EXPLORING RELATIONSHIPS BETWEEN NETWORK STRUCTURE & MANIFOLD STRUCTURE IN SIMPLE PATH MASS ACTION MODELS

by

George Evans

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

April 2021

BRIGHAM YOUNG UNIVERSITY - IDAHO

DEPARTMENT APPROVAL

of a senior thesis submitted by

George Evans

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

Date	Mark Transtrum, Advisor
Date	Richard Datwyler, Committee Member
Date	Lance Nelson, Committee Member
Date	R. Todd Lines, Chair

ABSTRACT

EXPLORING RELATIONSHIPS BETWEEN NETWORK STRUCTURE & MANIFOLD STRUCTURE IN SIMPLE PATH MASS ACTION MODELS

George Evans

Department of Physics and Astronomy

Bachelor of Science

Models describing how concentrations change over time in chemical reaction networks can become very complicated very fast. To help simplify such models, researchers often want to know the manifold structure of such models. As such, I explore the relation between network structure and manifold structure in mass action models, hoping to shorten the process of discovering the manifold structure. In this report, I describe some tools I created that will help future studies. I will also expose some complications that arise when trying to relate network structure to manifold structure.

ACKNOWLEDGMENTS

I would like to thank Mark Transtrum for welcoming me into this project and helping me through this whole endeavor. I would like to thank the faculty of Brigham Young University - Idaho for helping me gain the physics, mathematical, and computational competency needed to perform this research. I especially wish to acknowledge Richard Datwyler and Lance Nelson for their aid in refining this thesis and Todd Lines for his guidance in general. I also wish to thank Richard Sandberg at Brigham Young University for helping me get in contact with Dr. Transtrum in the first place.

Furthermore, I would like to thank my "physics friends," as I like to call them, for helping me get through these last few semesters, as well as my other friends for helping me take a break from school every once in a while.

I thank my parents and siblings for their support throughout college, high school, and life.

Most importantly to me, I thank my wife for her love and support that has helped me find joy despite the pandemic troubles compounded with stress due to school.

Contents

List of Figures

Chapter 1

Introduction and Background

1.1 Overview

Models, whether mathematical, computational or experimental, are one of the fundamental parts of science. They help us take what we know and make predictions about what we don't know. More specifically, we put in initial conditions, parameters, and points of measurement for the system being modeled, then the model gives us predictions for the behavior of the system at those points of measurement. Such models enable much of modern life, allowing inventors to test devices before they've been built, hiring managers to estimate how many employees they can afford, city planners to design effective road systems, etc.

However, modeling a complex system can rapidly become computationally intensive. To enable accurate yet effective models of complex systems, Dr. Mark Transtrum and his team have created a method by which they could systematically reduce the number of parameters needed to create an effective model, and they have been actively using it on models of power systems, developmental biology, and a few other, complex systems. This method, known as the Manifold Boundary Approximation

Method (MBAM), is explained well in Transtrum and Qiu's article "Model Reduction by Manifold Boundaries" [1]. It is used to find useful reduced models out of the set of all possible reduced models [2]. This set of all possible reduced models is called the information topology of the model [2, 3], and I spent my internship studying the information topology of mass action models.

Specifically, my task was to explore the possibility of applying Dr. Transtrum's research in a new area: nonlinear mass action models (see Section 1.9). Dr. Transtrum's team has already done some research with a similar class of models, linear compartment models (LCMs), and their discoveries led them to hypothesize that mass action models might behave similarly. If that were the case, the team's efforts in applying MBAM to LCMs would easily carry over to mass action models.

Over the course of the summer, I found that mass action models can be treated like linear compartment models, but only in a special case (see Section 3.1). In general, nonlinear mass action models turned out to be trickier to analyze than similarly sized LCMs. Furthermore, even linear mass action models had some complications that LCMs don't have. This document, after providing an introduction to relevant terminology and methodology, discusses various ways to recognize when a mass action model deviates from the analogous LCM, along with the likely effects of such deviations. I also propose a method of categorizing mass action models to guide future research.

1.2 Model Reduction

There are many different ways to categorize models. When it comes to model reduction, it is useful to think of models that were developed using a big-picture approach versus a small-picture approach (also called macroscopic versus microscopic [3]). The

big-picture approach focuses on describing the macroscopic details of what is going on, ignoring the bits and pieces. For example, $PV = NkT$, the ideal gas law, describes how a gas behaves when there are many, many particle, completely ignoring the microscopic movements of the particles involve [3]. This type of model is usually computationally simple and describes the overall idea of what is going on. As it turns out, this simple model, or a similar model with some small corrections added, is often all scientists and engineers need in order to accomplish their goals. Unfortunately, big-picture models can easily gloss over complications that arise and obscure *why* the system behaves the way it does.

On the other side of the spectrum, models built using a small-picture approach are built up by taking every single part of the system and connecting them together. For example, scientists trying to understand how the human body works have an extensive list of different proteins in the body and how they interact with each other. This allows for extreme accuracy in models of such systems, along with remarkable clarity regarding why the the system behaves as it does. However, such models are computationally intensive, often making them impractical to work with.

Bridging these two realms is both tricky and the essence of model reduction. Model reduction is about taking complex, accurate, small-picture models and simplifying them to be more manageable computationally. Unfortunately, many common methods of model reduction cause considerable loss of accuracy and clarity as the models are reduced [2], which is a large part of why my internship mentor has been striving to develop a better method of model reduction. His objective is to start with understandable, accurate, small-picture models, then simplify them in a way that preserves most of their clarity and accuracy while also making them manageable computationally [2].

1.3 Models — What Goes Into Them?

As mentioned before, models need inputs in order to make predictions. In this paper, there are two main types of inputs: parameters and independent variables. The most common independent variables are time and position, although any variable that defines the conditions under which you took a measurement could be considered an independent variable (e.g., the temperature at which the measurements were taken). Parameters, meanwhile, can be considered constant with respect to the independent variables. For example, consider the radioactive decay of a single radioactive isotope. Such a model is often of the form $f(t) = a_0 e^{-\theta t}$, where θ is the decay constant and a_0 is the initial concentration of the isotope. Note that θ is constant with respect to time; as such, θ is a parameter in this system. Meanwhile, a_0 can change depending on when we started measuring the concentration, so it will be considered an independent variable in this report.

Often, we are given a model with predefined parameters and asked to see what it predicts. For example, we might be asked to use our radioactive decay model and the decay rate of Technetium-99m (^{99m}Tc) to predict a later concentration of ^{99m}Tc given any initial concentration of ^{99m}Tc. Alternatively, we could be given a set of independent variables and asked to see what parameters make the model fit them. Such models are called parametric models.

To demonstrate how a parametric model might work, consider again a model of radioactive decay:

$$f(\vec{\theta}) = a_0 e^{-\theta_1 \vec{t}} + b_0 e^{-\theta_2 \vec{t}}. \tag{1.1}$$

In this model, suppose we already know the times at which the system was measured and the initial concentrations of the radioactive substance. Specifically, suppose $\vec{t} = (1, 2, 3)$ and $a_0 = b_0 = 1$. With such a model, we can try out different parameters to

see what the outcome would be. For example, suppose we put $\vec{\theta} = (\theta_1, \theta_2) = (0.1, 0.5)$ into our model. The output would be the prediction vector $f(\vec{\theta}) = (e^{-0.1}+e^{-0.5}, e^{-0.2}+e^{-1}, e^{-0.3}+e^{-1.5}) \approx (1.51137, 1.18661, 0.963948)$. Such models are especially useful when fitting data because we usually know the independent variables of the system (such as the concentrations of a radioactive isotope at different times), but we may not know the parameters (such as the decay constant).

1.4 Data Fitting

When Kepler discovered the laws that bear his name, he did so by trying various parameters in various models and, by hand, seeing if the model's predictions fit the data. Nowadays, we have computational tools that help us derive models for our systems. With these tools, we can take data points, their associated independent variables, and a model that we think will fit the data, feed them into the computer, and expect that the computer will tell us the likely parameters associated with the model. With those parameters, we can then go on to predict what will happen next in that specific system or, if it's a system that can be replicated, in another similar situation. For example, if we knew how much of a radioisotope there was at several different time points, we could feed that data into a computer along with the model $f(\theta) = a_0 e^{\theta \vec{t}}$ to find what the decay constant is likely to be.

In order for a computer to know how close a prediction is to the data, it needs a metric of some sort. Suppose we have a set of parameters $\vec{\theta}$. Plugging those parameters into the model, we get a set of predictions $f(\vec{\theta})$. Next, we check to see how close this set of predictions is to the measured data (denoted $\vec{d}$) with the equation [2]

$$C(\vec{\theta}) = \sum_{i=1}^{N} (f(\vec{\theta}) - \vec{d})^2. \tag{1.2}$$

This function, called the cost function, is a metric. Specifically, it tells us how far the predictions are from the data. Data fitting is all about making the cost function as small as it can be. Because data seldom perfectly fit even the correct model, we don't expect the cost function to go to zero. Nevertheless, finding the minimum of the cost function identifies those parameter values that best fit the data.

1.5 Sloppy Models

Despite the many benefits of data fitting, there are many potential ways that it can go wrong. In this document, we particularly care about when the model is what we call "sloppy," a type of model characterized by the fact that certain parameters or combinations of parameters can vary wildly without changing the prediction of the model [2, 4]. In such a situation, it can be extremely difficult, if not impossible, to get accurate predictions of the parameters without having a lucky guess or prior knowledge [2].[1]

The prototypical example of a sloppy model is the sum of decaying exponentials [2]:[2]

$$f(\vec{\theta}) = e^{-\theta_1 \vec{t}} + e^{-\theta_2 \vec{t}} \tag{1.3}$$

Noting that, in order to remain decaying exponentials, we must have $\theta_1 \geq 0$ and $\theta_2 \geq 0$, it is possible to generate a log-log plot that gives a qualitative idea of how

[1] Much of the motivation for model reduction is to convert sloppy models into more rigid models, allowing data fitting to be more effective [2]

[2] This is the radioactive decay model discussed earlier, with the initial concentrations again set to 1 for convenience in discussion.

changing the parameters changes the predictions. We do this by taking the value of the cost function for a set of parameters and displaying it via a contour plot. Such a plot, called the cost contour, can be seen in Figure 1.1.

