Techniques for the Study of Biological Coupled Oscillator Systems

Nathan Breitsch

April 25, 2014

Contents

1 Introduction 3

2 Overview of Work on Local Dynamics 4
   2.1 Chemical Oscillations in Bio-reactors 4
   2.2 Clustering and Negative Feedback Models 7
   2.3 The Stability Problem for the Parameter Plane 8
   2.4 History of the Stability Problem for the Parameter Plane 9

3 Cell Cycle Dynamics: Clustering is Universal in Negative Feedback Systems 9
   3.1 Abstract 9
   3.2 introduction 11
   3.3 Previous results and notation 14

4 $F$ Near Cyclic Solutions 17
   4.1 Single Event Maps 17
   4.2 The Cyclic Solution and Partial Return Map $F$ 20
   4.3 Simultaneous Points 21
   4.4 Convexity of Domains 22

5 Regions of stability for Negative Feedback 25
   5.1 Computation of Event Triangles and Their Stability 25
6 Positive Feedback

7 Event-Driven Coupled Oscillator Systems
7.1 Introduction
7.2 Symbolic Dynamics
7.3 Matrix Representations
7.4 Architecture of a Program for Computing Global Dynamics
7.5 Computing the Global Poincaré map

8 Application to Signaling and Response Model
8.1 Signaling and Response as an Event Driven Systems
8.2 Cross Sections of Pieces of the Poincaré Map
8.3 Images of Poincaré Map Tilings
8.4 Fixed Points of the Poincaré Map

9 Conclusions and Open Problems
9.1 Summary of Accomplishments
9.2 Making Sense of High Dimensional Global Dynamics
9.3 Unproven Conjectures About Stability
9.4 Extent of Generalization

10 Python Code for Computation of Global Poincaré Map
10.1 Permutation
10.2 Word
10.3 AffineMap
10.4 ConvexSet
10.5 ConvexGeometry
10.6 System
10.7 cddSolver
10.8 SystemCatalog
10.9 Navigator
1 Introduction

An oscillator is a system which exhibits periodic behavior, that is, its state repeats itself on a regular interval of time. Examples include the earth’s position around the sun, the hands on a clock, the motion of a pendulum, and the beating of a heart.

Mathematically, an oscillator may be described at any given time by two numbers called phase and frequency. Phase, denoted by $X$, is a number from the interval $[0,1)$ which represents the progress of the oscillator through its current period. Consider the earth’s revolution around the sun: if we let $X = 0$ on January 1st, then $X \approx 1$ on December 31st and $X \approx 0.5$ on July 2nd. Note that the phase $X$ goes back to 0 after passing 1 just like the calendar goes back to January 1st after passing December 31st. We represent phase graphically as a point on the circle. 0 and 1 lie together at the top with phase increasing clockwise around the circle.

Frequency is a non-negative number which represents the number of periods completed by the oscillator in a unit of time. The frequency of the earth’s orbit is roughly 1 Period per 365 days. Frequency may be fixed e.g. the movement of a pendulum with fixed length, or vary with time e.g. heart rate.

Given the initial phase $X(0)$ and frequency $f(t)$ as a function of time, we may determine the phase at any time using the expression $X(T) = \int_0^T f(t)dt \mod 1$. This allows us to study an oscillator mathematically as a dynamical system on the circle. The phase $X(t)$ solves the differential equation $\frac{dX}{dt} = F(t)$ on $[0,1]$ and returns to zero upon reaching 1.

The dynamics of a single oscillator is boring, so we instead study populations of oscillators and allow them to influence each other’s frequency. A system of $N$ oscillators may be described mathematically by a phase vector $\vec{X}(t)$ and a frequency vector $\vec{F}(\vec{X})$. Here, the frequency of each oscillator depends only on the phases of the other oscillators. Such mathematical objects are called “systems of coupled oscillators” or more compactly, “coupled oscillator systems” and are the topic of this thesis.

More specifically, this thesis concerns coupled oscillator systems that arise from the cell cycle and mitosis. Mitosis is the process through which one cell divides to create two daughter cells. Those cells each give rise to two more cells which themselves undergo mitosis. The process must continue indefinitely to keep life on earth. We can visualize the process of repeated cell division with a family tree in which each cell has one parent and two children. If we examine a single path down this tree, we recover periodic
behavior: a cell is born, performs a somewhat predictable series of chemical reactions, undergoes mitosis at a somewhat predictable time and begins again as a new cell. In other words, we can think of a cell as an oscillator.

Mathematical work from this thesis is organized into two parts:

Part I contains an article about clustering in populations of yeast cells. Clustering is phenomenon in which cells organize into two or more synchronized groups. The article was accepted for publication by the Journal of Mathematical Biology.

Part II generalizes techniques from Part I for use in the study of global dynamics. It details the construction of a computer program which automates these techniques. The theory is developed in a general setting, abstracted away from the biological model of Part I. The computer program represents a large portion of the work involved in this thesis as well as its main accomplishment.

**Part I: Local Dynamics**

2 Overview of Work on Local Dynamics

This section contains “Cell Cycle Dynamics: Clustering is Universal in the Cell Cycle,” a joint work by myself, Professor Todd Young, Dr. Erik Boczko, and Gregory Moses. Before presenting the article, we will discuss its motivations, basic concepts, and history using language accessible to readers from disciplines outside mathematical biology.

2.1 Chemical Oscillations in Bio-reactors

In the introduction, we defined the phrase “coupled oscillator system” and proposed that the cell cycle can be thought of as an oscillator. Now we will discuss a setting in which cells influence each other’s development rate, i.e. a biological coupled oscillator system.

A bio-reactor is a machine which provides a controlled environment for micro-organisms. It is usually shaped like a cylinder and has a variety of control mechanisms including entry and exit for liquids and gasses, electrical sensors for chemical concentrations, mechanisms for temperature control, and an agitator which stirs the contents of the bio-reactor to maintain a homogeneous mixture (see figure 1).

For over forty years, scientists have observed oscillations in the concentration of dissolved oxygen
within yeast bio-reactors (Chen et al., 2007; Finn and Wilson, 1954; Kuenzi and Fiechter, 1969; Meyenburg, 1969; PR, 2003; Robertson et al., 2008; Von Meyenburg, 1973). The period of the oscillations divides the period of the cell cycle suggesting that they are driven by the cell cycle. Furthermore, the oscillations only appear when the yeast concentration is sufficiently dense suggesting that cell interaction is required.

To explain why these observations support such bold claims, let us discuss the phase distribution of a population of cells. If we were to conduct a bio-reactor census and obtain the age of every one of the bio-reactor’s roughly $10^{10}$ inhabitants, we could construct a histogram plotting the population of each age (phase) bracket. Such a plot would capture the distribution of the population among the set of possible phases. Mathematically, the phase distribution is the probability density function giving the phase of a randomly selected cell.

As time passes, the cells’ phases change at approximately the same rate. As a result, the phase distribution curves slide to the right. What happens to the part of the distribution which passes 1? In a bio-reactor, population is held constant by pumping fluid out of the tank and replacing it with water. So when a cell divides, only one daughter cell, on average, actually stays in the bio-reactor. For the purpose of modeling phase distribution, we may assume that cells don’t divide. Thus, the part of the distribution that passes 1 wraps around to replace the cells at phase 0. We could also visualize this by wrapping the phase distribution around the circle.
Figure 2: The green curve depicts a theoretical phase dependent oxygen consumption rate that depends on phase. The blue curves depict wavy and uniform phase distributions and the purple curves depict their resulting population oxygen consumption curves.

Consider two phase distributions: a wavy distribution and a uniform distribution (see figure 2). As time passes, the cells’ phases change at approximately the same rate. As a result, the phase distribution curves slide to the right. The uniform distribution does not change with time as translating the distribution results in the same uniform distribution. The wavy distribution, however, does change with time. Peaks and valleys slide to the right changing the average phase within the population.
Next, suppose oxygen consumption is phase dependent, that is, yeast cells consume oxygen at different rates depending on their phases. Let the oxygen consumption rate for a single cell be given by \( f(x) \). Then we compute the population oxygen consumption rate by summing over all cells: \( \sum_{i=1}^{N} f(x_i) \) where \( N \) is the population of the bio-reactor. In terms of \( \rho \), the phase distribution of the population, total oxygen consumption is given by \( N \int_{x=0}^{1} \rho(x,t)f(x)dx \). For the uniform distribution, \( \rho(X,t) \) is time invariant and therefore, so is the population oxygen consumption rate. No oxygen oscillations occur for uniform phase distributions. For wavy distributions, however, \( \rho(x,t) \) oscillates periodically and therefore, population oxygen consumption may oscillate periodically. With the link between phase distribution and oxygen oscillation established, we turn our attention to the emergence of non-uniform distributions.

### 2.2 Clustering and Negative Feedback Models

One of the most studied phase distributions is that of synchronization. Synchronization occurs when all oscillators have exactly the same phase. If all oscillators have the same frequency function, then synchronization is a periodic orbit; oscillators oscillate at the same frequency in a phase-locked configuration for all time.

Synchronization never persists in yeast bio-reactors. Scientists have attempted to grow synchronized yeast cultures in bio-reactors, but when cells are even slightly out of phase, the synchronized group falls apart. Instead of a single synchronized group, cells segregate into multiple synchronized groups called clusters. Clusters are distributed nearly evenly around the phase circle. If there are \( k \) clusters, then it takes \( 1/k \) times the period of the cell cycle for clusters to progress into each other’s starting phases. Therefore, the clustered phase distribution and resulting oxygen oscillations have period \( 1/k \) times that of the cell cycle. We conclude that clustering is consistent with experimental observations.

Boczko et al. proposed several mathematical models for cell phase interaction which produce clustering, but not synchronization. Their simplest model defines two adjacent intervals on the phase circle and calls them the signaling region and the response region. Outside the response region, cells have frequency 1. Inside the response region, cells have a frequency inversely dependent on the number of cells in the signaling region. We may write this model as a set of first order o.d.e.’s:

\[
\frac{dc_i}{dt} = \begin{cases} 
1, & \text{if } c_i \notin R \\
1 + f(I), & \text{if } c_i \in R
\end{cases} \quad i = 1, \ldots, n, \tag{1}
\]
where
\[ I(c) = \frac{\#\{i : c_i \in S\}}{n} \] (fraction of cells in the signaling region). \hspace{1cm} (2)

and \( f(I) \) is increasing or decreasing.

The model asserts that cells in the signaling region tell cells in the response region to slow down. The actual mechanism for signaling is not specified; it may be a deliberate signaling chemical or simply competition for resources. One biological motivation for studying models of clustering is to narrow down candidates for signaling mechanisms.

Numerical simulations suggest that the signaling and response model always results in clusters. Theory proves that for any positive integer \( k \), there exists a periodic orbit corresponding to a phase distribution of \( k \) clusters.

2.3 The Stability Problem for the Parameter Plane

Even though all clustered configurations are theoretically possible, not all withstand noise or emerge from uniform distributions. We are interested in clustering configurations that can emerge in nature. The stability of a periodic orbit is one indicator of whether the periodic orbit can occur in nature.

A periodic orbit may be classified as stable, neutrally stable, or unstable. If a periodic orbit is stable, it attracts nearby orbits and can handle some noise without “breaking.” Stable periodic orbits are likely to occur in nature. If a periodic orbit is unstable, it eventually repels nearby orbits and will not occur in nature. Neutrally stable periodic orbits may attract or repel nearby orbits. Neutrality offers no conclusions about whether a periodic orbit may occur in nature.

The k-cluster solution’s stability depends on the geometry of the signaling and response regions. The equation’s parameters \( r \) and \( s \) denote the beginning of the response region and the end of the signaling region respectively. Therefore, the stability of a k-cluster solution depends on the values of \( r \) and \( s \). It was once strongly believed within our research group that stability does not depend on the exact form of the function \( f \) in the equation, but rather, the events in which cells reach parameter values. This belief was reinforced by a previously proven result which only required \( f \) to be monotone. Since such a theorem was never proven, we work with \( f(I) = -I/k \) or \( f(I) = -(I/k)^2 \). All recorded stability results are the same for these two forms.

Once we specify a function \( f \), the stability problem for the parameter plane asks “for which parameter values \( r \) and \( s \) is the k cluster solution stable, unstable, or neutrally stable?” It turns out that we may solve
this problem by breaking the parameter plane into triangular tiles for which we can determine uniquely the k cluster solution’s stability. To visualize the solution to this problem, we can assign r and s to two perpendicular axes and color points red, white, and blue for unstable, neutral, and stable k-cluster solutions respectively. The solution for four clusters is shown in figure 3.

2.4 History of the Stability Problem for the Parameter Plane

Dr. Todd Young and Dr. Erik Boczko introduced several models in 2010 as possible explanations for oscillations in yeast. They called equations 1 and 2 the immediate model. With the help of Gregory Moses and others, they developed the theory of the immediate model in an article titled “Clustering in Cell Cycle Dynamics With General Responsive/Signaling Feedback.” This article introduces an existence proof for the k cluster solution. It also poses the stability problem for the parameter plane and solves the problem in a special case.

One of my first assignments as a member of Dr. Young’s mathematical biology research group was to explore the stability problem for parameter values not covered by the special case.

Just before my sophomore year, I developed and automated a method which finds the parameters r and s for which k-cyclic solution is stable, neutrally stable, or unstable. It took the most of my sophomore year to prove that the method was correct and document the method in a readable form. Application of the method resulted in more questions than answers. It produced evidence of two exiting conjectures, and introduced three new conjectures about the signaling and response model. One resulting conjecture was that for any value of r and s such that $0 < s < r < 1$, there exists a positive integer k for which the k-cluster solution is stable in cluster space. We therefore published the method and its results under the title “Cell Cycle Dynamics: Clustering is Universal in Negative Feedback Systems.”

3 Cell Cycle Dynamics: Clustering is Universal in Negative Feedback Systems

3.1 Abstract

We study a model of cell cycle ensemble dynamics with cell-cell feedback in which cells in one fixed phase of the cycle S (Signaling) produce chemical agents that affect the growth and development rate of cells that are in another phase R (Responsive). For this type of system there are special periodic solutions
Figure 3: Blue - Stable; Red - Unstable; White - Neutral. Parameter regions of stability for cyclic clustered solutions with $k = 4$. Since $r \geq s$, only the top half triangle is computed. A triangular tiling determines stability.
that we call \( k \)-cyclic or clustered. Biologically, a \( k \)-cyclic solution represents \( k \) cohorts of synchronized cells spaced nearly evenly around the cell cycle. We show, under very general nonlinear feedback, that for a fixed \( k \) the stability of the \( k \)-cyclic solutions can be characterized completely in parameter space, a 2 dimensional triangle \( T \). We show that \( T \) is naturally partitioned into \( k^2 \) sub-triangles on each of which the \( k \)-cyclic solutions all have the same stability type.

