–214, 1998.

[10] D.F. Styer. Quantum revivals versus classical periodicity in the infinite square well. *Am. J. Phys.*, 69:56, January 2001.

[11] T.K. Timberlake and S. Camp. Decay of wave packet revivals in the asymmetric infinite square well. *Am. J. Phys.*, 79:607–614, 2011.