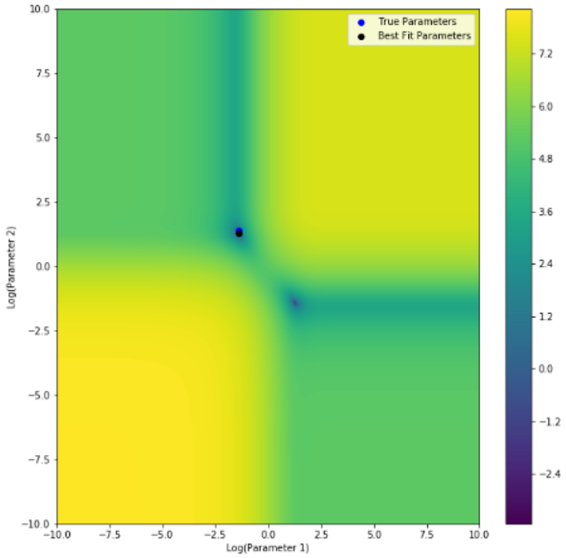


Figure 1.1 The cost contour of $f(\vec{\theta}) = e^{-\theta_1 \vec{t}} + e^{-\theta_2 \vec{t}}$. Darker colors mean the corresponding set of parameters generate predictions that are closer to the actual data, as seen by the scale on the right. Because this is a log-log plot, the far right and top of the graph correspond to when $\theta_1 \approx \infty$ and $\theta_2 \approx \infty$, respectively. Similarly, the far left and bottom of the graph correspond to when $\theta_1 \approx 0$ and $\theta_2 \approx 0$, respectively.

Notice the four regions on the corners where the cost function is approximately constant. This tells us that, as a parameter goes towards infinity or to zero, it has less and less of an effect on the predictions of the model. Such behavior is why the sum of decaying exponentials model is considered a sloppy model: there are ways in which the parameters can vary wildly without changing the prediction [2]. As a quick demonstration, consider the difference between setting θ_1 equal to ten thousand versus ten million. In both cases, $e^{-\theta_1} \approx 0$ — to over 4000 decimal places. Note that this also illustrates why a data fitting algorithm may have a difficult time finding the correct parameters: If the parameters seeded into the algorithm are in one of

these regions where the predictions don't change significantly even with significant changes in the parameters, the algorithm will struggle to discern how it should adjust its guesses to find the parameters that best fit the data.

Another simple example is the model $f(\vec{\theta}) = \frac{\theta_1}{\theta_2}t$. In this case, only the ratio between θ_1 and θ_2 matters — you can multiply θ_1 by whatever you want and still get the same prediction so long as you also multiply θ_2 by the same thing. In such situations, it becomes nearly impossible to find the actual parameters from data, and the best you can usually get is the ratio between the parameters (see Figure 1.2 for the cost contour of the model $f(\vec{\theta}) = \frac{\theta_1}{\theta_2}t$)

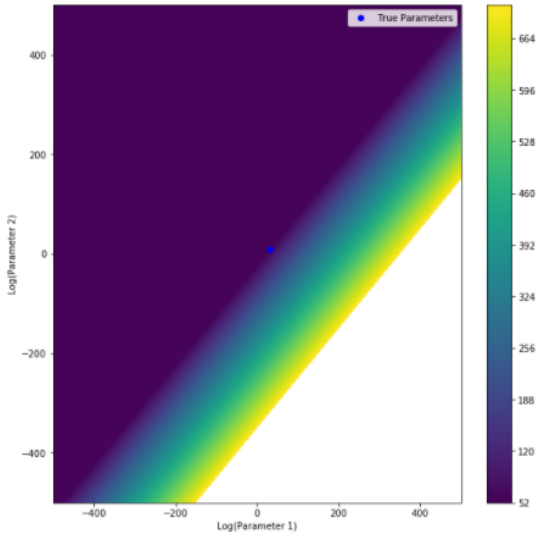


Figure 1.2 The cost contour of $f(\vec{\theta}) = \frac{\theta_1}{\theta_2}t$. Notice that the color doesn't vary if the ratio between θ_1 and θ_2 doesn't change. (The bottom corner is white because taking θ_2 to zero while θ_1 is not zero eventually generates an error as $\frac{\theta_1}{\theta_2}$ approaches infinity)

1.6 The Model Manifold

Our newfound understanding of the cost contour of the model now lets us better understand the parameter space of the model. The parameter space of a model is

the n-dimensional Euclidean space that houses all possible combinations of the n parameters used in the model. Each axis corresponds with a parameter, and each point in this space denotes a combination of parameters, often described with a parameter vector.

Throughout this paper, we've been discussing the sum of decaying exponentials:

$$f(\vec{\theta}) = e^{-\theta_1 \vec{t}} + e^{-\theta_2 \vec{t}}. \tag{1.4}$$

In this model, the parameter space is two dimensional, with one axis corresponding to θ_1 and the other axis corresponding to θ_2, as in the Figure 1.1's cost contour. In fact, the cost contour is a convenient representation of the parameter space for this model. Each possible set of parameters that could be plugged into the model correspond to a point in the parameter space.

When the input parameters are thought of this way, we can begin to see the model as a mapping from the parameter space to what is called the prediction space (also known as the data space) [2]. The prediction space is the m-dimensional Euclidean space that houses all possible predictions of the model given the m values of independent variables at which you want to make a prediction. If we measure the system twice, such as when $\vec{t} = (0.5, 2)$, our prediction space will be two dimensional — the first axis would correspond to all the possible measurements of the system after one second, while the second axis would correspond to all the possible measurements after two seconds. It is important to note that, generally, sloppy models do not fill the prediction space [2]. For example, our model of exponential decay cannot predict that there can be more of the radioisotope than we started with,[3] even though there are circumstances where that may happen.

When we do map the entire parameter space to the prediction space, we get

[3] Assuming $t > 0$

a cross section of what's called the model manifold, specifically the cross section corresponding to the specific set of values for the independent variables you used as axes [2]. A manifold is a topological space that, when viewed on a local ("very small") level, is identical to a Euclidean space no matter where you view it [5]. This particular manifold is the set of all possible predictions of the model at the specific values of the independent variable, such as when $t = (0.5, 2)$. We can see an example of this in Figure 1.3. To get the full manifold, we would need to map the parameter space to an infinite dimensional prediction space that has an axis corresponding to a value of f at each possible value of a independent variable. Fortunately, the model manifold of many simple models can be visualized adequately using a three-dimensional or even two-dimensional cross section, which is what we will do throughout this report.

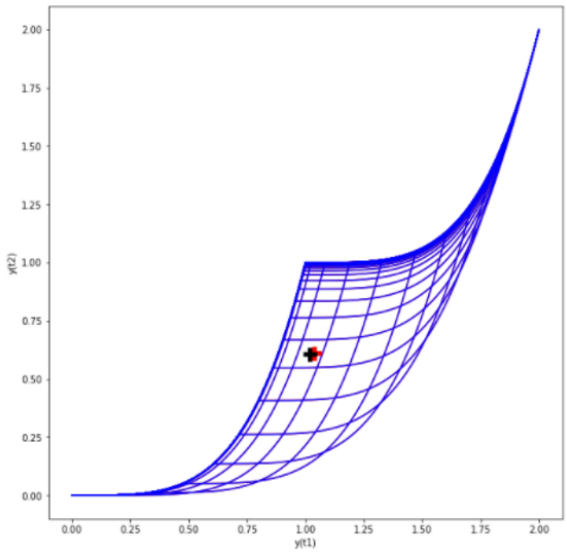
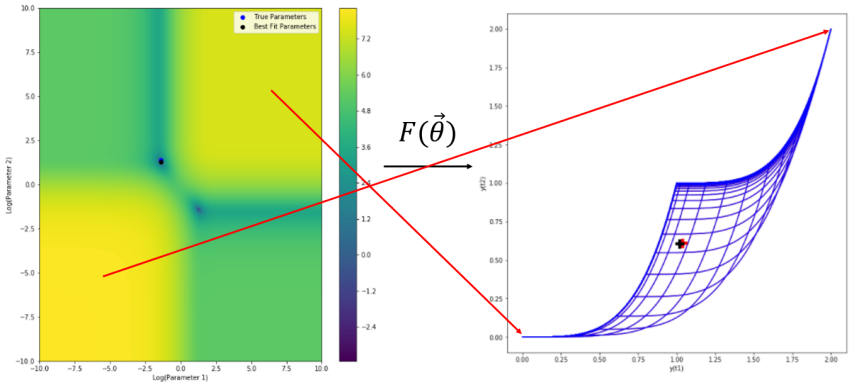


Figure 1.3 The model manifold of $f(\vec{\theta}) = e^{-\theta_1 \vec{t}} + e^{-\theta_2 \vec{t}}$.

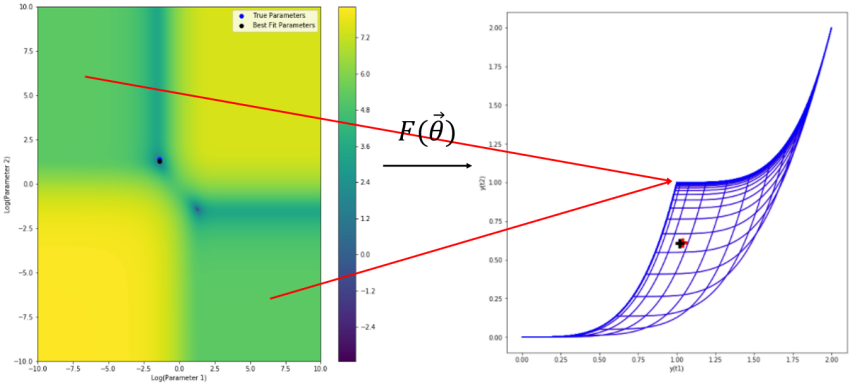
In Figure 1.4, I make connections between the cost contour and the model manifold to show how the parameter space maps to a bound manifold. Notice how the plateaus, regions with near-uniform prediction vectors, all map to points while the canyons, regions which have constant prediction vectors when the parameters are varied in

one direction but not another, map to curves, specifically curves on the edge of the manifold because one of the parameters has been taken to its limits. Finally, notice that the basins, small regions in the center of the graph where changing the parameters by a little bit in any direction affects the prediction, thus map to the majority of the model manifold. With this in mind, a structure begins to emerge: there are three boundary points, or corners, mapped to by the three unique regions[4]; three curves, or edges, mapped to by the three canyons; and a two-dimensional region mapped out by the basins. From this, we can guess that this particular model manifold is topologically equivalent to a triangle, as we saw in Figure 1.3. As such, if our guess was correct, the manifold structure of this model is a triangle. We will discuss this more quantitatively in the next section.