For negative feedback we observe that while the synchronous solution \( (k = 1) \) is unstable, regions of stability of \( k \geq 2 \) clustered solutions seem to occupy all of \( T \). We also observe bi-stability or multi-stability for many parameter values in negative feedback systems. Thus in systems with negative feedback we should expect to observe cyclic solutions for some \( k \). This is in contrast to the case of positive feedback, where we observe that the only asymptotically stable periodic orbit is the synchronous solution.

### 3.2 Introduction

We continue the analysis, initiated by Boczko et al. (2010), of dynamical models of the mitotic cell division cycle (CDC) for cultures of many cells in which cells in one fixed region of the cycle \( S \) (Signaling) produce chemical agents that affect the growth and development rate of cells in another fixed region \( R \) (Responsive). We consider a collection of a large number of cells, the position of the \( i \)-th cell is denoted \( c_i \in [0, 1) \equiv S^1 \) and its progression is governed by the equation:

\[
\frac{dc_i}{dt} = \begin{cases} 
1, & \text{if } c_i \not\in R \\
1 + f(I), & \text{if } c_i \in R,
\end{cases} \quad i = 1, \ldots, n, \tag{3}
\]

where

\[
I(c) = \frac{\# \{i : c_i \in S\}}{n} \quad \text{(fraction of cells in the signaling region).} \tag{4}
\]

The “response function” \( f(I) \) in (3) should satisfy \( f(0) = 0 \) and be monotone, but may be non-linear, and either positive or negative. When a cell reaches 1 (division) two cells appear 0 (birth). The number of cells \( n \) is not constant and in relevant applications may be on the order of \( 10^{10} \).

The system (3) is similar to “phase oscillator” equations that have been widely studied and used as models in a variety of contexts. The essential distinction of (3), (4) is in the form of the interaction between cells. In the phase oscillator models individual cells are coupled directly to each other in terms of the differences of their phases and those coupling forces must act additively, i.e. the total “force” on a cell is the sum of the “forces” due to all other individual cells. For the model (3), coupling is not additive.
by cell and is dependent on the cells’ absolute positions in the cycle, not phase differences.

Our main motivation for (3) is theoretical and experimental work on Yeast Metabolic Oscillations (YMO) – periodic oscillations of physiological variables that have been studied for over 40 years (Chen et al., 2007; Finn and Wilson, 1954; Kuenzi and Fiechter, 1969; Meyenburg, 1969; PR, 2003; Robertson et al., 2008; Von Meyenburg, 1973). There are clearly different types of YMO and these have been called by different names: glycolytic (Bier et al., 2000; Monte et al., 2007), autonomous (Keulers et al., 1996) or respiratory (Henson, 2005). All involve stable or quasi-stable periodic oscillations of metabolic variables, such as dissolved oxygen (DO₂). A correlation between YMO and the bud index (fraction of cells budded) was noted by Kuenzi and Fiechter (1969) and Meyenburg (1969), but a link between YMO and the CDC was obscured by the fact that the periods of YMO are shorter than the CDC times (in “fast glycolytic” oscillations the difference is especially large) and a relationship between YMO and the CDC seems to have been largely ignored. Recently, a correlation between YMO and CDC was noted in genetic expression data by Henson (2005), Klevecz and Murray (2001), and Tu et al. (2005), but not explained.

Figure 4: Phases of the mitotic cell cycle (CDC). The G1 phase begins following cell division. The beginning of the DNA synthesis phase, S, coincides with budding. G2 is a second “gap” phase. The M phase is characterized by narrowing or “necking” between the parent and daughter cell; it ends in cell division. The proposed R region is the later portion of G1 and the signaling region S is in the S phase. For convenient analysis, we shifted coordinates so that 0 lies between the two regions. A large sub-population of cells in the S phase may promote or inhibit progression of cells approaching the G1-S boundary.

Boczko et al. (2010) and Robertson et al. (2008) proposed cell cycle clustering as a link between YMO and the CDC. Figure 4 roughly illustrates the arrangement of phases in the cell cycle of yeast. We hypothesized that feedback effects on the CDC progression could cause groups of cells to segregate into CDC synchronized cohorts. Boczko et al. (2010) studied a few simple forms of (3) that included “delays”. We proposed that a group of cells in the critical S-phase might affect metabolism production and the metabolites may in turn inhibit or promote cell growth in the later part of the G1 phase, thus setting up a feedback mechanism in which YMO and CDC clustering are inextricably intertwined. We showed analytically and numerically that CDC feedback can robustly cause CDC clustering. By clustering we do not mean spatial clustering (cultures that exhibit YMO occur in well-mixed bioreactors), but temporal
clustering – groups of cells traversing the CDC in near synchrony. The idea of temporal clustering of cells has a long history (Rotenberg, 1977).

Guided by mathematical results, we verified the occurrence of clustering in two types of oscillating yeast using both bud index and cell density data (Stowers et al., 2011; Young et al., 2012). Some of the measurements from those experiments are shown in Figure 5. Experiments and analysis produced by Slavov and Botstein (2011) support this conclusion. We expanded our analysis of (3) to accommodate very general forms of feedback (Young et al., 2012).

The control of oscillation and regulation of yeast metabolism is important in the efficient management of bioprocesses (Kopmann et al., 1998; Kjeldsen et al., 2002; Uchiyama et al., 1996). YMO is of basic biological interest because it exposes questions regarding the coordination of the cell cycle and metabolism, and interconnectedness of various cellular and genetic processes (Boczko et al., 2005; Klevecz, 1976; Klevecz and Murray, 2001).

In this manuscript we study the stability properties of clustered periodic solutions of (3). A remarkable property of these systems is that the stability is not influenced by the form of the function $f$, but by the size
of the signaling and responsive regions $S$ and $R$. Our main conclusion is that for any negative feedback and any parameter values, there appears to be at least one integer $k \geq 2$ for which a solution consisting of $k$ cohorts is an attractor. On the other hand, for positive feedback we conclude that $k \geq 2$ clustered solutions are not asymptotically stable for any parameter values.

### 3.3 Previous results and notation

Let $R$ and $S$ be regions on the circle $([0, 1]$ where $1 \sim 0)$, with $R = [r, 1)$ and $S = [0, s)$. $R$ and $S$ are adjacent at $1 \sim 0$. We consider $r$ and $s$ as parameters and we assume $0 \leq s \leq r \leq 1$, i.e. the possible parameter values form a triangle.

In model (3), if two cells are initially synchronized in the cell cycle, they will remain so for all time. We will refer to subpopulations of synchronized cells as **cohorts**. For instance, (in the model) both of the cells resulting from a division will be in the same cohort.

In the clustered configuration, cells in the culture are grouped into $k$ cohorts of (approximately) equal total cell volume. We found that the cell density in the bioreactor oscillates periodically, entrained with the metabolic oscillations and in those experiments, all cohorts have approximately the same number of cells as they pass a specific milestone, such as budding (Boczko et al., 2010). In a steady state bioreactor, volume growth of cells approximately balances harvesting rate (See for instance Zhu et al. (2000)). Cohorts therefore maintain nearly constant cell volume while the population of each cohort is periodic. The number of cohorts $k$ remains constant.

Boczko et al. (2010), Young et al. (2012) and the results reported below make the modeling assumption that $n$ is fixed, but this is not essential in the present work. We justified this assumption on the basis of: the population of cohorts is periodic in the cycle and the volume of a cohort remains approximately constant. Also, under the modeling assumption that cells (3) are identical, tracking the progress of both cells resulting from a division is redundant.

We reformulate (3) so that cohorts take the place of cells. Denote the position of the $j$-th cohort at time $t$ by $x_j(t)$ for $j = 1, ..., k$. We will then study the dynamics of the cohorts; the equations governing the cohorts are similar to those governing individual cells,

\[
\frac{dx_j}{dt} = \begin{cases} 
1, & \text{if } x_j \notin R, \\
1 + f(I), & \text{if } x_j \in R,
\end{cases}
\]  

for $j = 1, ..., k$,  

(5)
where $I$ is the fraction of cohorts in $S$; $I = \# \{ j : x_j \in S \} / k$.

In this model, it is possible for a cohort to be situated in the cell cycle so that it does not interact with other cohorts, either by exerting feedback upon them or by having feedback exerted upon it during one entire transverse of a cell cycle. This occurs if the distance between a cohort and any other cohort is more than $|R| + |S|$. Define such a cohort to be isolated.

**Definition 3.1** Define $M \equiv \lfloor (|R| + |S|)^{-1} \rfloor$ where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to $x$. $M$ is the maximum number of isolated cohorts that can simultaneously exist.

If $1 \leq k \leq M$, then the initial condition: $x_j(0) = (j - 1)/k$ for all $1 \leq j \leq k$ trivially yields a periodic solution consisting of $k$ isolated cohorts. This periodic solution is neutrally stable and any initial condition in a neighborhood of it also yields a periodic solution in which the cohorts do not interact.

One type of isolated clustered solution consists of a single cohort, which we call synchronous. It is clear from (3) that a synchronous solution exists and is periodic with period 1. Young et al. (2012) showed for a more general class of feedback systems, including (3) that:

- For any positive feedback the synchronous solution is asymptotically stable.
- For any negative feedback the synchronous solution is unstable.

But these are not the only periodic solutions in (3). It is much more interesting to study solutions in which cohorts interact, i.e. $k > M$. Young et al. (2012) studied solutions of the form described in the following definition:

**Definition 3.2** Suppose that there exists a positive number $d$ such that $x_j(d) = x_{j+1}(0)$ for $j = 1, \ldots, k-1$ and $x_k(d) = x_1(0) \mod 1$. We call $\{x_j\}$ a $k$-cyclic solution. We call a solution uniform if it is $n$-cyclic, i.e. each cohort consists of a single cell.

A topological argument that a $k$-cyclic solution exists for any $k$ that is a divisor of $n$ (Young et al., 2012).

**Theorem 3.3 (Young et al. (2012))** For any monotone $f$ and any $0 \leq s \leq r \leq 1$ and any $k$, $k|n$, the system (3) possesses a $k$-cyclic solution. If $f$ is negative then the solution is unique.

For some cases, namely for $k = M + 1$, the stability of cyclic solutions is understood in the following.

**Theorem 3.4 (Young et al. (2012))** Denote $f(\frac{1}{k}) = \beta$. Under our assumptions, for any $0 < s < r < 1$, the cyclic solution consisting of $k = M + 1$ equal cohorts can be taken to have initial condition $x_1 = \frac{s}{k}$.
0, \(x_2 = d, x_3 = 2d, \ldots, x_k = kd\), for some \(d > 0\). If

\[ s < \frac{1}{k} \left( 1 + \frac{\beta r}{1 + \beta} \right) \quad \text{and} \quad r > \frac{k - 1}{k} (1 - s\beta) \]  

then the cyclic solution is unstable for positive \(\beta\) and stable (in the set of \(k\)-cohort solutions) for negative \(\beta\). Otherwise, the solution is neutrally stable. See Figure 6.

Since our assumptions imply that the cells are moving with positive speed, the set \(\{x_1 = 0\}\) is a Poincaré section for the flow, with a well-defined Poincaré map \(P\). It is also useful to define a map \(F : \{x_1 = 0\} \rightarrow \{x_k = 1\}\) as follows:

\[ F(x_1, x_2, \ldots, x_k) = (x_1(t^*), x_2(t^*), \ldots, x_k(t^*)), \]

where \(t^*\) is the time required for \(x_k(t)\) to reach 1. See Figure 7. Note that, with an appropriate reordering of indices, \(F^k\) is the Poincaré map. A cyclic solution is a fixed point of \(F\) and vice versa.

![Figure 7: An illustration of the map \(F\) with \(k = 3\).](image)

This thesis introduces new tools for studying the \(k\)-cyclic solutions. In particular, it solves the following question in general:

**Problem:** Fix \(k\). For what parameter values \((r,s)\) is a \(k\)-cyclic solution stable?
4 Near Cyclic Solutions

4.1 Single Event Maps

Calculation of $F$ hinges on the order of events – a cohort’s progress through the cell cycle can be described in terms of a sequence of events, such as the cohort entering $R$. Cohorts progress through the cell cycle at rates specified by the equation (5). These rates remain constant until a cohort reaches $s$, $r$, or 1 which we label as events 1, 2, and 3 respectively.

Figure 8: Event 1: a cohort reaches $s$. Event 2: a cohort reaches $r$. Event 3: a cohort reaches $1 \sim 0$.

We denote as $\sigma$ the number of cohorts in $S$, $\rho$ the number of cohorts not in $R$, i.e. $x_\sigma < s \leq x_{\sigma+1}$ and $x_\rho < r \leq x_{\rho+1}$. Thus at the beginning of a time interval on which the map $F$ is applied the initial positions of the $k$ cohorts will be:

$$0 = x_1 < \ldots < x_\sigma < s \leq x_{\sigma+1} < \ldots < x_\rho < r \leq x_{\rho+1} < \ldots < x_k < 1.$$ 

Given the current positions of the cohorts, we may calculate the time elapsed until the next event occurs. It is the minimum of

$$t_1 = s - x_\sigma, \quad t_2 = r - x_\rho, \quad t_3 = \frac{1 - x_k}{1 + f(I)}. \quad (7)$$

For each event, we define a corresponding function on a subset of the state space:

$$e_i(x) = e_i(x_1, x_2, \ldots x_k) = (x_1(t_1), x_2(t_2), \ldots, x_k(t_i)).$$

The domain of $e_i$ is restricted to those $x$ such that event $i$ is the first of the three events to occur (or concurrently as the first event). Because cohorts move at constant speeds between events, $e_i(x)$ can be easily calculated for any $x$. Define $\beta_\sigma = f(I)$. 