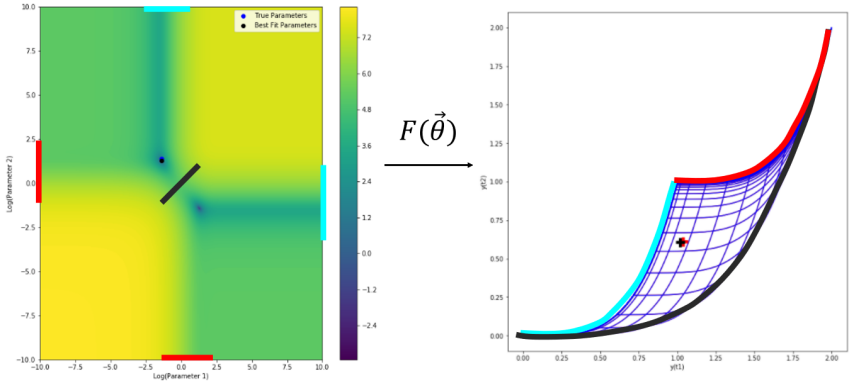
[4]Because swapping θ_1 for θ_2 doesn't affect the prediction of the model, there is symmetry such that the bottom-right half of the parameter space maps to the exact same region as the upper-left half of the parameter space



(a) The top right region of the cost contour (where $\theta_1 \approx \theta_2 \approx \infty$) maps approximately to the single point "0" in the prediction space. The bottom left region of the cost contour (where $\theta_1 \approx \theta_2 \approx 0$) approximately maps to the single point "2" in the prediction space.



(b) The top left and bottom right regions of the cost contour map approximately to the single point "1" in the prediction space.



(c) Crossing the canyons from one region to another is equivalent to moving along the edge of the model manifold.

Figure 1.4 Connecting the cost contour of $f(\vec{\theta}) = e^{-\theta_1 \vec{t}} + e^{-\theta_2 \vec{t}}$ to its model manifold.

1.7 Manifold Structure

Now that we have a qualitative idea of what a manifold structure is, it's time to gain a quantitative idea. We start yet again with our sum of decaying exponentials model:

$$f(\vec{\theta}) = e^{-\theta_1 t} + e^{-\theta_2 t}. \tag{1.5}$$

There are three different ways we can reduce this model such that we are left with a model that maps to an edge of the model manifold. First, we can take the limit of $f(\vec{\theta})$ as θ_2 goes to infinity:

$$\lim_{\theta_2 \to \infty} f(\vec{\theta}) = e^{-\theta_1 t} + 0 \tag{1.6}$$

$$= e^{-\theta_1 t}. \tag{1.7}$$

Notice that this reduced model has only one parameter remaining in it. As such, its model manifold will be one dimensional. Furthermore, because one of the original parameters has been taken to one of its limits (infinity), we know the reduced model manifold corresponds to an edge of the original model manifold. To find out which edge it corresponds to, we can take another limit as θ_1 goes to 0 or as θ_1 goes to infinity:

$$\lim_{\theta_1 \to 0} (e^{-\theta_1 t}) = 1 \tag{1.8}$$

$$\lim_{\theta_1 \to \infty} (e^{-\theta_1 t}) = 0. \tag{1.9}$$

Thus, the reduced model obtained when θ_2 is taken to zero corresponds to the edge of the model manifold that connects the points "1" and "0," as illustrated in Figure 1.5.

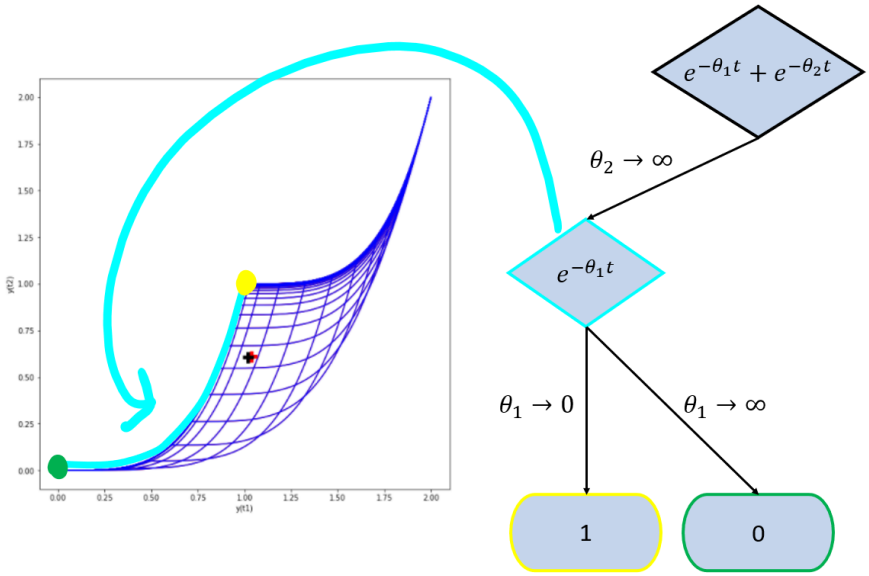


Figure 1.5 Taking θ_2 to infinity leaves behind a model that maps to the cyan edge of the model manifold. Taking another limit as θ_1 goes to zero or infinity leaves behind a model that maps to the yellow or green dots, respectively.

Another way to reduce our original model by one parameter is to take the limit of $f(\vec{\theta})$ as θ_2 goes to zero:

$$\lim_{\theta_2 \to 0} f(\vec{\theta}) = e^{-\theta_2 t} + e^0 \tag{1.10}$$

$$= e^{-\theta_2 t} + 1. \tag{1.11}$$

Again, this model has only one parameter remaining in it, and again, its model manifold corresponds to an edge of the model manifold. Taking limits as θ_1 goes to zero or infinity gets us

$$\lim_{\theta_1 \to 0} (e^{-\theta_2 t} + 1) = 2 \tag{1.12}$$

$$\lim_{\theta_1 \to \infty} (e^{-\theta_2 t} + 1) = 1. \tag{1.13}$$

As such, we find that this reduced model corresponds with the edge that connects the points "1" and "2," as illustrated in Figure 1.6.

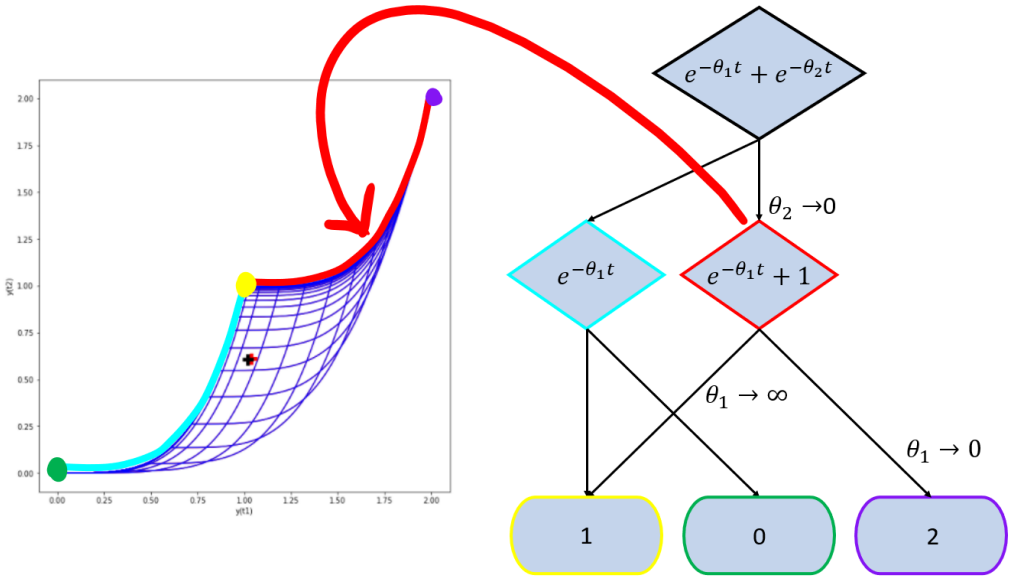


Figure 1.6 Taking θ_2 to zero leaves behind a model that maps to the red edge of the model manifold. Taking another limit as θ_1 goes to zero or infinity leaves behind a model that maps to the purple or yellow dots, respectively.

Finally, we find the last reduced model by taking the limit as θ_2 goes to θ_1:

$$\lim_{\theta_1 \to \theta_2} f(\vec{\theta}) = e^{-\theta_2} + e^{-\theta_2 t} \tag{1.14}$$

$$= 2e^{-\theta_2 t}. \tag{1.15}$$

From there, taking the remaining limits gets us

$$\lim_{\theta_1 \to 0} (2e^{-\theta_1 t}) = 2 \tag{1.16}$$

$$\lim_{\theta_1 \to \infty} (2e^{-\theta_1 t}) = 0, \tag{1.17}$$

which connects the last two points. This final side is illustrated in Figure 1.7. Note that we don't need to worry about taking θ_1 to zero, infinity, or θ_2 because, by symmetry, it would produce the exact same models that we just found, just with different labels.

The structure built on the right in Figure 1.7 (and seen full more detail in Figure 1.8) is known as a Hasse diagram. Hasse diagrams describe the information topology

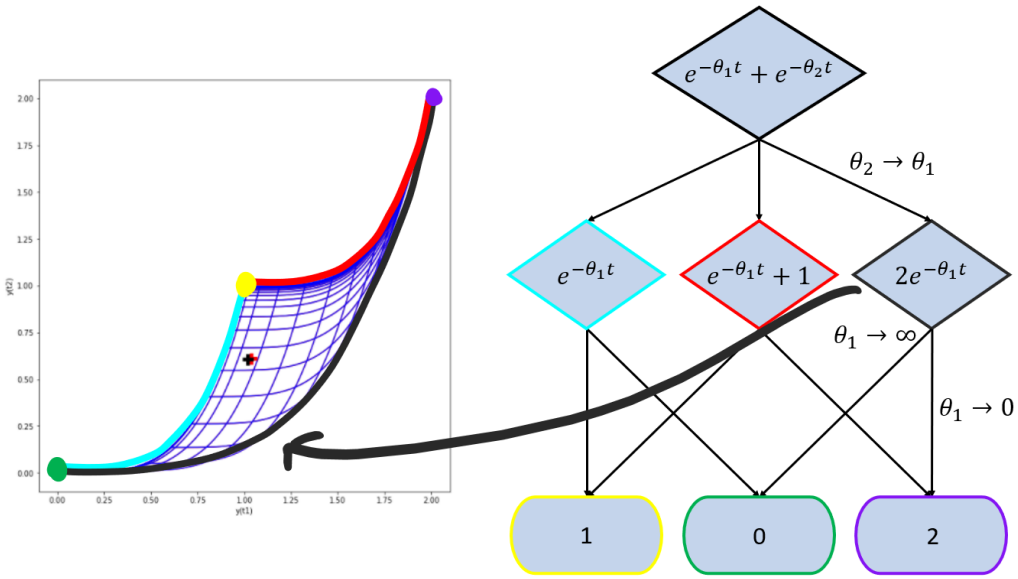


Figure 1.7 Taking θ_2 to θ_1 leaves behind a model that maps to the black edge of the model manifold. Taking another limit as θ_1 goes to zero or infinity leaves behind a model that maps to the purple or green dots, respectively.

of a model, which directly corresponds to the topological structure of the model manifold. As such, we consider the Hasse diagram we built to be the manifold structure of the model.

1.8 Cross Sections of Model Manifolds

Earlier, when introducing the model manifold, I mentioned that we needed to map the parameter space to an infinite dimensional prediction space that has an axis corresponding to the value of f at each possible value of each independent variable in order to get the full model manifold. I also mentioned that the model manifold of many simple models can be visualized adequately using a two- or three-dimensional model. In this section, I will elaborate on why these cross sections are often a good estimate of what the larger manifold is like.

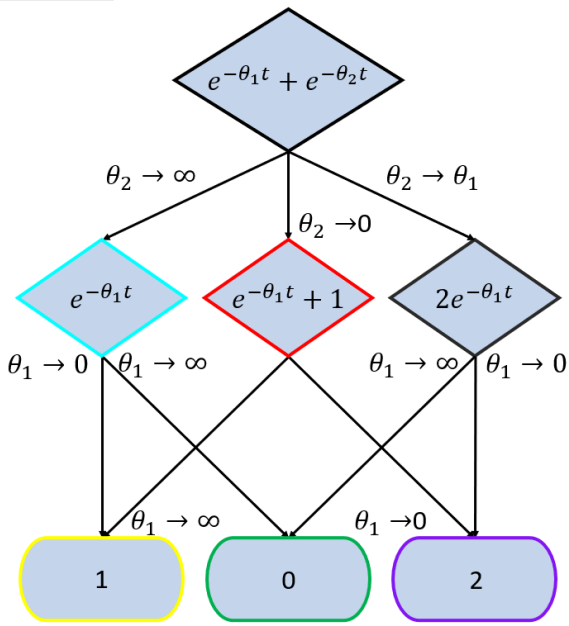


Figure 1.8 Hasse diagram/manifold structure of $f(\vec{\theta}) = e^{-\theta_1 t} + e^{-\theta_2 t}$

Throughout this section, we'll focus on the radioactive decay model:

$$f(\vec{\theta}) = a_0 e^{-\theta_1 t} + b_0 e^{-\theta_2 t}. \tag{1.18}$$

This is exactly like the sum decaying exponentials, just with the addition of initial concentrations a_0 and b_0 of radioactive substances A and B, respectively. In adding these variables, we expand the manifold space to incorporate all possible values of a_0 and b_0.

To elaborate, even when we didn't have a_0 and b_0, the model manifold included every prediction of every set of parameters at any time. This can include values of t such that $t < 0$, but we usually consider this situation to be non-physical and focus on $t \geq 0$. Regardless, because time is continuous, there is an infinite number of axes to this model's manifold, with each axis corresponding to possible values of f given different sets of parameter at a specific time. When we include a_0 and b_0, we then must have an axis for every possible combination of a_0, b_0, and t.

As you can imagine, such a manifold would be impossible to plot with any known

technology. Fortunately, for my research, I only need enough of it to discover its information topology/manifold structure/Hasse diagram, which is where cross sections come in. Any plot of the model manifold with fewer than every axis is a cross section of the model manifold [2]. For my purposes, three axes were usually sufficient to gain adequate visualization. This is because the vast majority of models described by an axis would have identical manifold structures and because three-dimensions was usually enough to visualize two-dimensional manifold structures.

For example, consider three of the many possible radioactive models:

$$f(\vec{\theta}) = 1e^{-1\theta_1} + 1e^{-1\theta_2} \tag{1.19}$$

$$f(\vec{\theta}) = 2.5e^{-50\theta_1} + 2.5e^{-50\theta_2} \tag{1.20}$$

$$f(\vec{\theta}) = 10000e^{-\pi\theta_1} + 10000e^{-\pi\theta_2}. \tag{1.21}$$

(We restrict the model to when $a_0 = b_0$ to preserve the symmetry that swapping θ_1 with θ_2 doesn't affect prediction.) Notice that, if we were to build the Hasse diagram of any of these models, we would get the same triangle as before. Upon farther reflection, you should be able to see that this would be the case for the vast majority of models. Therefore, if we pick random initial conditions, we will most likely choose a set of "generic" initial conditions — initial conditions that produce the same manifold structure as the vast majority of others.

There are, however, specific initial conditions for which this isn't the case. It is possible for this to happen due to symmetries, but, in my research, we were primarily concerned about avoiding picking a set of initial conditions such that one of the values was zero. Consider, for example, if I had chosen a_0 to be zero instead of one in Equation 1.19. The resulting model would have been

$$f(\vec{\theta}) = e^{-\theta_2}, \tag{1.22}$$

which is only an edge of the larger cross section that would have been found if we plotted the manifold of Equation 1.19. If we were to try to obtain the manifold structure of the initial model by examining the manifold structure of this specific model, we would have lost its two-dimensional nature and that it was shaped like a triangle.

I display many cross sections of the model manifold throughout this document for visualization purposes. Because I am careful to choose only generic initial conditions, these cross sections are a good representative of the full model manifold and, as such, I will typically refer to them as the model manifold. At the end of the document, I will also discuss briefly how choosing generic initial conditions affects the cross sections of the model manifold when focusing on mass action models, which I will introduce in this next section.

1.9 Introduction to Chemical Reaction Networks and Mass Action Models

We can represent any network of chemical reactions as a directed graph using graph theory. In these networks, each vertex is a stoichiometry—a set of numbers (called stoichiometric coefficients) describing how many molecules of each reactant are consumed by a reaction or how many molecules of each product are produced by a reaction [6]. Each edge, then, represents a reaction that can turn one stoichiometry into another stoichiometry. These edges are weighted, with each associated weight being the reaction rate constant of that reaction.[5] A network structure is any combination of edges and at least one vertex (see Figure 1.9 for an example).

The basic network structures are as follows: cyclic networks, which have the same

[5]These reaction rate constants are the parameters that we search for with data fitting.

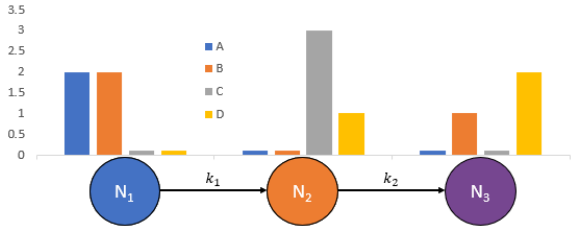


Figure 1.9 Vertices N_1=(2,2,0,0) and N_2=(0,0,3,1) are connected by an edge that represents a reaction with a reaction rate constant of k_1. Vertices N_2=(0,0,3,1) and N_3=(0,1,0,2) are connected by a different edge that represents a reaction with a reaction rate constant of k_2. Another way to represent this network is $2A + 2B \xrightarrow{k_1} 3C + D \xrightarrow{k_2} B + 2D$. In other words, two of species A can react with two of species B, creating three of species C and one of species D, using up the two of species A and the two of species B in the process. At the same moment, three of species C could react with one of species D to create one of species B and another species D, using up the three of species C in the process.

first and last vertex (see Figure 1.10a); branching networks, which have multiple reactions stem from the same stoichiometry (see Figure 1.10b); converging networks, which have multiple reactions producing in the same stoichiometry (basically the opposite of branching); and disconnected networks, which have portions of the network that don't connect in any way (see Figure 1.10c and note that even though the example doesn't show the same species in both networks, it is possible to have the same species in both networks so long as the specific stoichiometries don't repeat). You can then combine these different structures into much more complicated networks (see Figure 1.10d) or leave out any of these structure to create a simple path network (discussed in detail later).

Assuming these reactions follow the law of mass action,[6] we can we can generate a certain system of equations called the mass action model. This model describes how the concentrations of the various species in the reaction(s) change over time. To elaborate, we can assume that the rate of each chemical reaction being modeled is

[6]Note: even if it's not mentioned, we make this assumption for every network in this paper

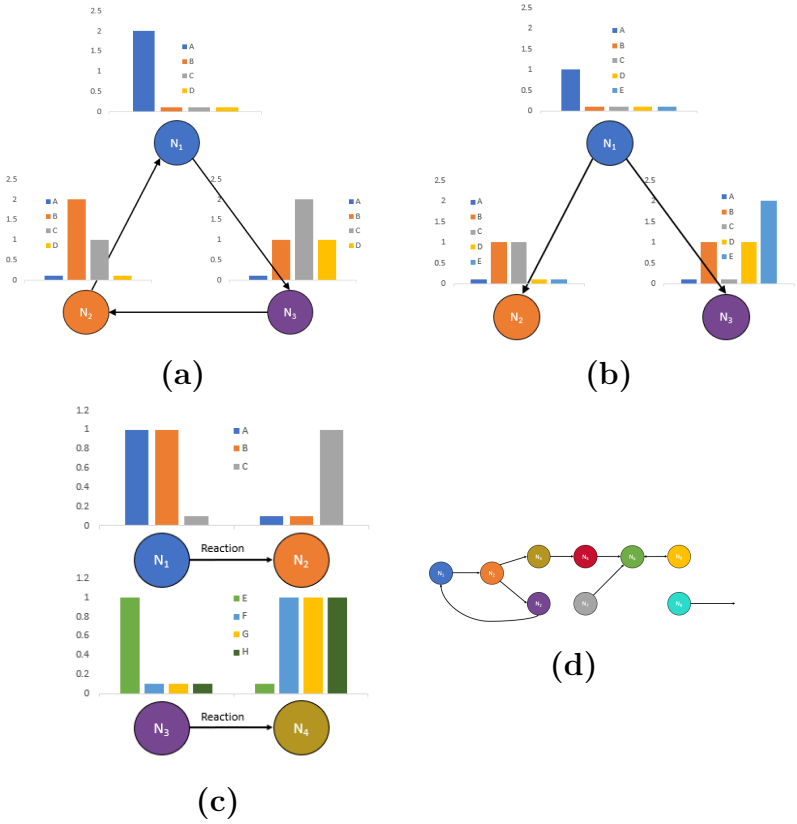


Figure 1.10 (a) Cyclic network structure example: $2A \to B + 2C + D \to 2B + C \to 2A$. (b) Branching network structure example: $A \to B + C$, $A \to B + D + 2E$. (c) Disconnected network structure example: $A + B \to C$, $D \to E + F + G$. (d) Combination of different types of network structure example.

directly proportional to the product of the concentrations of the reactants [7]. For example, let's look at the reaction network $A + B \xrightarrow{k_1} C$, shown graphically in Figure 1.11.