Each single event map is affine and has the form: $e_i(x_j) = x_j + rate_{ij} \times time$.

$$e_1(x_j) = x_j + \begin{cases} s - x_\sigma & \text{if } j \leq \rho \\ (1 + \beta_\sigma)(s - x_\sigma), & \text{if } j > \rho, \end{cases} \quad (8)$$

$$e_2(x_j) = x_j + \begin{cases} r - x_\rho & \text{if } j \leq \rho \\ (1 + \beta_\rho)(r - x_\rho), & \text{if } j > \rho, \end{cases} \quad (9)$$

$$e_3(x_j) = x_j + \begin{cases} \frac{1-x_k}{1+\beta_\sigma} & \text{if } j \leq \rho \\ (1-x_k), & \text{if } j > \rho. \end{cases} \quad (10)$$

When event 3 occurs, $x_k = 1 \sim 0$, and we define $e_4$ which reindexes the cohorts.

$$e_4(x_j) = \begin{cases} x_{j-1} & \text{if } j \neq 0 \\ x_k, & \text{if } j = 0. \end{cases} \quad (11)$$

The linear parts of these maps are easily calculated as below.

$$\left(\frac{\partial e_1}{\partial x}\right)_{i,j} = \begin{cases} 1 & \text{if } i = j \neq \sigma \\ -1 & \text{if } j = \sigma \text{ and } i < \sigma \\ -(1 + \beta_\sigma) & \text{if } j = \sigma \text{ and } i > \sigma - 1, \\ 0 & \text{otherwise}. \end{cases}$$

$$\left(\frac{\partial e_2}{\partial x}\right)_{i,j} = \begin{cases} 1 & \text{if } i = j \neq \rho \\ -1 & \text{if } j = \rho \text{ and } i < \rho \\ -(1 + \beta_\rho) & \text{if } j = \rho \text{ and } i > \rho, \\ 0 & \text{otherwise}. \end{cases}$$
\[
\left( \frac{\partial e_3}{\partial x} \right)_{i,j} = \begin{cases} 
1 & \text{if } i = j \neq k \\
-(1 + \beta) \sigma^{-1} & \text{if } j = \sigma \text{ and } i < \rho \\
-1 & \text{if } j = \sigma \text{ and } i > \rho, \\
0 & \text{otherwise.}
\end{cases}
\]

The relevant matrices \( De_1 \) and \( De_2 \) have similar structures. There are 1’s along the diagonal, except for one entry. Let \( j = \sigma \) for \( e_1 \) and \( j = \rho \) for \( e_2 \). The \( j \)-th column has \(-1\)'s down the first \( j - 1 \) rows, a 0 at the diagonal and \(-1 + \beta \sigma + 1\) in the rest of the rows.

\[
J_{1,2} = De_{1,2} = \begin{bmatrix} 
1 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 & -1 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & -1 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & -(1 + \beta) & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & -(1 + \beta) & 0 & \cdots & 1 
\end{bmatrix}. \tag{12}
\]

The matrix for \( e_3 \) has the following structure:

\[
J_3 = De_3 = \begin{bmatrix} 
1 & 0 & \cdots & 0 & 0 & 0 & \cdots & \frac{1}{1 + \beta \sigma} \\
0 & 1 & \cdots & 0 & 0 & 0 & \cdots & \frac{1}{1 + \beta \sigma} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & 0 & 0 & \cdots & \frac{1}{1 + \beta \sigma} \\
0 & 0 & \cdots & 0 & 1 & 0 & \cdots & \frac{1}{1 + \beta \sigma} \\
0 & 0 & \cdots & 0 & 0 & 1 & \cdots & -1 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 
\end{bmatrix}.
\]

In the final column, \(-1\) appears for rows \( \rho + 1 \leq j \leq k - 1 \).

All of these matrices are \( k \) by \( k \). The row corresponding to the cohort that will experience an event will consist entirely of 0’s, since the corresponding cohort is being moved to either \( s \), \( r \), or \( 1 \), regardless of where any of the cohorts lie.
4.2 The Cyclic Solution and Partial Return Map \( F \)

Given parameters \((r, s)\) and an initial condition \(x\), the partial return map \( F \) runs time until event 3 occurs. It is a composition of the maps \( e_1 \) and \( e_2 \), possibly repeated or empty, followed by \( e_3 \).

We have assumed above that all \( k \) cohorts have distinct positions. Clearly that we can extend the maps \( e_i \) continuously to the boundaries of this region \((x_i = x_{i+1})\). With this extension, the map \( F \) (along with reindexing \( e_4 \)) in a continuous mapping of the coordinate simplex:

\[
S = \{0 = x_1 \leq x_2 \leq \ldots \leq x_j \leq \ldots \leq x_k \leq 1\}
\]

into itself.

**Proposition 4.1** The sequence of events followed by the cyclic solution is either 1,2,3,... or 2,1,3,...

Proof: Since the cyclic solution is a fixed point of the map \( F \), the intervals \([0, s)\), \([s, r)\) and \([r, 1)\) must contain the same number of cohorts before and after an iteration of \( F \). Exactly one cohort leaves \([r, 1)\) and enters \([0, s)\) since event 3 occurs once. To balance, one cohort must leave \([0, s)\) and one cohort must enter \([r, 1)\). In other words, events 1 and 2 must occur exactly once before or simultaneously with event 3. \( \square \)

We wish to consider the map \( F \) nearby the cyclic solutions using a composition of single event maps. Since the form of \( F \) will depend upon the order of events, we first partition parameter space into regions for which \( k \) cyclic solutions have a fixed order of events.

**Definition 4.2** Fix \( k \). We call a subset \( \tau \subset T \) isosequential if the \( k \)-cyclic solutions corresponding to each parameter pair in the interior of \( \tau \) has the same \( \sigma \) and \( \rho \) and the same order of events.

For clarity, first consider a system with no feedback. In that case the \( k \)-cyclic solution is given by initial condition \( x_j = \frac{j-1}{k} \) for \( 1 \leq j \leq k \), irrespective of \( \sigma \) and \( \rho \). If we move \( r \) and \( s \) the order of events will change whenever \( r \) or \( s \) crosses the position of a cohort. This leads to the observation that the order of events partitions the parameter triangle precisely into a isosequential, regular sub-triangles. Figure 9 shows the \( T \) partitioned by event order for the cyclic solution with 3 cohorts and no feedback. We will call these isosequential sub-triangles event triangles.

We first partition parameter space using the values of \( \sigma \) and \( \rho \) in the cyclic solution. For \( 1 \leq i \leq j \leq k \), we may select all values of \( r \) and \( s \) such that \( \sigma = i \) and \( \rho = j \); The corresponding subset of parameter space is given by the inequalities \( \frac{i-1}{n} < s < \frac{i}{n} \) and \( \frac{j-1}{n} < r < \frac{j}{n} \). The two inequalities partition
Figure 9: Regions of parameter space and appropriate initial conditions for a $k = 3$ cyclic solution. For example, regions 7 and 8 both begin with $x_1 \in [s, r)$ and $x_2 \in R$. In region 7 cohort $x_1$ reaches $s$ before $x_1$ reaches $r$, while in region 8, $x_1$ reaches $r$ before $x_1$ reaches $s$. Boundaries between the regions correspond to simultaneous events.

parameter space into quadrilaterals except when $i = j$. In that case the region defined is a triangle with the line $r = s$ forming one of its sides. In figure 9, these correspond to triangles 1, 4, and 9.

Next, we refine the partition by dividing each quadrilateral into two parts depending on which event, 1 or 2, occurs first. If $(r - x_\rho) < (s - x_\sigma)$, then event 2 happens first. If $(r - x_\rho) > (s - x_\sigma)$, then event 1 happens first. Observe that the line $r = s + (x_\rho - x_\sigma)$ intercepts the corresponding quadrilateral at two of the vertices, and thus each quadrilateral is divided into a pair of triangles. Our partition now matches the one shown in figure 9. Within each triangle, $\rho, \sigma$, and event order are fixed.

For $k = 3$, we observe $3^2 = 9$ event triangles; this reflects the general fact that parameter space will be partitioned into $k^2$ triangles. There are $k + (k-1) + (k-2) + \ldots + 1$ ways to select integers $1 \leq \sigma \leq \rho \leq k$; of these ways $k$ (letting $\sigma = \rho$), result in triangles. The remaining $1 + 2 + \ldots + k$ ways yield quadrilaterals that will be divided into two triangles, and there are thus $2(1 + 2 + \ldots + (k-1)) + k = k^2$ triangles in total.

The event triangles constructed in this section (without feedback) are isosequential. We will show that isosequential regions remain triangular under non-zero feedback.

4.3 Simultaneous Points

Now we return to the model with non-zero feedback. By simultaneous points, we mean points in the $(r, s)$ plane for which the corresponding $k$-cyclic solution has events 1, 2, and 3 occurring simultaneously.
The vertices of each triangular region in figure 9 are simultaneous points. For a cyclic solution at a simultaneous point cohorts must initially lie directly on the milestones \( r \) and \( s \). See Figure 10.

![Figure 10: Initial condition of a cyclic solution with 26 cohorts corresponding to a simultaneous point. Here the feedback is negative causing cohorts in \( R \) to move more slowly and thus be more densely spaced.](image)

![Figure 11: Cyclic solutions corresponding to simultaneous points at the corners of “event triangles.”](image)

Fix integers \( \sigma \) and \( \rho \) with \( 1 \leq \sigma \leq \rho \leq k \). It is then easy to calculate the exact values of \( r \), \( s \), and \( x \) for a cyclic solution such \( x_{\sigma+1} = s \) and \( x_{\rho+1} = r \). As shown above, cohorts outside the responsive region are spaced by distance \( d \) and cohorts inside the responsive region by another distance \( d' \). These are the distances cohorts move during one iteration of the map \( F \). Since the distance around the circle is 1, we obtain the equation \( \rho d + (k - \rho)d' = 1 \). Because cohorts enter and leave the signaling region at the same time, feedback is always constant, \( f(I) = \beta_i \), so \( d' = (1 + \beta_\sigma)d \). These equations have a unique solution:

\[
    d = \frac{1}{\rho + (k - \rho)(1 + \beta_\sigma)}, \quad d' = \frac{1 + \beta_\sigma}{\rho + (k - \rho)(1 + \beta_\sigma)}.
\]

From \( d \) we can compute \( r \) and \( s \): \( r = \rho d \) and \( s = \sigma d \).

### 4.4 Convexity of Domains

**Proposition 4.3** Fix integers \( \sigma \) and \( \rho \) with \( 1 \leq \sigma \leq \rho \leq k \) and parameters \( r, s \). Then the domains of the maps \( e_1, e_2 \) and \( e_3 \) are convex subsets of the coordinate simplex \( S \).

Proof: First note that for fixed \( \sigma \), the value of \( \beta_\sigma = f(I) \) is constant and so it is constant throughout the domain of an event map. The domain of an event map is thus defined by linear constraints and is therefore...
Notice that the domain of a composite map is the affine pre-image of the domain of an event map. Convexity is preserved by affine pre-images, so the domain of any composite map is convex. Thus we have the following:

**Theorem 4.4** Fix integers $\sigma$ and $\rho$ with $1 \leq \sigma \leq \rho \leq k$ and parameters $r$, $s$. Then the map $F$ is a continuous, piece-wise affine map. On each subdomain on which it is affine it is equal to compositions of event maps $e_1$ and $e_2$ followed by $e_3$. These affine subdomains are convex.

For fixed $\sigma$ and $\rho$ write $E_{\sigma \rho}^i(r, s, x)$ for the map $(r, s, x) \mapsto (r, s, e_i(x))$ where $e_i(x)$ is applied with the parameters $r$ and $s$. We restrict the domains of these functions accordingly:

$$
\text{dom}(E_{\sigma \rho}^1) = \{(r, s, x) \in T_{\sigma \rho} : (s - x_\sigma) \leq (r - x_\rho) \text{ and } (s - x_\rho) \leq \frac{1 - x_k}{1 + \beta}\}
$$

$$
\text{dom}(E_{\sigma \rho}^2) = \{(r, s, x) \in T_{\sigma \rho} : (r - x_\rho) \leq (s - x_\sigma) \text{ and } (r - x_\rho) \leq \frac{1 - x_k}{1 + \beta}\},
$$

$$
\text{dom}(E_{\sigma \rho}^3) = \{(r, s, x) \in T_{\sigma \rho} : \frac{1 - x_k}{1 + \beta} \leq (s - x_\sigma) \text{ and } \frac{1 - x_k}{1 + \beta} \leq (r - x_\rho)\},
$$

where

$$
T_{\sigma \rho} = \{(r, s, x) : x_j < x_{j+1}, x_\rho < r < x_{\rho+1}, x_\sigma < s < x_{\sigma+1}\}.
$$

In the equations above, $T_{\sigma \rho}$ represents basic ordering assumptions.

**Proposition 4.5** For a fixed pair of integers $\sigma$, $\rho$, the domain of each $E_{\sigma \rho}^i$ is convex.

Proof: For a fixed $\sigma$, the domain is defined by a set of linear inequalities. □

**Definition 4.6** We define a generalized isosequential region as a maximal subset $I$ of $T \times S$ such that for any fixed $\sigma$ and $\rho$ and any $(r, s, x) \in I$ the map $F$ applied to $x$ is the same combination of event maps.

**Proposition 4.7** Let $(r_i, s_i, x_i)_{i=1}^m$ be a set of $m$ points in a generalized isosequential region such that each solution $x_i$ is cyclic. Any weighted average of these points is also cyclic solution which follows the same order of events.

Proof: Let $\bar{F}$ denote the map $(r, s, x) \mapsto (r, s, F(x))$ restricted to the isosequential region. Since each $x_i$ is cyclic $\bar{F}$ is a composition with events in the order $1, 2, 3$ or $2, 1, 3$. We argue that any weighted average $\sum_{i=1}^m \alpha_i (r_i, s_i, x_i)$ is also a fixed point in the domain of $\bar{F}$. We assume that $\alpha_i$ satisfies $\sum_{i=1}^m \alpha_i = 1$ and
\[(\forall i)(\alpha_i \geq 0).\] Since \(\text{dom}(\bar{F})\) is convex, \(\sum_{i=1}^{m} \alpha_i (r_i, s_i, x_i) \in \text{dom}(\bar{F})\). \(\bar{F}\) is affine, so it can be written in the form \(\bar{F} = (r, s, A x + b)\), where \(A\) is constant, depending only on the order of events. The vector \(b\) depends on \(r\) and \(s\), but can easily be seen from (8)-(10) to be linear in those terms, allowing us to write \(\bar{F} = (r, s, A x + k + ru + sv)\), where neither \(u\) nor \(v\) depend on \(r\), \(s\), or \(x\).

Now \((r, s, x)\) is a fixed point of \(F\) if and only if \(F(x) = x\) for parameter values \(r\) and \(s\). This condition can be rewritten as \((A - I)x = b\). Substituting \((R, S, Y) = \sum_{i=1}^{m} \alpha_i (r_i, s_i, x_i)\) yields

\[
(A - I)Y = (A - I) \sum_{i=1}^{m} \alpha_i x_j
\]

\[
= \sum_{i=1}^{m} \alpha_i (A - I) x_j
\]

\[
= \sum_{i=1}^{m} \alpha_i (-b_i)
\]

\[
= - \sum_{i=1}^{m} \alpha_i (k + r_i u + s_i v)
\]

\[
= -(k + \sum_{i=1}^{m} \alpha_i r_i) u + \sum_{i=1}^{m} \alpha_i s_i v
\]

\[
= -(k + R u + S v)
\]

\[
= -b.
\]

Thus any weighted average of fixed points is a fixed point also. \(\square\)

**Theorem 4.8** For any feedback function and given \(k\) each isosequential region is convex. The parameter triangle \(T\) is partitioned into \(k^2\) isosequential regions which are sub-triangles with simultaneous points at the corners. In the interior of each sub-triangle, all cyclic solutions have the same order of events.

Proof: By its definition an isosequential region is a set of parameters values in \(T\) such for which the fixed points of \(F\) have the same \(\sigma\) and \(\rho\) and the same order of event. It follows that it is the projection onto the parameter variables of a cross section of a generalized isosequential region consisting of the fixed points of \(\bar{F}\). By the above proposition the cross section is convex and thus its projection is convex.

If we fix \(1 \leq \sigma \leq \rho \leq k\) and the order in which events 1 and 2 occur, there are three simultaneous points \(S_1, S_2,\) and \(S_3\) on the boundary of the domain of the corresponding composite map. At \(S_1\) where \(x_\sigma = s, x_\rho = r\). At \(S_2\) where \(x_\sigma = s, x_{\rho+1} = r\) if event 1 happens first, or, where \(x_{\sigma+1} = s, x_\rho = r\) if event 2 happens first. Finally, at \(S_3, x_{\sigma+1} = s, x_{\rho+1} = r\). By the convexity of the domain of \(E_{1\sigma\rho}\) and \(E_{2\sigma\rho}\) in the parameter triangle, the sub-triangle spanned by these three points defines an order of events
for the fixed point associated with any pair \((r, s)\) is its interior. See figure 11.

Thus generally isosequential regions are triangles and we refer to them *event triangles*.

## 5 Regions of stability for Negative Feedback

### 5.1 Computation of Event Triangles and Their Stability

In this section we produce plots of event triangles of the parameter triangle for various values of the number of cohorts, \(k\), using the algorithm described in the previous sections and summarized in Appendix B.

In Figures 12, 14, 15, and 16 the parameter triangle is partitioned into event triangles for small negative feedback. For each triangle the linear part, \(DF\), of the map \(F\) is generated and its eigenvalues are calculated numerically using standard software. Each event triangle is then colored according to the linear stability of cyclic solutions associated with parameter values in its interior. If the eigenvalues of \(DF\) are all in the interior of the unit circle in \(\mathbb{C}\) then the cyclic solutions are stable and the triangle is colored blue. If at least one eigenvalues of \(DF\) is on the unit circle and the rest are inside the circle then the cyclic solutions are neutrally stable and the triangle is colored white. If at least one of the eigenvalues of \(DF\) has modulus greater than one then the \(k\)-cyclic solutions for those parameter values are unstable and the triangle is colored red.

From an inspection of the plots we quickly reach the following observations.

- Large numbers of cohorts are unstable for most parameters and regions of stability appear mostly on the edges of the parameter triangle, but,

- Any parameter pair seems to belong to the stable region of at least some \(k\) number of cohorts.

In Figure 12 we plot the regions of stability, neutrality and instability for the cyclic solution for \(k = 2\) to \(k = 9\). For \(k = 2\) there is a single triangle of parameter values in the triangle for which the 2 cyclic solution is stable, consistent with analysis by Young et al. (2012). For \(k = 3\) there are three triangles on which the 3 cyclic solution is stable. These calculations are rigorously verified in Appendix A. Note from the two plots for \(k = 2\) and \(k = 3\) we already observe regions of bi-stability. For these parameters there exist at least two different stable cyclic solutions and each will have some basin of attraction.

If we look at the union of the regions of stability for the first few \(k\), we find that these regions of stability rapidly cover most of the area of the parameter triangle. In Figure 13, we plot the union of the
Figure 12: Blue - Stable; Red - Unstable; White - Neutral. Parameter regions of stability for cyclic clustered solutions with $k = 2, 3, \ldots, 9$.

Figure 13: (a) An overlay of regions of stability for $k = 1, \ldots, 7$. Notice that the area not covered by the stability regions are only adjacent to the edges. We conjecture that every parameter pair in the interior of the triangle is eventually covered by a stability region for some $k$. (b) Numbers of cohorts realized in a cell cycle simulation using a mediated feedback. There is strong agreement between the two plots. In both of these plots the feedback function was $f(I) = -0.6I$
regions of stability for $k = 2, \ldots, 7$. We find that these regions account for 82\% of the area.

In Figure 15 we look at the stability for several prime values of $k$ and find simple patterns; the interior is all red and the edges are alternating white and blue.

In Figure 16 we plot the stability of each event triangles for several composite values of $k$. The patterns are more complex than for $k$ prime.

### 5.2 Neutrality on the Edges

We may compute the DF near the $k$ cyclic solution:

The matrix contains 1’s down the sub-diagonal except for the $\sigma$-th row $(1+w)$ and the $\rho$-th row $(\nu(x))$. The terms in the last columns change to $-1$’s at the $\rho$-th row. The three columns with terms other than 0 and 1 are the $\sigma$-th, the $\rho$’s, and the $n-1$-th. The $1 + \beta_{\sigma+1}$ term is in the $\rho$-th column and the $\rho$-st row.

**Proposition 5.1** For parameters in the triangles that share an edge with either $s = 0$ or $r = 1$, the $k$ cyclic solution is neutrally stable.

Proof: The regions in parameter space described in this theorem are where $x_\sigma$ leaves $S$ before $x_\rho$ enters $R$, and either $R$ is initially empty ($r \approx 1$) or $S$ initially contains only $x_1$. Consider first the
Figure 15: Blue - Stable; Red - Unstable; White - Neutral. Parameter regions of stability for cyclic clustered solutions for \( k \) prime, \( k = 17, 19, 23, 29, 31, 37 \). For all prime \( k \) we have investigated, the pattern is completely regular.

Figure 16: Blue - Stable; Red - Unstable; White - Neutral. Parameter regions of stability for cyclic clustered solutions for some \( k \) composite, \( k = 16, 18, 20, 21, 22, 24 \).

triangles with one side lying along the \( r \)-axis. For solutions lying in this region, \( S \) is so small that it contains only \( x_1 \); and the fact that they make up the upper halves of their parallelogram indicates that \( x_1 \)
leaves S before $x_ρ$ enters R. Thus cohorts not initially in R experience no feedback, while cohorts initially in R experience constant feedback until time $s$, whereafter they travel with speed 1. This is exactly the same behavior as one of the neutral cases in $k = M + 1$, described as Case II of Proposition 7.1 of Young et al. (2012), and it is a trivial exercise to verify that the linear part of the matrix representing F in this situation is the same as the one given in that paper.

Likewise, we can easily consider those neutral triangles lying along the s-axis; in those cases, R is so small that it is initially empty, and the order of events is given by (1,2,3); this again corresponds to one of the neutral cases in Proposition 7.1, described as case III, and it is again trivial to verify that the linear part of its matrix is the same as for the $k = M + 1$ case.

We observe in the diagram that we also have neutral regions along the diagonal. Again, the Jacobian matrix is easy to compute:

$$J = \begin{bmatrix}
0 & 0 & \cdots & 0 & w & 0 & \cdots & -\frac{1}{1+\beta_σ} \\
1 & 0 & \cdots & 0 & w & 0 & \cdots & -\frac{1}{1+\beta_σ} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & \cdots & 1 & w & 0 & \cdots & -\frac{1}{1+\beta_σ} \\
0 & 0 & \cdots & 0 & 1 + \beta_σ & 0 & \cdots & -1 \\
0 & 0 & \cdots & 0 & 0 & 1 & \cdots & -1 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & -1
\end{bmatrix}.$$  

It has 1’s lying along the sub-diagonal except for the $1 + \beta_σ$ in the $ρ$’th row; also at that row, the elements in the last column become −1.

5.3 Some Conjectures from the Plots

We have numerically investigated the stability regions of parameter space for all values of $k$ up to 100. Based on the results from these simulations, we offer the following conjectures.

**Conjecture 1:** Given any pair $(r,s)$ in the interior of the triangle $0 < s < r < 1$, there is some $k \geq 2$ such that the $k$ cyclic solution is stable for $(r,s)$.

This is based on Figure 13 and analogous plots for larger $k$.

**Conjecture 2:** For any $k$ that is prime, the stability diagram is perfectly regular. It alternates between stable and neutral on the edge. Triangles with an edge on the boundary are neutral and triangles with only a vertex on the edge are stable. All triangles that are not on the boundary are regions of instability.
This is based on the plots for all the prime \( k \) less than 100. We have observed no exceptions to this pattern.

**Conjecture 3:** If \( k \) is composite then the stability diagram is irregular. Triangles with edges on the boundary are neutral. Triangles with a vertex on the edge are either neutral or stable. If one numbers them starting from a corner of the triangle by an index \( i \), a triangle is stable if \( k \) and \( i \) are relatively prime and neutral otherwise.

We think that this is caused by a phenomenon that groups of cohorts may effectively decouple. For instance for \( k = 9 \) we conjecture that for the neutral regions in question, the cohorts decouple into 3 set of 3.

**Conjecture 4:** If \( k \) is composite, triangles in the interior of the simplex are mostly unstable, but some may be neutral along the lines \( r = 1/2 \), \( s = 1/2 \) and \( 1 - r + s = 1/2 \). As \( k \) grows large the area covered by neutral triangles goes to zero.

If Conjectures 2-4 are true, they would imply the following:

**Conjecture 5:** For \( k \) large, the \( k \) cyclic solution is unstable for all except a small set of parameter values with area \( O(1/n) \).

Conjecture 5 is relevant to the “uniform solution” where \( k = n = O(10^{10}) \). It would imply the uniform solution, which is the “steady-state” solution is unstable for practically all parameter values and any feedback.

### 6 Positive Feedback

The case of positive feedback is very different. Recall that the synchronous solution \( (k = 1) \) is asymptotically stable in the full space. For \( k = 2 \), we know that the 2-cyclic solution is never asymptotically stable, it is either repelling or part of a set of neutrally stable periodic orbits that are globally repelling (Young et al., 2012).

In Figure 17 we observe that \( k \)-cyclic solutions are never asymptotically stable for any \( k \geq 2 \). We have completed these calculations for \( k = 2 \) up to \( k = 100 \) and there are no regions of stability in parameter space for any of the cases. In fact all of the plots are exactly the same as those in Figure 17.

The sub-triangles that have an edge on the boundary are neutral while all other sub-triangles are unstable.

It would seem that for positive feedback the synchronized solution is the only asymptotically stable periodic orbit. If \( M = 1 \), i.e. for all \( r - s < 1/2 \) which covers 3/4 of the total area of the parameter
Figure 17: Red - Unstable; White - Neutral. Parameter regions of instability for $k$-cyclic clustered solutions with $k = 3, 4, 10, 15$. The picture is similar for any $k \geq 2$. There are no regions of stability for the clustered solutions with positive feedback.

triangle $T$ (see Figure 6), there can be no isolated periodic orbits. We conjecture:

**Conjecture 6:** For $r - s < 1/2$ the synchronous solution is the only stable invariant set.

If $M \geq 2$ that is, $r - s > 1/2$, then there are isolated $k$-cyclic solutions for every $2 \leq k \leq M$, but these periodic solutions are only neutrally stable. Young et al. (2012) showed that the set of such solutions union with the synchronous solution is an asymptotically stable set. The synchronous solution is the only attractor contained in it.

**Conjecture 7:** For $r - s > 1/2$ the set of $k$-cyclic solutions $1 \leq k \leq M$ is the only attracting set

The neutrally stable $k$-cyclic solutions in this case could appear as meta-stable states.

**Part II: Global Dynamics**

7 Event-Driven Coupled Oscillator Systems

The dynamics of the signaling and response model local to its $k$-cyclic solutions gives us a list of candidate phase distributions, yet it provides absolutely no insight into how or why these phase distributions emerge. Furthermore, local dynamics cannot rule out more complicated behavior overlooked by simulations and incomplete theory. Part II is the result of an attempt to discover how clustering emerges from incoherence and and to exhaust all possible behaviors of the signaling and response model. While part II achieves neither of these goals completely, it produces a promising set of techniques including the ability to construct a global Poincaré map.

Recall that for the signaling and response model, the Poincaré map is defined on the set $\{\vec{X} : X_0 = 0\}$. 
It runs time until $X_0$ returns to 0. In part I, we constructed the Poincaré map local to periodic orbits. In part II, we construct the Poincaré map everywhere. Since each cell passes 1 regularly for all time, the global Poincaré map captures the dynamics of signaling and response model.

The Poincaré map and each of its iterates are piece-wise affine. The pieces are convex. Convex pieces may be represented using matrix inequalities and affine maps may be represented as a matrix plus an offset vector. This means we can represent the Poincaré map and its iterates using a finite collection of matrices and vectors.

The theory of part II is formulated in a setting agnostic of the signaling and response model. It turns out that the only property needed to compute global dynamics is piece-wise constant frequency. To allow for convenient labeling of regions, we assert that the frequency depends only on the order of all oscillators and parameters (which we think of as oscillators of frequency 0). Frequency only changes when events (phase collisions of oscillators) occur. We therefore call such systems "event-driven coupled oscillator systems."

### 7.1 Introduction

Consider a model with $N$ coupled oscillators and the phase vector $X$. If no two oscillators have the same phase, then there exists a unique permutation $\sigma \in S_N$ such that $X_{\sigma^{-1}(0)} > X_{\sigma^{-1}(1)} > \ldots > X_{\sigma^{-1}(N-1)}$.

In this section, we study models with frequency functions that depend only on the permutation describing $\bar{X}$. Our formulation captures the properties that allowed study of the signaling/response systems. The signaling/response systems are easily represented in this context. To see this, represent the parameters $r$ and $s$ with oscillators of frequency zero. The frequency vector does not change unless cells cross $r$, `$
Definition 7.1 We call a system of coupled oscillators “event-driven” if and only if for any two phase vectors \( \vec{X} \) and \( \vec{Y} \), if there exists \( \sigma \) such that \( X_\sigma(0) > X_\sigma(1) > \ldots > X_\sigma(N-1) \) and \( Y_\sigma(0) > Y_\sigma(1) > \ldots > Y_\sigma(N-1) \), then \( f(\vec{X}) = f(\vec{Y}) \)

Each such system corresponds to a function \( \Phi : S_N \rightarrow [0, \infty) \) which assigns frequency vectors to permutations. Given \( \Phi \), the corresponding frequency function is given by:

\[
\vec{F}(\vec{X}) = \vec{\Phi}(\sigma \vec{X})
\]  

(14)

so the phase evolution is given by

\[
\frac{d\vec{X}}{dt} = \vec{\Phi}(\sigma \vec{X})
\]  

(15)

\( \vec{F} \) assigns a frequency vector to every point for which no two oscillators have the same phase. In order for equation 15 to induce a dynamical system, there must exist exactly one solution for every point in phase space defined for all time. This condition does not hold since \( \vec{F} \) is not defined everywhere. We can resolve this, however, when permutations change transversely. We shall assume that if \( X_i \) passes \( X_j \) at time \( t_0 \), then \( \Phi_i > \Phi_j \) for \( t \in (t_0 - \epsilon, t_0 + \epsilon) \).

7.2 Symbolic Dynamics

We study order-driven systems within the framework of symbolic dynamics. Given a sequence of permutations, we ask whether there exists an orbit which “follows” that sequence (we elaborate below). This approach is inspired by the “order of events” concept used to study the cell cycle feedback model.