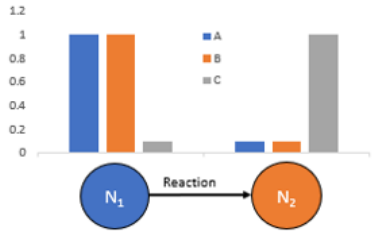


Figure 1.11 $A + B \xrightarrow{k_1} C$

Because we can assume that this network obeys the law of mass action, we know the following equations describe how the concentration of each chemical species changes:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -k_1 ab \tag{1.23}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = -k_1 ab \tag{1.24}$$

$$\frac{\mathrm{d}c}{\mathrm{d}t} = k_1 ab, \tag{1.25}$$

where $a \equiv$ the concentration of A, $b \equiv$ the concentration of B, and $c \equiv$ the concentration of C.[7] Once we have these equations, we have the mass action model associated with this network and can start making predictions. Furthermore, we can start the process of discovering the manifold structure of the specific mass action model.

1.10 The Objective

Knowing the manifold structure of a model is an important step in applying Dr. Transtrum's model reduction methods, but obtaining that knowledge can be a time

[7]Note: for the rest of this paper, the lower case of a capital letter that labels a species is the concentration of that species.

consuming process. At the beginning of this internship, we would start with the chemical network, generate the mass action model, plug it and our initial conditions into various code we had written (found in the appendix), analyze the contour plots, model manifolds, and limits that the parameters were taken to, then finally get the Hasse diagram. Beyond the many step process, most of our methods worked best for models that had one or two parameters, far simpler than the models that we were hoping to study.

Our goal was to explore the relationship between the network structure and the manifold structure. In doing so, we were hoping to find a definitive relationship between the network structure and the manifold structure that would allow us to deduce the manifold structure directly from the network structure. This would greatly streamline the process of applying MBAM to mass action models and hopefully allow us to apply it to arbitrarily large chemical reaction networks.

Prior to my internship, Dr. Transtrum's team had done research on linear compartment models, another type of sloppy model. This research led to their conjecture that linear compartment models always have hypercubic model manifolds. Because of similarities between simple path reaction networks (introduced in Section 2.2) and linear compartment models, we hypothesized that all simple path reaction networks, whether they be linear or nonlinear, would also have hypercubic model manifolds. As such, we decided to start my research with simple path reaction networks and expand from there. To further simplify the problem, we decided to start my focus with one and two parameter models, with some three parameter models as well.

Over the course of the internship, I found that nonlinear mass action models frequently deviated from having hypercubic manifold structures. Furthermore, even linear simple path models didn't always have hypercubic model manifolds. After describing my process of getting the Hasse diagram/manifold structure from the net-

work structure, I will discuss in detail what some of those deviations may be as well as how we might be able to predict their occurrence. I will also propose some tools that may be of use to future researchers in categorizing models according to their manifold structure.

Chapter 2

Method

2.1 Generating Mass Action Models and Predicting Manifold Structure

Earlier, I mentioned that most models don't fill the prediction space; mass action models follow that trend. For example, concentrations cannot be negative, so all predictions will be greater than or equal to zero. Beyond that, the maximum and minimum concentration of each chemical species is limited by the values of the model when the parameters are at their extremes. A simple example of these limits can be seen by examining the model $A \xrightarrow{k_1} B$, displayed graphically in Figure 2.1.

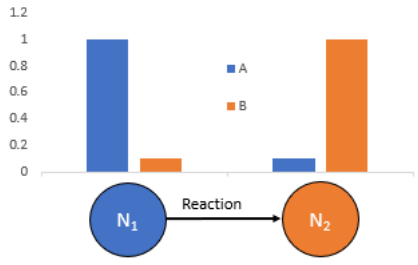


Figure 2.1 $A \xrightarrow{k_1} B$

Using the law of mass action, we can see that this network can be modeled with the following equations:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -k_1 a \tag{2.1}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = k_1 a \tag{2.2}$$

$$a(0) = a_0 \tag{2.3}$$

$$b(0) = b_0 \tag{2.4}$$

From there, it's fairly simple to solve for $a(t)$ and $b(t)$ and get

$$a(t) = a_0 e^{-k_1 t} \tag{2.5}$$

$$b(t) = a_0 - a_0 e^{-k_1 t} + b_0. \tag{2.6}$$

Upon further analysis, you can see that the largest a can ever be is a_0, when $k_1 \to 0$, and the smallest it can ever be is 0, when $k_1 \to \infty$. As for b, it is at its largest, $b = a_0 + b_0$, when $k_1 \to \infty$, and its smallest, $b = b_0$, when $k_1 \to 0$. Therefore, the manifold of the mass action model describing $A \xrightarrow{k_1} B$ has two boundaries: when $k_1 \to 0$ and when $k_1 \to \infty$. Because this is a one-parameter model, there is only one degree of freedom, and the manifold will be one-dimensional. Therefore, the manifold must be a line segment (see Figure 2.2).

All manifolds of one-parameter models are at most one-dimensional. Such models also seem to have at least one boundary—when the single parameter goes to zero. In general, the dimensionality of a model manifold is n, the number of structurally identifiable parameters, so long as none of the reactions are trivial[1]. This means that the manifolds of two-parameter models are two-dimensional (see Figure 2.8 much

[1] An example of a trivial reaction: $\mathrm{A} \xrightarrow{k_1} \mathrm{A}$

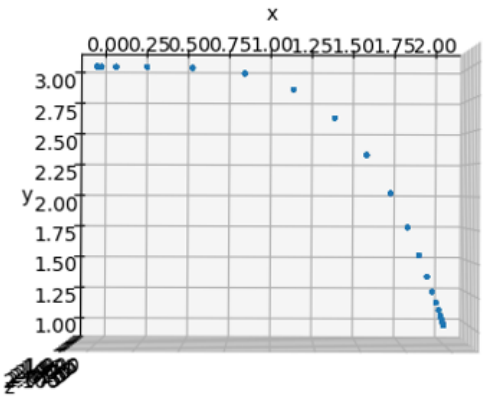
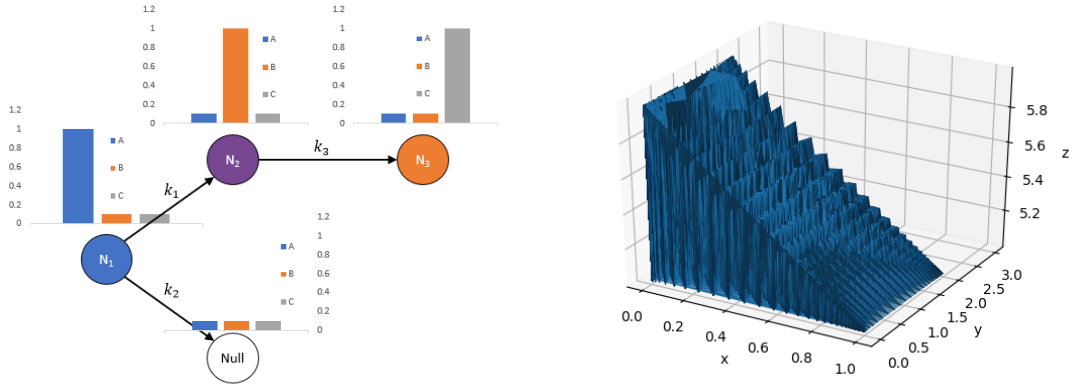


Figure 2.2 Plot of the model manifold of $A \xrightarrow{k_1} B$. The x-axis is the concentration of A and the y-axis is the concentration of B. The unused z-axis is an artifact of my plotting program.

farther down for an example) and the manifolds of three-parameter models are three-dimensional (see Figure 2.3 for an example). Like the one-parameter models, I have also consistently seen that two-parameter models have manifolds with at least two boundaries (when either parameter goes to zero) and three-parameter models have manifolds with at least three boundaries (when any of the parameters go to zero). For reasons not explained here, we expect higher dimensions to follow similar patterns. Therefore, we conjecture and will assume that each manifold has at least as many sides as it has parameters. Note that it is very common for there to be more than this minimum number of boundaries, but getting the precise number of boundaries the manifold can have is much harder and will be discussed throughout this report.

The example shown here was one of the simplest examples that exists. As you can imagine, every node, species, and/or parameter that is added complicates the model further. In order to build a foundation of analyzed networks, we decided to start with networks which have relatively straightforward mass action models.



(a) (b)

Figure 2.3 Example of how three-parameter models have three-dimensional model manifolds. This is a plot of the model manifold of $A \xrightarrow{k_1} B \xrightarrow{k_2} C$, $A \xrightarrow{k_3}$ ___. The x-axis is the concentration of A, the y-axis is the concentration of B, and the z-axis is the concentration of C. Note that conservation of mass is not required of mass action models.

2.2 Linear Simple Path Networks and Hasse Diagrams

The simplest networks are linear simple paths.[2] Neither linear nor nonlinear simple path networks (such as shown in Figures 1.9, 1.11, and 2.1) have any branches, cycles, convergence, or disconnected vertices. Networks that produce linear mass action models (such as shown in Figures 2.1, 2.3a, and 3.11) never have more than one reactant in a reaction. Therefore, a linear simple path network is simply a combination of the two: a network structure that doesn't have any branches, cycles, convergence,

[2]Note: strictly speaking, networks can be simple path and models can be linear, but not vice versa. However, throughout this report I will frequently use terminology that should only apply to a model or a network interchangeably. For example, a simple path model should be interpreted to mean a mass action model that describes a simple path network. Similarly, a network's model manifold should be interpreted as the model manifold of the mass action model that describes the network.