Definition 7.2 The set (or domain) corresponding to a permutation, denoted \( dom(\sigma) \) is the set \( \{ \vec{X} \in [0,1]^N : 0 \leq X_{\sigma^{-1}(0)} \leq \ldots \leq X_{\sigma^{-1}(N-1)} \} \).

Observe that \( \{ dom(\sigma) : \sigma \in S_N \} \) covers the n-dimensional torus. For every trajectory \( \gamma(t) \), there exists a (not necessarily unique) sequence of permutations \( \{ \sigma_i \} \) and time steps \( t_i \) such that \( \gamma(t) \in dom(\sigma_i) \) for all \( t \in [t_i, t_{i+1}] \).

Let \( \sigma' = \sigma \circ (q,q+1) \) where \((q,q+1)\) is a transposition and \( q < n - 1 \). We introduce a function \( t_{\sigma,q} : dom(\sigma) \rightarrow \mathbb{R} \) given by

\[
t_{\sigma,q}(\vec{X}) = \frac{X_{\sigma^{-1}(q)} - X_{\sigma^{-1}(q+1)}}{\Phi_{\sigma^{-1}(q+1)} - \Phi_{\sigma^{-1}(q)}}
\]  

(16)
$t_{\sigma,q}$ is defined whenever $\Phi_{\sigma^{-1}(q)} \neq \Phi_{\sigma^{-1}(q+1)}$ and represents the time it would take $X_{\sigma^{-1}(q)}$ to pass $X_{\sigma^{-1}(q+1)}$ assuming the frequency vector remains constant. It is useful for describing the domains (sets in phase space) corresponding to sequences of permutations.

We must also consider the permutation that occurs when an oscillator reaches the end of its period. We denote by $R$ the permutation $(n-1, 0, 1, \ldots, n-2)$. Let $\sigma' = \sigma \circ R$. We define $t_{\sigma,n-1}$ by:

$$t_{\sigma,n-1}(\vec{X}) = 1 - \frac{X_{\sigma^{-1}(n-1)}}{\Phi_{\sigma^{-1}(n-1)}}$$

(17)

$t_{\sigma,n-1}(\vec{X})$ represents the time it would take $X_{\sigma^{-1}(n-1)}$ to reach 1, the end of the period, assuming the frequency vector remains constant.

Next, we introduce the map $T_{\sigma,q}$ which is given by

$$T_{\sigma,q}(\vec{X}) = \vec{X} + t_{\sigma,q}(\vec{X}) \ast \vec{\Phi}(\sigma)$$

(18)

$T_{\sigma,q}$ runs time for $t_{\sigma,q}(\vec{X})$ units assuming the frequency vector remains constant. From now on, we will use $P$ to denote a generic permutation of the form $(q, q+1)$ or $R$. Let $\sigma_0, \sigma_1, \ldots, \sigma_k$ be a sequence of permutations on $n$ symbols such that for all $0 \leq i < k-1$, $\sigma_{i+1} = \sigma_i \circ P_i$. The map corresponding to this sequence is given by:

$$T_{\sigma_0 \ldots \sigma_k} = \bigcirc_{i=0}^{k-1} T_{\sigma_i,q_i}$$

(19)

where $\bigcirc$ represents iterative function composition. $T_{\sigma_0 \ldots \sigma_k}$ maps points in $\text{dom}(\sigma_0)$ to their destinations in $\text{dom}(\sigma_k)$ after traveling through regions $\text{dom}(\sigma_0) \ldots \text{dom}(\sigma_{k-1})$. In order to conveniently write expressions in summation notation, we define the map corresponding to a sequence of length one to be the identity, i.e., $T_{\sigma_0}(\vec{X}) = \vec{X}$.

We can extend the time function $t$ to sequences of the form $\sigma_0, \sigma_1, \ldots, \sigma_k$ with $\sigma_{i+1} = \sigma_i \circ P_i$:

$$t_{\sigma_0 \ldots \sigma_k} = t_{\sigma_0,q_0}(\vec{X}) + t_{\sigma_1,q_1}(T_{\sigma_0,q_0}(\vec{X})) + \ldots + t_{\sigma_{k-1},q_{k-1}}(T_{\sigma_0 \ldots \sigma_{k-1}}(\vec{X})) = \sum_{i=0}^{k-1} t_{\sigma_i,q_i}(T_{\sigma_0 \ldots \sigma_i}(\vec{X}))$$

(20)

$t_{\sigma_0 \ldots \sigma_k}$ gives the time required for $\vec{X}(t)$ to traverse the given sequence. Since the expression above does not cover sequences of length 1, we define $t_{\sigma_0} = 0$. 

34
Next we define the set \( S_{\sigma,q} \) by

\[
\{ \vec{X} \in \text{dom}(\sigma) : (\forall p \in \{0, 1, \ldots, n-1\})(t_{\sigma,q} \leq t_{\sigma,p}) \}
\]  

(21)

\( S_{\sigma_0,q} \) denotes the subset of \( \text{dom}(\sigma) \) for which the event related to \( q \) occurs first in the dynamics of our system. We are now ready to define the domain of a sequence of permutations.

Let \( \sigma_0, \sigma_1, \ldots, \sigma_k \) be a sequence of permutations such that for all \( 0 \leq i < k-1, \sigma_{i+1} = \sigma_i \circ P_i \). The domain of the sequence is given by

\[
\text{dom}(\{\sigma_i\}_{i=0}^k) = S_{\sigma_0,q_0} \cap T_{\sigma_0,q_0}^{-1}(S_{\sigma_1,q_1}) \cap \ldots \cap T_{\sigma_0,\ldots,\sigma_{k-1}}^{-1}(S_{\sigma_{k-1},q_{k-1}})
\]  

(22)

\( \text{dom}(\{\sigma_i\}_{i=0}^{k-1}) \) gives the set of initial conditions whose trajectories traverse the given sequence.

### 7.3 Matrix Representations

In this subsection, we discuss matrix representation of functions and sets discussed above. Matrix representations provide insight into the structure of these concepts as well as a means to use them for computation. The set corresponding to a permutation, \( \text{dom}(\sigma) \), may be re-written in terms of a matrix inequality:

\[
\text{dom}(\sigma) = \{ \vec{X} : A_{\sigma} \vec{X} \leq \vec{b}_\sigma \}
\]  

(23)

where

\[
A_{\sigma} = \begin{bmatrix}
-1 & 0 & 0 & 0 & \ldots & 0 \\
1 & -1 & 0 & 0 & \ldots & 0 \\
0 & 1 & -1 & 0 & \ldots & 0 \\
0 & 0 & 1 & -1 & \ddots & \vdots \\
0 & 0 & 0 & 1 & \ddots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & -1 \\
0 & 0 & 0 & \ldots & 0 & 1
\end{bmatrix}
\]  

(24)
and

\[ \vec{b}_\sigma = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \]  

(25)

The map \( T_{\sigma,q} \) may be expressed as an affine map \( C_{\sigma,q} \vec{x} + \vec{d}_{\sigma,q} \) where \( C_{\sigma,q} \) is an \( n \times n \) matrix and \( \vec{d}_{\sigma,q} \) is a \( 1 \times n \) matrix.

\[
(C_{\sigma,q})_{i,j} = \begin{cases} 
\frac{\Phi_i(\sigma)}{\Phi_{\sigma^{-1}(q)}(\sigma) - \Phi_{\sigma^{-1}(q+1)}(\sigma)} [\delta_{i,\sigma^{-1}(q)} - \delta_{i,\sigma^{-1}(q+1)}] & \text{if } q < n - 1 \\
\frac{1}{\Phi_{\sigma^{-1}(n-1)}(\sigma)} \delta_{i,\sigma^{-1}(n-1)} & \text{if } q = n - 1
\end{cases}
\]  

(26)

\[
(d_{\sigma,q})_i = \begin{cases} 
0 & \text{if } q < n - 1 \\
\frac{1}{\Phi_{\sigma^{-1}(n)}(\sigma)} & \text{if } q = n - 1
\end{cases}
\]  

(27)

The function \( t_{\sigma,q} \) has the form \( e_{\sigma,q} \vec{x} + f_{\sigma,q} \) where \( e \) is a \( 1 \times n \) matrix and \( f \) is a constant.

\[
e_{\sigma,q} = \begin{cases} 
(e_{\sigma,q})_i = \frac{1}{\Phi_{\sigma^{-1}(q+1)}(\sigma) - \Phi_{\sigma^{-1}(q)}(\sigma)} [\delta_{i,\sigma^{-1}(q)} - \delta_{i,\sigma^{-1}(q+1)}] & \text{if } q < n - 1 \\
(e_{\sigma,q})_i = \frac{-1}{\Phi_{n-1}(\sigma)} \delta_{i,n-1} & \text{if } q = n - 1
\end{cases}
\]  

(28)

\[
f_{\sigma,q} = \begin{cases} 
f_{\sigma,q} = 0 & \text{if } q < n - 1 \\
f_{\sigma,q} = \frac{1}{\Phi_{\sigma^{-1}(n-1)}(\sigma)} & \text{if } q = n - 1
\end{cases}
\]  

(29)

We can write \( S_{\sigma_0,q} \) in terms of matrix inequalities by appending to \( A_\sigma \) and \( b_\sigma \) constraints that capture the event time condition: \( \{ \vec{X} \in \text{dom}(\sigma) : (\forall p \in \{0,1,\ldots,n-1\})(t_{\sigma,q} \leq t_{\sigma,p}) \} \). For each \( p \in \{0,1,\ldots,n-1\} \) we append the row \( e_{\sigma,q} - e_{\sigma,p} \) (defined in equation 28) to \( A_\sigma \) and append 0 to \( b_\sigma \). Thus, \( S_{\sigma,q} \) may be expressed as the matrix inequality \( A_{\sigma,q} \vec{x} \leq \vec{b}_{\sigma,q} \).

\[
A_{\sigma,q} = \begin{bmatrix} 
A_\sigma \\
t_{\sigma,q} - t_{\sigma,p_0} \\
\vdots \\
t_{\sigma,p} - t_{\sigma,p_m}
\end{bmatrix}
\]  

(30)
\[ \vec{b}_{\sigma,q} = \begin{bmatrix} b_{\sigma} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \] (31)

\( T_{\sigma_0...\sigma_k} \) is the composition of affine maps and is therefore affine; it has the form \( C_{\sigma_0...\sigma_k} \vec{x} + d_{\sigma_0...\sigma_k} \) where \( C_{\sigma_0...\sigma_k} \) is an \( n \times n \) matrix and \( d_{\sigma_0...\sigma_k} \) is an \( n \times 1 \) matrix.

\[ C_{\sigma_0...\sigma_k} = \prod_{i=0}^{k-1} C_{\sigma_i,q_i} \] (32)

\[ d_{\sigma_0...\sigma_k} = \sum_{i=0}^{k-1} C_{\sigma_0...\sigma_{k-i-1}} d_{\sigma_i,q_i} \] (33)

t_{\sigma_0...\sigma_k} has the form \( e_{\sigma_0...\sigma_k} \vec{x} + f_{\sigma_0...\sigma_k} \) where \( e \) is a \( 1 \times n \) matrix and \( f \) is a constant.

\[ e_{\sigma_0...\sigma_k} = \sum_{i=0}^{k-1} e_{\sigma_i,q_i} C_{\sigma_0...\sigma_i}(\vec{X}) \] (34)

\[ f_{\sigma_0...\sigma_k} = \sum_{i=0}^{k-1} f_{\sigma_i,q_i} \] (35)

dom\((\{\sigma_i\}_{i=0}^{k})\) may be written as a matrix inequality \( A_{\sigma_0...\sigma_k} \vec{X} \leq \vec{b}_{\sigma_0...\sigma_k} \) where \( A_{\sigma_0...\sigma_k} \) has \( n \) columns, and \( n(k+1) \) rows.

\[ A_{\sigma_0...\sigma_k} = \bigm\oplus_{i=0}^{k-1} A_{\sigma_i,q_i} C_{\sigma_i,q_i} \] (36)

\[ b_{\sigma_0...\sigma_k} = \bigm\oplus_{i=0}^{k-1} b_{\sigma_i,q_i} \] (37)

Where \( \bigm\oplus \) denotes vertical matrix concatenation. In practice, one may omit the rows corresponding to \( A_{\sigma_i} \) after \( \sigma_0 \) since those constraints are satisfied as an implication of previous constraints.

### 7.4 Architecture of a Program for Computing Global Dynamics

We construct an object oriented python program for automating the construction of domains and maps. The program consists of five instantiated classes: permutations, words, maps, sets, systems.

Permutations capture the order of a set of oscillators. We represent permutation as re-orderings of the integers 0, 1, 2, \( ...n-1 \). Permutations may be composed with each other, and may operate on the integers 0, 1, 2, \( ...n-1 \) via the image and its inverse. We utilize the image and inverse operations to compute
proper indices when constructing matrix representations for maps and sets.

A set object consists of a Matrix $A$ and a vector $b$ and represent a convex region defined by $Ax \leq b$. Sets may be intersected with each other and can compute cross sections. Maps consist of a matrix and a vector. Maps may be composed with each other, or may operate on sets via image and pre-image operations. Words are sequences of permutations. They also store, but do not compute maps and sets.

The system class is the logical workhorse of the program. It defines the details of oscillator feedback and computes the set and map representations stored by words.

The program also relies on four static classes to manipulate the instantiated classes. These classes are Navigator, ReportBuilder, SystemCatalog, and CddSolver. Navigator constructs word objects using system objects. SystemCatalog is a structure for storing different models. CddSolver determines whether sets are nonempty and converts sets of linear inequalities to sets of generating vertices.

7.5 Computing the Global Poincaré map

Suppose there are $m$ oscillators with strictly positive frequency. Every trajectory eventually has one of these oscillators reach 0. Without loss of generality, we can give this oscillator the index 0. The Poincaré map is defined on the set of states for which $X_0 = 0$ (this set is called the Poincaré section). It runs time until $X_0$ completes a full period and returns to zero. Since all orbits intersect with the Poincaré section regularly, the Poincaré map captures the dynamics of the coupled oscillator system.

The Poincaré map is piece-wise affine with convex pieces. The tools developed in this section allow us to construct the pieces and their corresponding affine maps. The pieces and maps are simply the domains and maps corresponding to a special set of permutation sequences for which $\sigma_0(0) = \sigma_1(0) = ... \sigma_j(0) = 0$ and $\sigma_k(0) = 0$ but $\sigma_i(0) \neq 0$ for $j < i < k$. Here, $k$ is the length of the sequence. The conditions indicate that $X_0$ at some point loses its status as the first oscillator but then returns to this position. If we select all such sequences, the word objects representing them will contain the sets and maps that make up the global Poincaré map.

8 Application to Signaling and Response Model

8.1 Signaling and Response as an Event Driven Systems

In this section, we apply methods for studying event driven systems to signaling and response models. In order to formulate a system corresponding to the model, we must index all oscillators and define
\( \phi : S_n \rightarrow R \). The parameters \( s \) and \( r \), which define the geometry of signaling and response systems, are represented by oscillators with frequency 0. Thus, a model with \( n \) cells corresponds to an event driven system with \( n + 2 \) oscillators. Let \( X_i \) in the event driven system represent \( x_i \) in the signaling-response model for \( 0 \leq i \leq n - 1 \). Let \( X_n = s \) and \( X_{n+1} = r \). Finally, let \( \sigma \) represent the order of the oscillators. That is, \( X_{\sigma^{-1}(0)} \leq X_{\sigma^{-1}(1)} \leq \ldots \leq X_{\sigma^{-1}(n+1)} \). Then the number of cells in the signaling region is given by

\[
I = \sigma(n)
\]

and \( X_i \) is in the response region if and only if

\[
\sigma(i) > \sigma(n + 1)
\]

This allows us to define \( \Phi \) for the signaling/response system:

\[
\Phi_i(\sigma) = \begin{cases} 
0 & \text{if } i = n \text{ or } i = n + 1 \\
1 & \text{if } \sigma(i) < \sigma(n + 1) \text{ and } i < n \\
1 - h(\sigma(n)) & \text{if } \sigma(i) > \sigma(n + 1) \text{ and } i < n
\end{cases}
\]

where \( h(k) \) is a monotonic function such that \( 0 \leq h(k) \leq 1 \) for \( 1 \leq k \leq n \). Equipped with \( \Phi \), we are ready to apply our new techniques to gain insight into the dynamics of our models.

### 8.2 Cross Sections of Pieces of the Poincaré Map

In order to visualize the geometry of the global Poincaré map for three cells, we take a two dimensional cross section by setting \( X_0 = 0 \) and fixing the parameters. Fixing three values collapses the five dimensional constraints into two dimensional constraints. We then use cddSolver to convert the constraint representations into vertex representations. Once we have the vertices for the two dimensional slices of each piece of the global 5 dimensional Poincaré map, we can plot these pieces to reveal the pieces of the global two dimensional Poincaré map.

The same process applies to maps which run time for a fixed number of events. We obtain the pieces of such maps by selecting words of a desired length rather than those corresponding to the Poincaré map. Figure 19 shows the pieces for the map corresponding to 1, 2, 3, 4, and 8-event maps for little or no feedback.
Figure 19: Cross section of pieces covering phase space for words of length 1, 2, 3, 4, and 8. $X_0 = 0$, $s = 0.3$, and $r = 0.7$. 
Notice that each picture in figure 19 is a refinement of the previous. We color boarders black if they are introduced by the 1-event map, red if they are introduced by the 2-event map, yellow for 3, and green for 4 or more. Figures 24, 25, and 26 depict the pieces of the eight-event map under various parameter values and feedback types. Figure 27 depicts the pieces of the Poincaré map without feedback. Notice that without feedback, the picture looks the same for the Poincaré map and eight words. The uncoupled system behaves like a simple oscillator so no new pieces appear after one period. For non-zero feedback, new pieces continue to appear long after the first period.

8.3 Images of Poincaré Map Tilings

We can visualize the action of the Poincaré map by looking at where its pieces go. Since the affine maps corresponding to each piece are not necessarily invertible, we must first use cddSolver to convert the constraint representations into vertex representations. We then apply the appropriate affine maps to each vertex to get the vertex representations for the images. Finally, we convert the vertex representations of the images back to a constraint representation.

Figures 28 through 37 show the pieces and pieces for the Poincaré map as well as their overlays for various parameter values and both positive and negative feedback. For negative feedback tiles are stretched towards the center of the triangle. This captures the movement of trajectories away from synchrony and towards a three cluster configuration. For positive feedback, the opposite occurs. All pieces stretch towards the corners indicating movement towards synchronization.

8.4 Fixed Points of the Poincaré Map

Let $F(x) = Ax + b$ be the affine map corresponding to a piece of the Poincaré map. We may find fixed points of the Poincaré map by appending the constraint $(A - I)x = -b$ to the set of inequalities defining the piece. We then compute the eigenvalues of $A$ to determine the stability of the fixed points. Once again, we visualize the results by taking a two dimensional cross section. In figures 20, 21, 22, and 23, we plot cross sections of the Poincaré pieces with fixed points for various parameter values. We include the parameter plane stability plots to the right. The red star indicates the parameter values for which these plots are generated.

Figure 20 depicts a neutrally stable set of fixed points in the middle of phase space with blue dots on the corner. The blue dots indicate that the region attracts orbits near the edge and therefore, 3-cluster behavior can be robust for $s = 0.1$ and $r = 0.9$ despite the lack of stability for the three cluster solution.
This is an example of knowledge that local analysis cannot provide.

Figure 22 depicts a stable three cluster solution as well as three neutrally stable bands connecting the uneven two cluster solution to the basin of attraction for the three cluster solution. Since the neutral bands are isolated and do not fall on the boundary between pieces of the Poincaré map, they attract from the sides. Existing theory did not predict these bands. This is an example of new behavior discovered by computing the global Poincaré map.

Figures 38 through 47 show closeups of these images for positive as well as negative feedback.
Figure 23

Figure 24: Cross sections of pieces covering phase space for words of length 8 with no feedback. \(X_0 = 0, s = 0.1, 0.2, 0.3, 0.4, \) and 0.5. \(r = 0.9, 0.8, 0.7, 0.6, \) and 0.5
Figure 25: Cross sections of pieces covering phase space for words of length 8 with negative feedback. $X_0 = 0$, $s = 0.1, 0.2, 0.3, 0.4$, and 0.5. $r = 0.9, 0.8, 0.7, 0.6$, and 0.5
Figure 26: Cross sections of pieces covering phase space for words of length 8 with positive feedback. $X_0 = 0, s = 0.1, 0.2, 0.3, 0.4, \text{and } 0.5, r = 0.9, 0.8, 0.7, 0.6, \text{and } 0.5$
Figure 27: Cross sections of pieces covering phase space for words corresponding to Poincaré map with no feedback. $X_0 = 0, s = 0.1, 0.2, 0.3, 0.4, \text{ and } 0.5$. $r = 0.9, 0.8, 0.7, 0.6, \text{ and } 0.5$ Notice that nothing is changed from the eight word tiling. Without feedback, the entire system is a simple oscillator. No new pieces appear after one period.
Figure 28: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.1, r = 0.9$ feedback is negative.
Figure 29: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.2$, $r = 0.8$ feedback is negative.
Figure 30: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.3, r = 0.7$ feedback is negative.
Figure 31: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.4$, $r = 0.6$ feedback is negative.
Figure 32: (a) - Cross Section, (b) - Image, (c) Overlay. \( s = 0.5, r = 0.5 \) feedback is negative.
Figure 33: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.1, r = 0.9$ feedback is positive.
Figure 34: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.2$, $r = 0.8$ feedback is positive.
Figure 35: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.3, r = 0.7$ feedback is positive.
Figure 36: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.4$, $r = 0.6$ feedback is positive.
Figure 37: (a) - Cross Section, (b) - Image, (c) Overlay. $s = 0.5, r = 0.5$ feedback is positive.
Figure 38: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.1$, $r = 0.9$, and feedback is negative.
Figure 39: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.2$, $r = 0.8$, and feedback is negative.
Figure 40: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.3$, $r = 0.7$, and feedback is negative.
Figure 41: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.4$, $r = 0.6$, and feedback is negative.
Figure 42: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.5$, $r = 0.5$, and feedback is negative.
Figure 43: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.1$, $r = 0.9$, and feedback is positive.
Figure 44: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. \( s = 0.2, r = 0.8 \), and feedback is positive.
Figure 45: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.3$, $r = 0.7$, and feedback is positive.
Figure 46: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.4$, $r = 0.6$, and feedback is positive.
Figure 47: Fixed points of the Poincaré map. Blue points are stable, red points are unstable, and green points are neutrally stable. $s = 0.5$, $r = 0.5$, and feedback is positive.
9 Conclusions and Open Problems

9.1 Summary of Accomplishments

In part I, we developed and automated a method for computing the Poincaré map near k-cyclic solutions in the signaling and response model. This allowed us to classify the stability of k-cyclic solutions given parameters $r$ and $s$ as well as the function $f$. We used the method to construct stability plots which graphically depict where the k-cyclic solution is stable. The stability plots resulted in mysterious new conjectures and a publication in the Journal of Mathematical Biology.

In Part II, we developed and automated a method for computing the Poincaré map globally for a very general class of systems. We used the method to produce a global dynamical picture of the three cluster signaling and response system including all fixed points of the Poincaré map. The pictures uncovered previously unstudied periodic orbits as well as an example of robust behavior that does not come in the form of a stable periodic orbit.

9.2 Making Sense of High Dimensional Global Dynamics

We can construct the global Poincaré map for any number of oscillators, yet we can only draw two dimensional pictures. The first open problem asks how to make sense of higher dimensional Poincaré maps.

One possible strategy is to investigate more clever two dimensional cross sections of higher dimensional partitions. The code presented in the appendix allows the computation of cross sections by fixing values for all but two of the oscillators. It is conceivable that we could take arbitrary linear cross sections as well as certain non-linear cross sections.

One could also focus on the periodic orbits and their connecting heteroclinic orbits. It is simple to compute the stable and unstable subspaces of the linear region around a periodic orbit. One could then compute the images of these subspaces through iterations of the Poincaré map until the image of an unstable subspace intersects with a stable subspace. The connectivity of periodic orbits could then be represented with a directed graph and embedded into two dimensions, or embedded into three dimensions and then projected onto two dimensions. Such a visualization could help to understand the transient behavior between unstable and stable configurations, i.e. how uniform and synchronized phase distributions re-organize into clustered distributions.
9.3 Unproven Conjectures About Stability

The second open problem asks how to reason about stability for arbitrary values of $k$. The stability plots of Part I suggest a beautiful relationship between stability and the relative primality of $\sigma$, $\rho$, and $\sigma - \rho$ with $k$. It also reveals a pattern for prime $k$ and dominating instability for large $k$. Currently, none of these conjectures have been proven.

The only proven result for general $k$ is the stability of the M+1 cyclic solution where M is the maximum number of clusters that never interact. This result can be proven because the matrix part of the F map has an easy to compute and easy to solve characteristic equation.

We have attempted without success to extract information from the general form of the F map matrix. We have also studied maps with different Poincaré sections or no sections at all. Most recently, we attempted to find a circulant matrix which captures the stability of k-cyclic solutions. A circulant matrix is a matrix whose columns are successive rotations of its first column. The general form can be written as:

$$
\begin{bmatrix}
c_0 & c_1 & \cdots & c_n \\
c_1 & c_2 & \cdots & c_0 \\
\vdots & \vdots & \ddots & \vdots \\
c_n & c_0 & \cdots & c_{n-1}
\end{bmatrix}
$$

(41)

Circulant matrices have eigenvalues of the form

$$
\lambda_j = \sum_{i=0}^{n-1} c_i \omega_j^{n-i}
$$

(42)

where $\omega_j$ is the primitive $j$'th root of unity. We believe that a circulant matrix can capture the stability of cyclic solutions because the long term interaction between two cells should depend only on their relative position. Intuitively, it makes sense that the interaction matrix representing how much cell i slows down cell j should have circulant form. We are excited about the form of a circulant matrix’s eigenvalues because it captures a key aspect of primality. If $j$ and $k$ are relatively prime, then $\omega_j^i$ hits all roots of unity for $i$ from 0 to $n - 1$. Otherwise, it only hits a proper subset. We believe this property could be exploited to prove conjectures relating primality and relative primality to stability.
9.4 Extent of Generalization

The third open problem is to explore how other systems can be modeled using event-driven coupled oscillator systems. Systems related to the cell cycle are the best candidates since cellular processes often occur during well defined intervals. Of course, just about any coupled oscillator system can be approximated by an event-driven one. Simply place parameters (oscillators of frequency 0) densely around the phase circle and then assign the proper frequency to each permutation of oscillators. The result is a grid approximation of the function $F$. Application of techniques to fine approximations of complicated functions will probably not be computationally feasible. Rough approximations, however, may be successful in reproducing important aspects of the dynamics of a system. It is exciting to know that for any event-driven system, we can compute a global Poincaré and classify its fixed points. Whether this will help study models other than signaling and response remains to be seen.

10 Python Code for Computation of Global Poincaré Map.

10.1 Permutation

```python
from copy import deepcopy
from AffineMap import AffineMap

# represents order of oscillators
# provides tools for reasoning about indices
class Permutation:
    def __init__(self, perm):
        self.permutation = perm

    # output: integer (number oscillators represented)
    def dim(self):
        return len(self.permutation)

    # input: integer (order index)
    # output: integer (innate index)
    def sigmaInv(self, index):
        return self.permutation[index]

    # input: integer (innate index)
    # output: integer (order index)
    def sigma(self, index):
        for i in range(0, self.dim()):
            if self.permutation[i] == index:
                return i
        return -1
```

# output: Permutation (inverse permutation)
```python
def inverse(self):
    temp = []
    for i in range(0, self.dim()):
        temp.append(self.sigmaInv(i))
    return Permutation(temp)

#input: integer (number of oscillators)
#output: Permutation (identity permutation with given dim)
@staticmethod
def identity(dim):
    temp = []
    for i in range(0, dim):
        temp.append(i)
    return Permutation(temp)

#input: Permutation^2 (permutations to compose)
#output: Permutation (composition of inputs)
@staticmethod
def compose(perm1, perm2):
    p1 = perm1.permutation
    p2 = perm2.permutation
    temp = []
    for i in range(0, len(p1)):
        temp.append(p1[p2[i]])
    return Permutation(temp)

#input: "R" or list of two integers (transposition representation)
#output: permutation resulting from transposition
def transpose(self, transposition):
    newPerm = deepcopy(self.permutation)
    if transposition == "R":  
        temp = newPerm[len(newPerm)-1]
        for i in range(1, len(newPerm)):  
            newPerm[len(newPerm) - i] = newPerm[len(newPerm)-i-1]
        newPerm[0] = temp
    else:
        a = transposition[0]
        b = transposition[1]
        temp = newPerm[a]
        newPerm[a] = newPerm[b]
        newPerm[b] = temp
    return Permutation(newPerm)

#output: AffineMap (return AffineMap which induces permutation on i.c.)
def mapRepresentation(self):
    b = Permutation.makeZeros(self.dim())
    A = []
    for i in range(0, self.dim()):
```

---

Additional content not shown in the image.
nextRow = []
for j in range(0, self.dim()):
    if j == self.sigma(i):
        nextRow.append(1)
    else:
        nextRow.append(0)
A.append(nextRow)
return AffineMap(A, b)

#output: dict (dictionary representation)
def serialize(self):
    rep = {}
    rep["permutation"] = self.permutation
    return rep

#input: dict (dictionary representation)
#output: Permutation (represented object)
@staticmethod
def deserialize(rep):
    return Permutation(rep["permutation"])