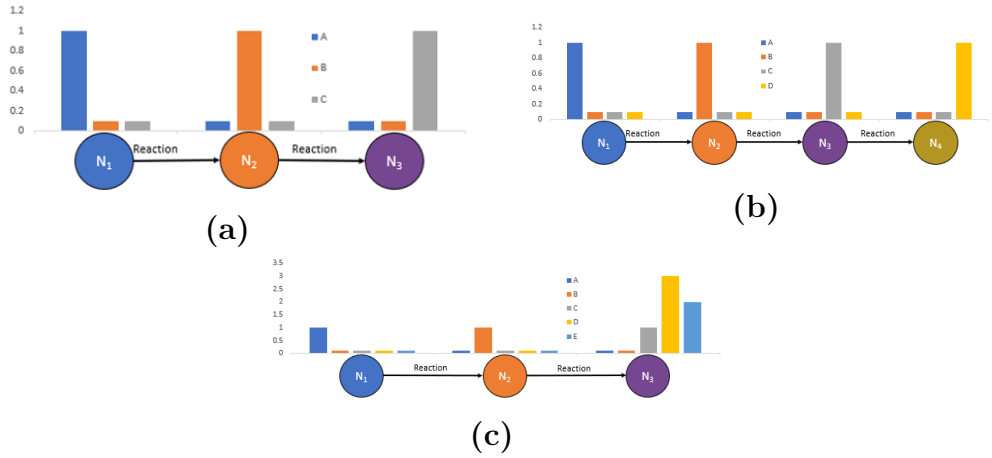


Figure 2.4 (a) Example of a two-parameter linear simple path network: $A \xrightarrow{k_1} B \xrightarrow{k_2} C$. (b) Example of a three-parameter linear simple path network: $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$. (c) Another example of a two-parameter linear simple path network: $A \xrightarrow{k_1} B \xrightarrow{k_2} C + 3D + 2E$. Note that the final product of a linear simple path network can have more than one species in the stoichiometry.

or disconnected vertices, and never has more than one reactant in a reaction (see Figures 2.1, 2.4a, 2.4b, and 2.4c for examples).

When we began this research, we believed that all linear simple path networks would have model manifolds with a manifold structure that was an n-dimensional hypercube, where n was the number of parameters in the model. To elaborate, we believed the following: a one-parameter linear simple path model would always have a manifold that was a line segment (a one-dimensional hypercube); a two-parameter linear simple path model would always have a manifold that was topologically equivalent to a square (a two-dimensional hypercube); a three-parameter linear simple path model would always have a manifold that was topologically equivalent to a cube; etc.

We believed this because, based on the models we had observed, every parameter added to the model would also add two sides to the manifold related to when that parameter was taken to zero and when that parameter was taken to infinity. Adding

a parameter would also add a dimension to the manifold. Therefore, changing a one-parameter model to a two-parameter model would change a line bounded by two endpoints into a square bounded by four line segments. Adding another parameter to get a three-parameter model would change that square bounded by four line segments into a cube bounded by six squares. This could, in theory, continue up to any arbitrary number of parameters you choose. Let's analyze the network $A \xrightarrow{k_1} B \xrightarrow{k_2} C$ (see Figure 2.4a) to see how this works.

First, let's get the mass action model:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -k_1 a \tag{2.7}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = k_1 a - k_2 b \tag{2.8}$$

$$\frac{\mathrm{d}c}{\mathrm{d}t} = k_2 b \tag{2.9}$$

$$a(0) = a_0 \tag{2.10}$$

$$b(0) = b_0 \tag{2.11}$$

$$c(0) = c_0 \tag{2.12}$$

Taking k_1 to zero results in a model equivalent to the model of $B \to C$. From what we saw of the last model, $B \to C$ likely has a manifold that is a line segment. We can check this by running this set of equations through a solver[3] then looking at what happens to the equations as we take k_1 to zero and then separately take k_2 to zero or infinity:

[3]The solver referenced here and in other places throughout this report can be found in the Appendix, under the section "Mathematica Base Model Solver." You can find it on page 94.

$k_1 \to 0$

$$\begin{aligned} a &= a_0 \\ b &= b_0 e^{-tk_2} \\ c &= b_0 + c_0 - b_0 e^{-tk_2} \end{aligned} \tag{2.13}$$

$k_1 \to 0$, then $k_2 \to 0$

$$\begin{aligned} a_f &= a_0 \\ b_f &= b_0 \\ c_f &= c_0 \end{aligned} \tag{2.14}$$

$k_1 \to 0$, then $k_2 \to \infty$

$$\begin{aligned} a_f &= a_0 \\ b_f &= 0 \\ c_f &= b_0 + c_0 \end{aligned} \tag{2.15}$$

where a_f denotes the final concentration of species A, b_f denotes the final concentration of species B, etc.

The same sort of thing happens when we take k_2 to zero first:

$k_2 \to 0$

$$(a, b, c) = (a_0 e^{-tk_1}, a_0 + b_0 - a_0 e^{-tk_1}, c_0) \tag{2.16}$$

$k_2 \to 0$, then $k_1 \to 0$

$$(a_f, b_f, c_f) = (a_0, b_0, c_0) \tag{2.17}$$

$k_2 \to 0$, then $k_1 \to \infty$

$$(a_f, b_f, c_f) = (0, a_0 + b_0, c_0) \tag{2.18}$$

Therefore, not only do we know that this manifold is a two-dimensional shape with at least two boundaries, we know that at least two of the boundaries are line segments. This is important because, in general, not all sides of the model manifold need to be bounded.

In addition, we can see by looking at the above equations that $k_1 \to 0$, $k_2 \to 0$ results in the same zero-dimensional model as $k_2 \to 0$, $k_1 \to 0$. This means that the side of the manifold corresponding to $k_1 \to 0$ meets up with the side of the manifold corresponding to $k_2 \to 0$, specifically at the corner of the manifold where both k_1 and k_2 are zero. The other line segment boundaries (where a parameter is taken to zero first then the other parameter is taken to infinity) don't seem to have sides that meet up with them, but intuitively we know there must be at least one more side to connect those corners.

Trying to keep track of this in your head can get pretty difficult pretty fast, which is why we use the Hasse diagram described earlier to help us. The collection of reduced models form a partially ordered set (a poset), and the Hasse diagram is a tool to organize the elements as a graded poset.

With the Hasse diagram, we can see what we have so far mapped out in Figure 2.5. Note that there are two one-dimensional models (the circles) and three zero-dimensional models (the oval-like shapes), like we've seen so far. Beyond that, note how both one-dimensional models connect at ZD1. This represents the two line segments connecting at the point (a_0, b_0, c_0).

Now consider when we begin by taking the parameters to infinity first, instead of taking the parameters to zero first. Starting with $k_1 \to \infty$ leaves a network which changes just like the network $B \xrightarrow{k_2} C$, which we have just found to be a line segment. However, this is a different line segment than before because $a \to 0$ as we take k_1 to infinity:

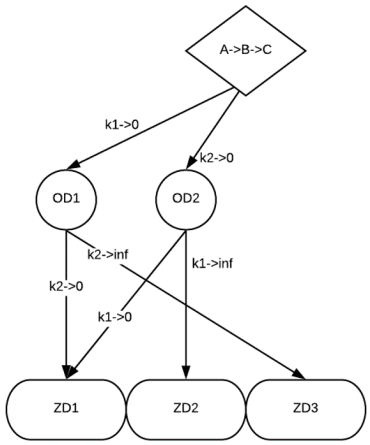


Figure 2.5 Incomplete Hasse Diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} C$

$k_1 \to \infty$

$$(a,b,c) = \tag{2.19}$$
$$(0, (a_0 + b_0)e^{-tk_2}, a_0 + b_0 + c_0 - a_0e^{-tk_2} - b_0e^{-tk_2})$$

$k_1 \to \infty$, $k_2 \to 0$

$$(a, b, c) = (0, a_0 + b_0, c_0) \tag{2.20}$$

$k_1 \to \infty$, $k_2 \to \infty$

$$(a, b, c) = (0, 0, a_0 + b_0 + c_0) \tag{2.21}$$

With this new limit taken, Figure 2.6 has the most up to date Hasse diagram. Notice how we now have two sides connected to ZD3 as well, but we left a new corner, ZD4, hanging.

Now for the last side:

$k_2 \to \infty$

$$(a, b, c) = (a_0e^{-tk_1}, 0, a_0 + b_0 + c_0 - a_0e^{-tk_1}) \tag{2.22}$$

$k_2 \to \infty$, $k_1 \to 0$

$$(a, b, c) = (a_0, 0, b_0 + c_0) \tag{2.23}$$

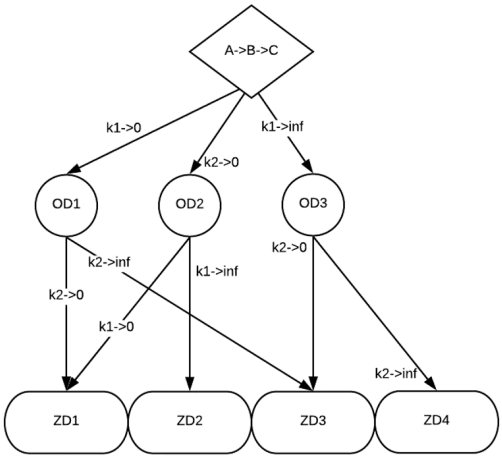


Figure 2.6 Almost Finished Hasse Diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} C$

$k_2 \to \infty$, $k_1 \to \infty$

$$(a, b, c) = (0, 0, a_0 + b_0 + c_0) \tag{2.24}$$

And with that, we've completed our Hasse diagram. See Figure 2.7 and note that Equations 2.14 and 2.17 correspond to node ZD1, Equations 2.15 and 2.23 correspond to node ZD2, Equations 2.18 and 2.20 correspond to node ZD3, and Equations 2.21 and 2.24 correspond to node ZD4. Furthermore, OD1-OD4 correspond to the one dimensional models discussed above.