#output: Boolean (True if identity)
def isIdentity(self):
    for i in range(1, self.dim()):
        if self.permutation[i] != self.permutation[i-1] + 1:
            return False
    return True

#input: Integer^2 (indices of elements to swap)
#action: swap indices
@staticmethod
def swap(dim, n1, n2):
    swapper = Permutation.identity(dim)
    temp = swapper.permutation[n1]
    swapper.permutation[n1] = swapper.permutation[n2]
    swapper.permutation[n2] = temp
    return swapper

@staticmethod
def test():
    swap1 = Permutation.swap(5, 2, 3)
    swap2 = Permutation.swap(5, 4, 1)
    swap3 = Permutation.compose(swap1, swap2)
    print swap1.permutation
    print swap2.permutation
    print swap3.permutation

@staticmethod
def makeZeros(n):
    temp = []
for i in range(0, n):
    temp.append(0.0)
return temp

#Permutation.test()

10.2 Word

from numpy import *
from cddSolver import cddSolver
from Permutation import Permutation
from AffineMap import AffineMap
from ConvexSet import ConvexSet
import SystemCatalog

class Word:
    def __init__(self, sequence):
        self.sequence = sequence
        self.flips = []
        self.feasible = "unknown"
        self.set = "unspeicified"
        self.map = "unspeicified"

    #output: Integer (number oscillators)
    def dim(self):
        if self.sequence == []:
            return 0
        else:
            return self.sequence[0].dim()

    #output: Boolean (True if feasible)
    def testFeasibility(self):
        if self.set == []:
            return False
        else:
            return cddSolver.solve(self.set)

    #output: Permutation (effective action on oscillators represented by word)
    def netTransposition(self):
        initial = self.sequence[0]
        inverse = initial.inverse()
        e = Permutation.identity(self.dim())
        for flip in self.flips:
            e = e.transpose(flip)
        e = Permutation.compose(initial, Permutation.compose(e, inverse))
        return e

    #output: Permutation (last element of sequence of permutations)
    def lastInSequence(self):
```python
return self.sequence[len(self.sequence) - 1]
```

```python
#output: dict (representation)
def serialize(self):
    representation = {}
#flips, feasible should already be ok
    representation["flips"] = self.flips
    representation["feasible"] = self.feasible
#need to do serialization of the maps and permutations
    sequenceList = []
    for p in self.sequence:
        sequenceList.append(p.serialize())
    representation["sequence"] = sequenceList
#map
    representation["map"] = self.map.serialize()
#set
    representation["set"] = self.set.serialize()
    return representation
```

```python
#input: dict (representation)
#output: Word
@staticmethod
def deserialize(representation):
    sequence = []
    for p in representation["sequence"]:
        sequence.append(Permutation.deserialize(p))
    newWord = Word(sequence)
    newWord.flips = representation["flips"]
    newWord.feasible = representation["feasible"]
    newWord.map = AffineMap.deserialize(representation["map"])
    newWord.set = ConvexSet.deserialize(representation["set"])
    return newWord
```

```python
#output: true if FPrimitive Condition Holds
def isFPrimitive(self):
    #only F map
    if self.flips[-1] != "R":
        return False
    #no double F
    for i in range(0, len(self.flips) - 1):
        if self.flips[i] == "R":
            return False
    return True
```

### 10.3 AffineMap

```python
from numpy import *
from numpy import linalg as LA
class AffineMap:
    def __init__(self, A, b):
```
self.A = matrix(A)
self.b = matrix(b)
if self.b.shape[0] != self.A.shape[0]:  # its counterintuitive,
    but shape[0] gives cols in A and rows in b
    self.b = transpose(self.b)

#input: float (scaling constant)
#operation: multiply elements by scalar
def scale(self, scalar):
    tempA = copy.deepcopy(self.A)
    tempb = copy.deepcopy(self.b)
    for i in range(0, len(A)):
        for j in range(0, len(A[0])):
            tempA[i, j] *= scalar
    for i in range(0, len(b)):
        tempb[i] *= scalar

#output: dict (representation)
def serialize(self):
    representation = {}
    representation["A"] = self.A.tolist()
    representation["b"] = self.b.tolist()
    return representation

#output: String or Integer (returns string "stable", "unstable",
or number neutral e.w.)
def stability(self):
    w, v = linalg.eig(self.A)
    maxMod = max(abs(w))
    if maxMod < 0.9999999:
        return "stable"
    elif maxMod > 1.0000001:
        return "unstable"
    else:
        #count number of neutral eigenvalues
        neutralCount = 0
        for ew in w:
            if abs(ew) > 0.9999999:
                neutralCount += 1
        return str(neutralCount)

#input: dict (representation)
#output: AffineMap
@staticmethod
def deserialize(rep):
    return AffineMap(rep["A"], rep["b"])

#input: AffineMap '2 (maps to compose)
#output: AffineMap (composition)
@staticmethod
def compose(m1, m2):
```python
# print ml.A
# print m2.A
tempA = ml.A * m2.A
tempb = ml.A + m2.b + m1.b
return AffineMap(tempA, tempb)

# input Integer (number oscillators)
# output AffineMap (identity map)
@staticmethod
def identity(n):
    A = identity(n)
    b = []
    for i in range(0, n):
        b.append(0)
    return AffineMap(A, b)

10.4 ConvexSet

import numpy as np
import math
import cdd

class ConvexSet:
    def __init__(self, A, b):
        self.A = np.matrix(A)
        self.b = np.matrix(b)
        if self.b.shape[0] != self.A.shape[0]:  # its counterintuitive, but shape[0] gives cols in A and rows in b
            self.b = np.transpose(self.b)

    def __call__(self):
        return ConvexSet(self.A, self.b)

    # return dictionary representation of the set
    def serialize(self):
        representation = {}
        representation["A"] = self.A.tolist()
        representation["b"] = self.b.tolist()
        return representation

    # given dictionary representation, returns map
    @staticmethod
    def deserialize(rep):
        return ConvexSet(rep["A"], rep["b"])

    # pycddlib wants the form (b,-A) for some reason
    # returns matrix of generators (each row is a generator)
    def getVertices(self):
        tableau = np.concatenate((self.b, np.multiply(-1, self.A)), axis=1)  # join b to the right of a to make tableau
        mat = cdd.Matrix(tableau.tolist())
        mat.rep_type = cdd.RepType.INEQUALITY
```

76
poly = cdd.Polyhedron(mat)
vertices = poly.get_generators()
return vertices

# pycddlib wants the form (b, A) for some reason
# return refined set of constraints
def getInequalities(self):
    tableau = np.concatenate((self.b, np.multiply(-1, self.A)),
                              axis=1)  # join b to the right of a to make tableau
    mat = cdd.Matrix(tableau.tolist())
    matcanonicalize()  # get rid of redundancies
    poly = cdd.Polyhedron(mat)
    return poly.get_inequalities()

# writes vertices to file
def writeVertices(self, path):
    f = open(path, "w")
    f.write(str(self.getVertices()))
    f.close()

# return dictionary representing the object
def getDict(self, info):
    stringForm = str(self.getVertices())
    vector = stringForm.split("\n")
    matrix = []
    for line in vector[3:-1]:
        line = line.replace("\", ",")
        line = line.replace("\", ",")
        line = line.replace("\", ",")
        array = line.split("\")[2:]
        matrix.append(array)
    dict = {"points": matrix, "info": info}
    return dict

# returns representation of crosssection (certain variables are
# fixed)
# @param values: value at which to fix each variable, '?' if not
# projected
def standardCrossSection(self, values):
    AxSection = self.A[:, [index for index in range(0, len(values)) if values[index] == '?']]  
    bxSection = self.b - sum(values[index] * self.A[:, index] for index in range(0, len(values)) if values[index] != '?')
    unboundConstraints = ConvexSet(AxSection, bxSection)
    numCols = AxSection.shape[1]
    return unboundConstraints

# return constraints for numCols dimensional cube
@staticmethod
def makeSquareConstraints(numCols):
    id = np.identity(numCols)
```python
b0 = np.zeros(numCols)
b1 = np.ones(numCols)
lower = ConvexSet(np.multiply(-1, id), b0)
upper = ConvexSet(id, b1)
return ConvexSet.intersect(lower, upper)
```

```python
# return intersection of two convex sets
@staticmethod
def intersect(cs1, cs2):
    tempA = np.concatenate((cs1.A, cs2.A))
    tempb = np.concatenate((cs1.b, cs2.b))
    temp = ConvexSet(tempA, tempb)
    # temp.A = tempA
    # temp.b = tempb
    return temp
```

## 10.5 ConvexGeometry

```python
import numpy as np
from ConvexSet import ConvexSet
import cdd

class ConvexGeometry:
    @staticmethod
def preImage(map, set):
        tempA = set.A * map.A
        tempb = set.b - set.A * map.b
        return ConvexSet(tempA, tempb)
```

## 10.6 System

```python
from Word import Word
from Permutation import Permutation
from ConvexGeometry import ConvexGeometry
from ConvexSet import ConvexSet
from AffineMap import AffineMap
from numpy import add, concatenate
from cddSolver import cddSolver

# phi is indexed with innate coordinates
# trans uses order coordinates
# sigma converts innate to order
# sigmaInv converts order to innate
# all constraints of the form Ax <= b
# all constraints use innate coordinates
# all maps use innate coordinates
# all intermediate permutations (transpositions) use order coordinates

class System:
    # a system is defined by dimension and phi function
def __init__(self, dim, phi):
```

78
self.dim = dim
self.phi = phi  # phi is indexed by innate indices

#output: Integer system id
def getId(self):
    return self.id

#input: Permutation
#output: Word (length one, containing Permutation)
def word(self, symbol):
    sequence = []
    sequence.append(symbol)
    temp = Word(sequence)
    # just make map the identity
    temp.map = AffineMap.identity(self.dim)
    # next make the constraints
    constraintMatrix = []
    constraintVector = []
    for q in range(0, symbol.dim() - 1):
        newRow = System.makeZeros(symbol.dim())
        newRow[symbol.sigmaInv(q + 1)] = -1.0
        newRow[symbol.sigmaInv(q)] = 1.0
        constraintMatrix.append(newRow)
        constraintVector.append(0.0)
    # first cell greater than 0
    newRow = System.makeZeros(symbol.dim())
    newRow[symbol.sigmaInv(0)] = -1.0
    constraintMatrix.append(newRow)
    constraintVector.append(0.0)
    # second cell less than 1
    newRow = System.makeZeros(symbol.dim())
    newRow[symbol.sigmaInv(symbol.dim() - 1)] = 1.0
    constraintMatrix.append(newRow)
    constraintVector.append(1.0)
    temp.set = ConvexSet(constraintMatrix, constraintVector)
    return temp

#input: Permutation
#output: Boolean (True if perm is invariant for system)
def invariant(self, perm):
    return perm.sigma(3) == 3 and perm.sigma(4) == 4

#input: Word, Transposition (base word and intermediate transposition)
#output: Word (result of appending transposition of last permutation)
def concat(self, w1, trans):
    #trans uses order indices rather than innate indices
    currentPermutation = w1.lastInSequence()
    newWord = Word([])
newWord.sequence.extend(wl.sequence)
newWord.sequence.append(currentPermutation.transpose(trans))  
# append the result of the transposition?
newWord.flips.extend(wl.flips)
newWord.flips.append(trans)

frequency = self.phi(currentPermutation) # frequency vector 
for current permutation

# new maps
if (trans == "R"):
    indexFront = 0
    indexBack = currentPermutation.sigmaInv(self.dim - 1)
    fback = frequency[indexBack]
    ffront = 0
    newMatrix = []
    newVector = []
    # cyclic reindex: there is none, we’re using innate coordinates
    # just set the xsigmainv−1 to 0
    for i in range(0, self.dim):  
        # i is innate
        temp = System.makeZeros(self.dim)  
        if i != currentPermutation.sigmaInv(self.dim - 1):  
            everywhere except for oscillator hitting 1
            assert(fback != 0)
            freqRatio = frequency[i] / fback
            temp[i] = 1
            temp[indexBack] -= freqRatio
            newMatrix.append(temp)
            newVector.append(freqRatio)
        else:  # the row of the oscillator hitting 1, set to zero
            newMatrix.append(temp)  
            newVector.append(0)
else:
    indexBack = currentPermutation.sigmaInv(trans[0])
    indexFront = currentPermutation.sigmaInv(trans[1])
    fback = frequency[indexBack]
    ffront = frequency[indexFront]
    newMatrix = []
    newVector = System.makeZeros(self.dim)
    for i in range(0, self.dim):  
        # i is innate
        freqRatio = frequency[i] / (fback - ffront)
        temp = System.makeZeros(self.dim)  
        # zero vector
        temp[i] = 1
        temp[indexFront] += freqRatio
        temp[indexBack] -= freqRatio
        newMatrix.append(temp)
    newMap = AffineMap(newMatrix, newVector)
newWord.map = AffineMap.compose(newMap, w1.map)#compose from left

# make row for mainTrans
mainRow = System.makeZeros(self.dim)
if trans == 'R':
    tBack = currentPermutation.sigmaInv(self.dim - 1)#innate
    mainRow[tBack] += -1/frequency[tBack]
    mainConstant = 1/frequency[tBack]
else:
    tFront = currentPermutation.sigmaInv(trans[1])
    tBack = currentPermutation.sigmaInv(trans[0])
    mainRow[tBack] += -1/(frequency[tBack] - frequency[tFront])
    mainRow[tFront] += 1/(frequency[tBack] - frequency[tFront])
    mainConstant = 0
transpositions = self.admissibleTranspositions(
    currentPermutation)
constraintA = []
constraintb = []
for competingTrans in transpositions:
    if competingTrans != trans:
        if competingTrans == 'R':
            ctBack = currentPermutation.sigmaInv(self.dim - 1)#innate
            competingRow = System.makeZeros(self.dim)
            competingRow[ctBack] = -1/frequency[ctBack]#temp
            and frequency use innate indices
            competingConstant = 1/frequency[ctBack]
        else:
            q = competingTrans[1]
            ctBack = currentPermutation.sigmaInv(q-1)
            ctFront = currentPermutation.sigmaInv(q)
            competingRow = System.makeZeros(self.dim)#temp
            will use innate indices since it is used to make constraints
            assert(frequency[ctBack] != frequency[ctFront])
            competingRow[ctBack] = -1/(frequency[ctBack] -
                frequency[ctFront])
            competingRow[ctFront] = 1/(frequency[ctBack] -
                frequency[ctFront])
            competingConstant = 0
        constraintRow = []
        for i in range(0, self.dim):
            constraintRow.append(mainRow[i] - competingRow[i])
        constraintConstant = competingConstant - mainConstant
        constraintA.append(constraintRow)
        constraintb.append(constraintConstant)
# append zero row for empty set of constraints
constraintA.append(System.makeZeros(self.dim))
constraintb.append(0.0)
constraintsAtTrans = ConvexSet(constraintA, constraintb)
constraintsAt0 = ConvexGeometry.preImage(w1.map, constraintsAtTrans)
newWord.set = ConvexSet.intersect(constraintsAt0, w1.set)
return newWord