The reasons we know this is a complete Hasse diagram are beyond the scope of this paper, but we can still gain an intuitive sense of why by analyzing what sort of shape this Hasse diagram predicts. It predicts that there will be a two-dimensional surface bounded by four line segments, corresponding to the two parameter model and the four limits to which its parameters could be taken. These line segments each share an endpoint with two different line segments, with each end point representing when both parameters are at an extreme. With some further thought, it becomes clear that this Hasse diagram is predicting the model manifold will topologically be

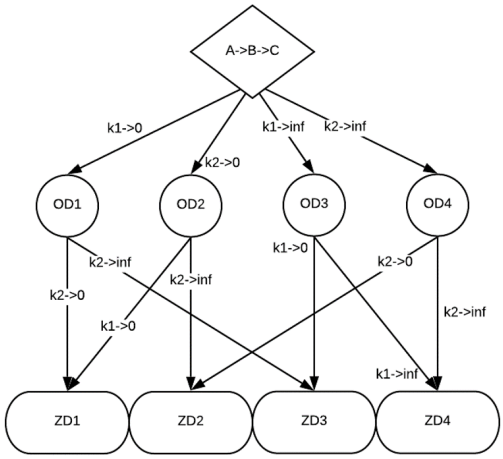


Figure 2.7 Completed Hasse Diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} C$

a square,[4] which is exactly what we see when we plot the manifold (see Figure 2.8).

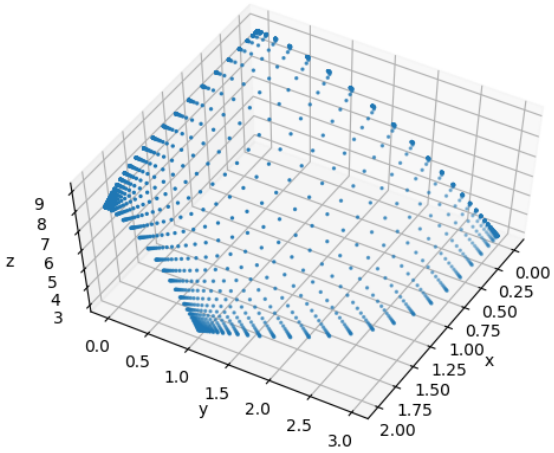


Figure 2.8 The network $A \xrightarrow{k_1} B \xrightarrow{k_2} C$ produces a manifold that, topologically, is a square, just like the Hasse diagram predicted.

[4]Note: whenever I refer to a shape in this paper, I am referring to it in the topological sense. Therefore, a square is a two-dimensional shape with 4 bounded sides and 4 bounded corners connected to each other in a certain way; a cube is a three-dimensional shape with 6 bounded faces, 12 bounded sides, and 8 bounded corners that are all connected in a certain way; a triangle has 3 bounded sides and 3 bounded corners connected together in a certain way; etc. Whether or not the sides are the same length, or even straight, doesn't matter topologically.

We can use this same process to find the manifold of a three-parameter model, such as the one describing the network in Figure 2.4b. Taking any of the parameters to zero or to infinity leaves a two-parameter model describing a network that changes just like $A \to B \to C$. Each of these two-parameter models represents a unique side of the model manifold. Because there are three different ways to take a parameter to zero and three different ways to take a parameter to infinity, there must be six different sides of this manifold. Beyond that, each of those sides is represented by a model nearly identical to $A \to B \to C$, so they must all be squares. With all this in mind, the manifold of the model describing $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$ must be a cube, which is exactly what we see in Figure 2.9's plot of the manifold.

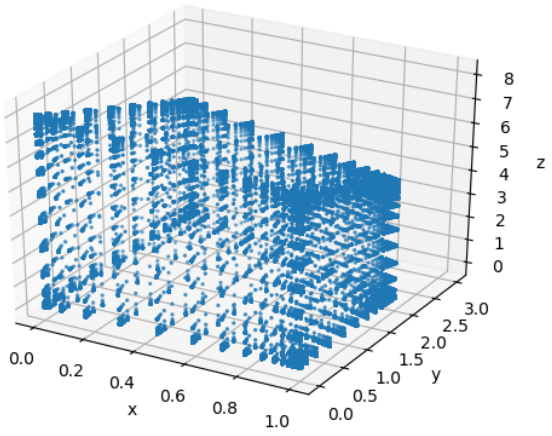


Figure 2.9 Plot of the model manifold of the mass action model that describes $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$

You can work this out for yourself if you desire. A word of warning, though: the Hasse diagram for this is enormous. It is the only three-parameter Hasse diagram that I've worked out in it's entirety, and you can see it for yourself in Figure 2.10.

Note that the example in Figure 2.4c also has a hypercubic manifold structure. You can see this more easily by looking at the differential equations describing the network $A \xrightarrow{k_1} B \xrightarrow{k_2} C + 3D + 2E$ (see Equations 2.25, 2.26, 2.27, 2.28, and 2.29

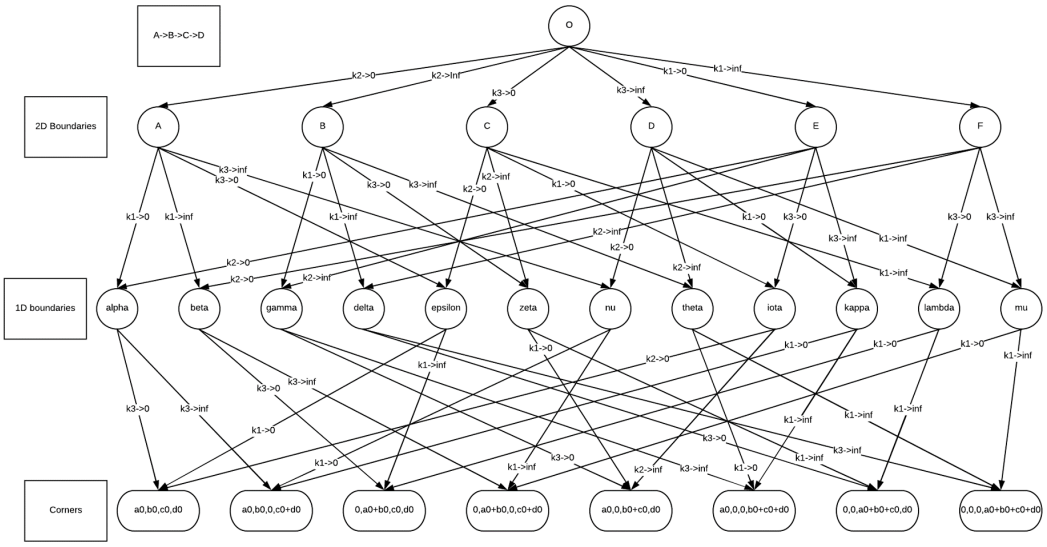


Figure 2.10 Completed Hasse Diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$

immediately after this paragraph) and comparing them to the differential equations used to describe the network $A \xrightarrow{k_1} B \xrightarrow{k_2} C$ (see Equations 2.7, 2.8, and 2.9).

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -k_1 a \tag{2.25}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = k_1 a - k_2 b \tag{2.26}$$

$$\frac{\mathrm{d}c}{\mathrm{d}t} = k_2 b \tag{2.27}$$

$$\frac{\mathrm{d}d}{\mathrm{d}t} = 3k_2 b \tag{2.28}$$

$$\frac{\mathrm{d}e}{\mathrm{d}t} = 2k_2 b \tag{2.29}$$

Comparing the two sets of differential equations, you can see that a, b, and c, all behave exactly the same as before. The only difference is that we now have to worry about d and e. However, there are still only the same number of ways to take unique limits and get unique one-dimensional models. Then, from those one-dimensional models, taking the remaining parameter to zero or infinity results in

zero-dimensional models that are connected to the one-dimensional models in the same way that the zero-dimensional models were connected to the one-dimensional models of the network $A \xrightarrow{k_1} B \xrightarrow{k_2} C$. In other words, a Hasse diagram representing the manifold structure of $A \xrightarrow{k_1} B \xrightarrow{k_2} C+3D+2E$ would be indistinguishable from the Hasse diagram representing the manifold structure of $A \xrightarrow{k_1} B \xrightarrow{k_2} C$. You can see this for yourself by looking at the equations for said one-dimensional and zero-dimensional models in Figure 2.11.

$$
\begin{aligned}
&k[2]\to 0 \text{ gets } \left\{a0\, e^{-tk[1]},\ a0+b0-a0\, e^{-tk[1]},\ c0,\ d0,\ e0\right\}\\
&k[2]\to 0 \text{ and } k[1]\to 0 \text{ gets } \{a0,\ b0,\ c0,\ d0,\ e0\}\\
&k[2]\to 0 \text{ and } k[1]\to \infty \text{ gets } \{0,\ a0+b0,\ c0,\ d0,\ e0\}\\
&k[2]\to \infty \text{ gets } \left\{a0\, e^{-tk[1]},\ 0,\ a0+b0+c0-a0\, e^{-tk[1]},\right.\\
&\quad \left. 3\,a0+3\,b0+d0-3\,a0\, e^{-tk[1]},\ 2\,a0+2\,b0-2\,a0\, e^{-tk[1]}+e0\right\}\\
&k[2]\to \infty \text{ and } k[1]\to 0 \text{ gets } \{a0,\ 0,\ b0+c0,\ 3\,b0+d0,\ 2\,b0+e0\}\\
&k[2]\to \infty \text{ and } k[1]\to \infty \text{ gets } \{0,\ 0,\ a0+b0+c0,\ 3\,a0+3\,b0+d0,\ 2\,a0+2\,b0+e0\}\\
&k[1]\to 0 \text{ gets }\\
&\quad \left\{a0,\ b0\, e^{-tk[2]},\ b0+c0-b0\, e^{-tk[2]},\ 3\,b0+d0-3\,b0\, e^{-tk[2]},\ 2\,b0-2\,b0\, e^{-tk[2]}+e0\right\}\\
&k[1]\to 0 \text{ and } k[2]\to 0 \text{ gets } \{a0,\ b0,\ c0,\ d0,\ e0\}\\
&k[1]\to 0 \text{ and } k[2]\to \infty \text{ gets } \{a0,\ 0,\ b0+c0,\ 3\,b0+d0,\ 2\,b0+e0\}\\
&k[1]\to \infty \text{ gets } \left\{0,\ a0\, e^{-tk[2]}+b0\, e^{-tk[2]},\ a0+b0+c0-a0\, e^{-tk[2]}-b0\, e^{-tk[2]},\right.\\
&\quad \left. 3\,a0+3\,b0+d0-3\,a0\, e^{-tk[2]}-3\,b0\, e^{-tk[2]},\ 2\,a0+2\,b0-2\,a0\, e^{-tk[2]}-2\,b0\, e^{-tk[2]}+e0\right\}\\
&k[1]\to \infty \text{ and } k[2]\to 0 \text{ gets } \{0,\ a0+b0,\ c0,\ d0,\ e0\}\\
&k[1]\to \infty \text{ and } k[2]\to \infty \text{ gets } \{0,\ 0,\ a0+b0+c0,\ 3\,a0+3\,b0+d0,\ 2\,a0+2\,b0+e0\}
\end{aligned}
$$

Figure 2.11 Equations for the one- and zero-dimensional models of $A \xrightarrow{k_1} B \xrightarrow{k_2} C+3D+2E$

The similarities between $A \xrightarrow{k_1} B \xrightarrow{k_2} C$ and models like $A \xrightarrow{k_1} B \xrightarrow{k_2} C+3D+2E$ have led me to develop a tool for separating networks according to which ones have the possibility of producing a new, unique manifold, rather than a manifold that is nearly identical to another. I'll explain this tool in the next section.