# input: Permutation
# output: list of dynamically admissible transpositions from given permutation
def admissibleTranspositions(self, perm):
    admissible = []
    frequency = self.phi(perm)
    if (frequency[perm.sigmaInv(self.dim - 1)] > 0):
        admissible.append("R")
    for i in range(1, self.dim):
        fback = frequency[perm.sigmaInv(i - 1)]
        ffront = frequency[perm.sigmaInv(i)]
        if (ffront < fback):
            admissible.append((i - 1, i))
    return admissible

# input: Word
# output: Boolean (True if word corresponds to Poincare map)
def testFixedCellPoincare(self, word):
    wc = word.flips.count('R')
    if len(word.flips) > 0 and wc > 0:
        last = word.flips[-1]
    return word.flips.count('R') == self.dim - 2 and word.flips[-1] == 'R'

# input: Word
# output: Boolean (True if word corresponds to Poincare map for 3 Params)
def testFixedCellPoincare3Params(self, word):
    wc = word.flips.count('R')
    if len(word.flips) > 0 and wc > 0:
        last = word.flips[-1]
    return word.flips.count('R') == self.dim - 3 and word.flips[-1] == 'R'

# input: Word
# output: Boolean (True if word corresponds to F map)
def testFixedCellFMap(self, word):
    wc = word.flips.count('R')
    if len(word.flips) > 0 and wc > 0:
        last = word.flips[-1]

return word.flips.count('R') == 1 and word.flips[-1] == 'R'

# input: Word
# output: ConvexSet (fixed points of map)
def getFixedPoints(self, word):
    # next, obtain the full poincare map
    Fmap = word.map
    # construct the set of constraints for fixed point
    A1 = add(Fmap.A, -1.0 * AffineMap.identity(self.dim).A)
    b2 = Fmap.b
    b1 = -1.0 * b2
    A = concatenate((A1, A2))
    b = concatenate((b1, b2))
    cs = ConvexSet(A, b)
    FixedPoints = cs.intersect(cs, word.set())
    return FixedPoints

# output: permutation for representing set of initial conditions
def makeSymbols(self):
symbols = []

    for sIndex in range(0, 4):
        for rIndex in range(sIndex + 1, 5):
            temp = [0, 1, 2]
            temp.insert(sIndex, 3)
            temp.insert(rIndex, 4)
            symbols.append(Permutation(temp))

    for sIndex in range(0, 4):
        for rIndex in range(sIndex + 1, 5):
            temp = [2, 0, 1]
            temp.insert(sIndex, 3)
            temp.insert(rIndex, 4)
            symbols.append(Permutation(temp))

    for sIndex in range(0, 4):
        for rIndex in range(sIndex + 1, 5):
            temp = [1, 2, 0]
            temp.insert(sIndex, 3)
            temp.insert(rIndex, 4)
            symbols.append(Permutation(temp))

    symbols = list(set(symbols)) # remove duplicates
    return symbols

@staticmethod
def makeZeros(n):
    temp = []
    for i in range(0, n):
        temp.append(0.0)
    return temp
10.7 cddSolver

import cdd
import numpy as np

class cddSolver:
    @staticmethod
    def solve(set):
        tableau = np.concatenate((set.b, np.multiply(-1, set.A)), axis=1)  # join b to the right of a to make tableau
        mat = cdd.Matrix(tableau.tolist())
        mat.obj_type = cdd.LPObjType.MAX
        numCols = set.A.shape[1]
        mat.obj_func = tuple(cddSolver.makeZeros(numCols + 1))
        lp = cdd.LinProg(mat)
        lp.solve()
        return (lp.status == cdd.LPStatusType.OPTIMAL)

    # given convex set return cddMatrix of vertices
    @staticmethod
    def getVertexMatrix(set):
        tableau = np.concatenate((set.b, np.multiply(-1, set.A)), axis=1)  # join b to the right of a to make tableau
        mat = cdd.Matrix(tableau.tolist())
        matcanonicalize()  # get rid of redundancies
        mat.reptype = cdd.RepType.INEQUALITY
        poly = cdd.Polyhedron(mat)
        vertices = poly.get_generators()
        return vertices

    # converts cddMatrix to list matrix
    @staticmethod
    def oldCddToList(cddMatrix):
        numrows = cddMatrix.row_size
        numcols = cddMatrix.col_size

        matrix = []
        for i in range(0, numrows):
            temp = []
            for j in range(0, numcols):
                temp.append(cddMatrix[i][j])
            matrix.append(temp)
        return matrix

    # converts cddMatrix to list matrix
    @staticmethod
    def cddToList(cddMatrix):
        stringForm = str(cddMatrix)
        vector = stringForm.split("\n")
        matrix = []
        for line in vector[3:-1]:
            line = line.replace("∞", "∞")
line = line.replace("\r", "")
line = line.replace("\n", "")
array = line.split("\n")[2:]
temp = []
for element in array:
    temp.append(float(element))
matrix.append(temp)
return matrix

#returns image of vertices (list matrix representation) after application of map
@staticmethod
def image(vertices, map):
    temp = []
    for row in vertices:
        mat = np.matrix(row)
        mat = np.transpose(mat)#make row stand up for operation
        mat = map.A * mat + map.b
        mat = np.transpose(mat)#make row lie back down
        temp.append(np.array(mat)[0].tolist())
    return temp

#returns json representation of vertices
@staticmethod
def oldGetJson(vertices, info):
    vertices = cdd.Matrix(vertices)
    stringForm = str(vertices)
    vector = stringForm.split("\n")
    matrix = []
    for line in vector[3:1]:
        line = line.replace("\r", "")
        line = line.replace("\n", "")
        line = line.replace("\r", "")
        array = line.split("\n")[2:]
        matrix.append(array)
    dict = {"points": matrix, "info": info}
    return dict

#returns json representation of vertices from matrix rep
@staticmethod
def getJson(vertices, info):
    dict = {"points": vertices, "info": info}
    return dict

@staticmethod
def makeZeros(n):
    temp = []
    for i in range(0, n):
        temp.append(0.0)
    return temp
10.8 SystemCatalog

from System import System

# defines a list of Systems

# negative feedback

def Sysphi(perm):
    # let 3 ~ s, 4 ~ r
    sIndex = perm.sigma(3)
    temp = []
    for i in range(0,3):
        if perm.sigma(i) < perm.sigma(4): # case: not in R
            temp.append(1)
        else: # case in R
            temp.append(1-sIndex * (1.0/10.0))
    for i in range(3,5): # parameters
        temp.append(0)
    return temp

system0 = System(5, Sysphi)

# no feedback

def Sysphi1(perm):
    # let 3 ~ s, 4 ~ r
    sIndex = perm.sigma(3)
    temp = []
    for i in range(0,3):
        if perm.sigma(i) < perm.sigma(4): # case: not in R
            temp.append(1)
        else: # case in R
            temp.append(1-sIndex * (0.0/10.0))
    for i in range(3,5): # parameters
        temp.append(0)
    return temp

system1 = System(5, Sysphi1)

# positive feedback

def Sysphi2(perm):
    # let 3 ~ s, 4 ~ r
    sIndex = perm.sigma(3)
    temp = []
    for i in range(0,3):
        if perm.sigma(i) < perm.sigma(4): # case: not in R
            temp.append(1)
        else: # case in R
            temp.append(1-sIndex * (-1.0/10.0))
    for i in range(3,5): # parameters
        temp.append(0)
    return temp

system2 = System(5, Sysphi2)

systemList = []
systemList.append(system0)
systemList.append(system1)
systemList.append(system2)

10.9 Navigator

from ConvexSet import ConvexSet
from Permutation import Permutation
from AffineMap import AffineMap
from numpy import add, concatenate
import pickle
import SystemCatalog
from Word import Word
from cdSolver import cdSolver

class Navigator:
    def __init__(self, systemId):
        self.system = SystemCatalog.systemList[systemId]
        self.systemId = systemId
        self.wordsGenerated = []
        self.length = 0
        self.cutoff_length = 12

        # construct set of words
        def navigate(self):
            self.wordsGenerated = []
            # generate representatives of each symmetry class (currently not gen purpose)
            symbols = self.system.makeSymbols()
            gen1 = []
            for symbol in symbols:
                newWord = self.system.word(symbol)
                gen1.append(newWord)
                self.wordsGenerated.append(newWord)
            generations = []
            generations.append(gen1)
            for currentLength in range(2, self.cutoff_length + 1):
                print("generation" + str(currentLength))
                currentList = generations[-1]
                nextList = []
                for word in currentList:
                    for transposition in self.system.admissibleTranspositions(word.lastInSequence()):
                        candidateWord = self.system.concat(word, transposition)
                        if candidateWord.testFeasibility():
                            nextList.append(candidateWord)
                            self.wordsGenerated.append(candidateWord)
                            generations.append(nextList)
                self.length = self.cutoff_length

            # extend set of words generated
def iterate(self):
    iterationIndex = 0
    totalIterations = len(self.wordsGenerated)
    for word in self.wordsGenerated:
        iterationIndex += 1
        print("Progress: {} / {}".format(iterationIndex, totalIterations))
        if len(word.sequence) == self.length:
            for transposition in self.system.admissibleTranspositions(word.lastInSequence()):
                candidateWord = self.system.concat(word, transposition)
                if candidateWord.testFeasibility():
                    self.wordsGenerated.append(candidateWord)
                    self.length += 1

#write pickle representation to file
def writePickle(self):
    with open("./Pickle/wordsGenerated.pickle", "wb") as f:
        pickle.dump(self.serialize(), f)

#input: String (filename)
#output: Navigator object represented by Pickle file
@staticmethod
def readPickle(fileName):
    rep = pickle.load(open("./Pickle/" + str(fileName), "rb"))
    return Navigator.deserialize(rep)

#output dict (representation)
def serialize(self):
    rep = {
        "systemId": self.systemId,
        "wordsGenerated": [],
    }
    for word in self.wordsGenerated:
        word.reps.append(word.serialize())
        rep["wordsGenerated"] = word.reps
    rep["length"] = self.length
    rep["cutoff_length"] = self.cutoff_length
    return rep

@staticmethod
#input: dict (representation)
#output: Navigator
def deserialze(rep):
    navi = Navigator(rep["systemId"],)
    wordsGenerated = []
    for word in rep["wordsGenerated"]: wordsGenerated.append(Word.deserialze(word))
    navi.wordsGenerated = wordsGenerated
    navi.length = rep["length"]
nave.cutoff_length = rep["cutoff_length"]
return nave

# upload relevant data about the word to the cloud
def sendToCloud(self):
    # client = MongoClient('localhost', 28017)
    client = MongoClient()
    db = client.Hilbert
    wordCollection = db.words
    representation = self.serialize()
    wordCollection.insert(representation)

@staticmethod
def SingleIteration():
    nave = Navigator.readPickle("wordsGenerated.p")
    print(str(nave.length) + ",generations")
    nave.iterate()
    print("now", str(nave.length) + ",generations")
    nave.writePickle()

10.10 LocalReportBuilder

from Navigator import Navigator
from ConvexSet import ConvexSet
from cddSolver import cddSolver
import SystemCatalog
import json
from ConvexGeometry import ConvexGeometry
import math

class LocalReportBuilder:
    def __init__(self, sysCode):
        #print("reading pickle")
        #self.nave = Navigator.readPickle("wordsGenerated12.p")
        #print("pickle read")
        self.nave = Navigator(sysCode)
        self.nave.navigate()
        self.words = self.nave.wordsGenerated
        self.system = SystemCatalog.systemList[sysCode]
        self.syscode = sysCode

        #input s: Float (end of signalling region)
        #input r: Float (beginning of response region)
        #input name: String (what to call the data file)
        #input condition: function (how to choose words to include)
        #input numImages: integer (number of times to take image)
        #input Fmap: weather or not to use Fmap instead of Poincare Map
    def makeXSectionReport(self, s, r, name, condition, numImages, Fmap):
        rParam = r
        sParam = s

89
dict = {}
dict["regions"] = []
crossSections = []

for word in self.words:
    # test if condition applies to word
    if condition(word):
        stability = "unknown"
        # new logic
        crossSection = word.set.standardCrossSection([0, '?', '?', sParam, rParam])
        # crossSection = word.set.standardCrossSection([0, 0.25, 0.75, '?', '?'])

        # solution = cvxSolver.solve(crossSection)
        # if solution["status"] == "optimal":
            if cddSolver.solve(crossSection):
                cddVertexMatrix = cddSolver.getVertexMatrix(crossSection)
                listVertexMatrix = cddSolver.cddToList(cddVertexMatrix)
                stability = word.map.stability()

                fixedPointSet = self.system.getFixedPoints(word)
                if cddSolver.solve(fixedPointSet):
                    fixedPointCrossSection = fixedPointSet.standardCrossSection([0, '?', '?', sParam, rParam])
                    cddFixedPointMatrix = cddSolver.getVertexMatrix(fixedPointCrossSection)
                    listFixedPointMatrix = cddSolver.cddToList(cddFixedPointMatrix)
                else:
                    listFixedPointMatrix = False

        # code for images
        if numImages > 0:
            for row in listVertexMatrix:
                # put the other params back inside
                # print("length of row " + str(len(row)))
                # print(str(row))
                row.insert(0, 0)
                row.append(sParam)
                row.append(rParam)
                # print("length of row after " + str(len(row)))

            for j in range(0, numImages):
                listVertexMatrix = cddSolver.image(listVertexMatrix, word.map)

        # now select the wanted region
        for row in listVertexMatrix:
del row[4] # get rid of r
del row[3] # get rid of s
if Fmap:
    del row[2] # get rid of element at 0
else:
    del row[0] # get rid of zero

# somehow compute vertices of the set in 2d
# vertices = LocalReportBuilder.computeVertices(tempSet)

cxName = ""
for event in word.flips:
    cxName += str(event)
cxName = cxName.replace("\", "")
cxName = cxName.replace("/", "")
cxName = cxName.replace("[", "")
cxName = cxName.replace("]", "")
cxName = cxName.replace(",", "")
cxName = cxName.replace(" '", "")

# crossSection.writeIne(cxName, ".\data/" + cxName + ".ine")
# crossSection.writeVertices(".\data/" + cxName + ".ver")
jsonDict = cddSolver.getJson(listVertexMatrix, cxName)# get string rep
jsonDict["fixedPoints"] = listFixedPointMatrix
jsonDict["stability"] = stability
# dict["regions"].append(crossSection.getDict(cxName))
dict["regions"].append(jsonDict)

f = open("data/xSection"+name+".js", "w")
f.write("var jsonObject = ")
f.write(json.dumps(dict))
f.close()

tnDict = {}
tnDict["data"] = dict
tnDict["r"] = rParam
tnDict["s"] = sParam
return tnnDict

References