Chapter 3

Results

3.1 Independent Species: A Tool for Grouping Similar Networks

There is still much to explore with the idea of independent species, and, like most concepts in this report, it is not guaranteed to always work. Nevertheless, I have found it to be a useful tool in finding the manifold structure/Hasse diagrams of mass action models.

In many models, there are chemical species which, whenever you find one of it in the network, there's a certain other species which shows up too. To help me describe this, I've started using what I call the coefficient vector, $\vec{c}_A$,

$$\vec{c}_A = (s_1, s_2, ..., s_M), \tag{3.1}$$

where s_i is the stoichiometric coefficient of species A in the ith node of the network and M is the number of nodes that are in the network

To understand this better, let's analyze the nonlinear simple path network $A + B + C \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + 2C + E$. To get the coefficient vector of A, $\vec{c}_A$, start by taking

the stoichiometric coefficient of A in the first node and put it in the first slot of your vector. In this example, that number would be "1":

$$\vec{c}_A = (1, \_, \_) \tag{3.2}$$

Next, get the stoichiometric coefficient of A in the second node and put that number in the second slot of your coefficient vector, followed by putting the stoichiometric coefficient of A in the third node into the third slot of the vector:

$$\vec{c}_A = (1, 0, 2) \tag{3.3}$$

Repeat for each species in the network:

$$\vec{c}_A = (1, 0, 2) \tag{3.4}$$

$$\vec{c}_B = (1, 0, 0) \tag{3.5}$$

$$\vec{c}_C = (1, 0, 2) \tag{3.6}$$

$$\vec{c}_D = (0, 3, 0) \tag{3.7}$$

$$\vec{c}_E = (0, 0, 1) \tag{3.8}$$

Once you have the vectors, it's easier to see that A and C are always paired together in these reactions; after all, they have the exact same coefficient vector. It is also possible to have something like A and $2C$ or A and $12C$ paired together if the coefficient vectors are proportional to each other (e.g., if $\vec{c}_C = (12, 0, 24)$ instead of $\vec{c}_C = (1, 0, 2)$).

To summarize this, **I conjecture that if $c_A \propto c_B$ for some species A and B in a mass action model, then a new model that is identical to the original model except that either A or B has been removed will have the same**

manifold structure as the original model. In other words, you can often, if not always, treat pairs of species that have linearly dependent coefficient vectors as a single species and still get the same manifold structure. Equations derived from that model would have different reaction rates, and the final concentrations would be different, but the overall structure would be the same. For now, I call this the **Principle of Independent Species**.

Applying this principle to the example given, the manifold structure of $A + B + C \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + 2C + E$ should be the same as the manifold structure if we were to remove either A or C from the model:

$$B + C \xrightarrow{k_1} 3D \xrightarrow{k_2} 2C + E \tag{3.9}$$

$$A + B \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + E. \tag{3.10}$$

Finding the new coefficient vectors sometimes reveals more species with proportional coefficient vectors, meaning the model can then be condensed again. Note that the vectors need to be proportional to at least one specific vector, not simply a linear combination of multiple vectors, for it to be dependent in this context.

Further applying the principle of independent species, adding a species S which has a coefficient vector proportional to one already in the network also shouldn't change the manifold structure. For example, all of the following networks should have the same manifold structure as the example we've been discussing:

$$A + B + C + S \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + 2C + E + 2S \tag{3.11}$$

$$A + B + C \xrightarrow{k_1} 3D + 3S \xrightarrow{k_2} 2A + 2C + E \tag{3.12}$$

$$A + B + C \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + 2C + E + 7S \tag{3.13}$$

In order to have a different network, you would need to add a species in a way that its coefficient vector isn't proportional to any that currently existed. The following, for example, cannot be reduced to the prior example via the principle of independent species:

$$A + B + C + S \xrightarrow{k_1} 3D \xrightarrow{k_2} 2A + 2C + E + 7S \tag{3.14}$$

$$A + B + C \xrightarrow{k_1} 3D + S \xrightarrow{k_2} 2A + 2C + E + 2S \tag{3.15}$$

Checking if there are any dependent species will be more relevant when we talk about nonlinear models later on, but, for now, it is part of the justification that $A \xrightarrow{k_1} B \xrightarrow{k_2} C + 3D + 2E$ (see Figure 2.4c), along with similar networks, behave the same as $A \xrightarrow{k_1} B \xrightarrow{k_2} C$. After all, it has the following coefficient vectors:

$$\vec{c}_A = (1, 0, 0) \tag{3.16}$$

$$\vec{c}_B = (0, 1, 0) \tag{3.17}$$

$$\vec{c}_C = (0, 0, 1) \tag{3.18}$$

$$\vec{c}_D = (0, 0, 3) \tag{3.19}$$

$$\vec{c}_E = (0, 0, 2) \tag{3.20}$$

Note that $\vec{c}_C$, $\vec{c}_D$, and $\vec{c}_E$ are each scalar multiples of each other. As such, by the principle of independent species, this network can be reduced down to the simple path network $A \xrightarrow{k_1} B \xrightarrow{k_2} C$, which we know has a hypercubic model manifold. As such, **I conjecture that every linear simple path model which doesn't have a reactant as a final product will have a hypercubic model manifold**. After all, adding new species to the product of the reaction and only the product of the reaction simply adds another species that has a coefficient vector proportional to the

coefficient vector of the last species in the reaction.[1]

This opens another possibility, though: networks where a reactant *is* in the final product. For example, let's talk about the linear simple path network $A \xrightarrow{k_1} A + B$.

3.2 Unbounded Manifolds

First, an important clarification. Because mass action models are simply approximations of a larger system, they don't need to conserve mass. As such, models like $\_\_\_ \xrightarrow{k_1} A$ or $A \xrightarrow{k_1} \_\_\_$ are legitimate mass action models.[2]

This leads us to another discovery: **Not all linear simple path models have hypercubic model manifolds**. To see why this is, let's look closer at the model $A \xrightarrow{k_1} A+B$. As usual, we can start by looking at the differential equations describing it:

$$(\frac{\mathrm{d}a}{\mathrm{d}t}, \frac{\mathrm{d}b}{\mathrm{d}t}) = (0, k_1 a) \tag{3.21}$$

which, when solved, produces

$$(a, b) = (0, b_0 + a_0 t k_1). \tag{3.22}$$

From there, we can see what happens when we take k_1 to zero or infinity:

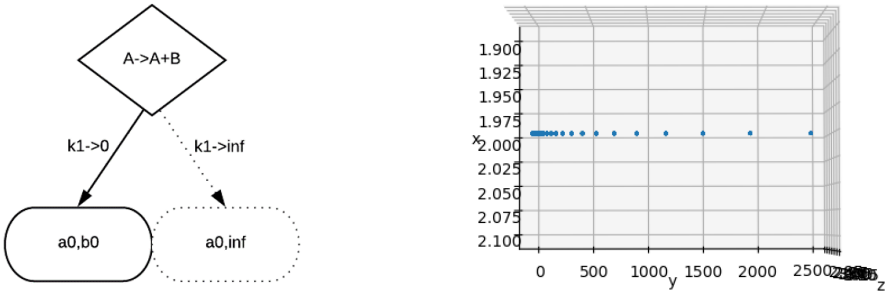
[1]Note that this is the specific type of network mentioned earlier that seems to consistently behave like linear compartment models.

[2]Here's an example of how a reaction can be well approximated by $A \xrightarrow{k_1} A + B$: if species A reacts with water in such a way that it produces species A again and also species B, it might seem like we would need an additional species to represent the water, say, species C. However, if this reaction was happening in the middle of the ocean with only a few molecules of A at a time, the concentration of C is approximately constant and can be factored into k_1, leaving behind the model $A \xrightarrow{k_1} A + B$.

$$k_1 \to 0 : \ (a_f, b_f) = (a_0, b_0) \tag{3.23}$$

$$k_1 \to \infty : \ (a_f, b_f) = (a_0, \infty) \tag{3.24}$$

Rather than having two finite points as boundaries, we have a finite point, $(a_f, b_f) = (a_0, b_0)$, and a point at infinity, $(a_f, b_f) = (a_0, \infty)$. Therefore, this manifold is a ray, rather than a line segment. See Figure 3.1 for the Hasse diagram and a plot of this model's manifold.



(a) (b)

Figure 3.1 (a) The Hasse diagram of $A \xrightarrow{k_1} A + B$ predicts a ray: a one-dimensional shape with one side bounded (the solid line) and one side unbounded (the dotted line) (b) A plot of $A \xrightarrow{k_1} A+B$'s model manifold. Notice that, though a (portrayed on the x-axis, up and down) stays at 2, b does in fact go off to infinity, implying a ray.

A similar thing can happen with models that have more parameters. For example, $A \to B \to B+C$ has one of its four sides at infinity along with two of its four corners (see Figure 3.2 for a plot of the manifold). Adding another parameter and following the pattern of making the last reactant a product of the reaction, we get the model $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} C + D$. Though my plotting program can't get a plot of this, what we have observed strongly implies that this model manifold is shaped like a cube that has one of its faces at infinity. We expect this pattern to continue for all linear simple

path network structures that have the only reactant in the final product be the final reactant.

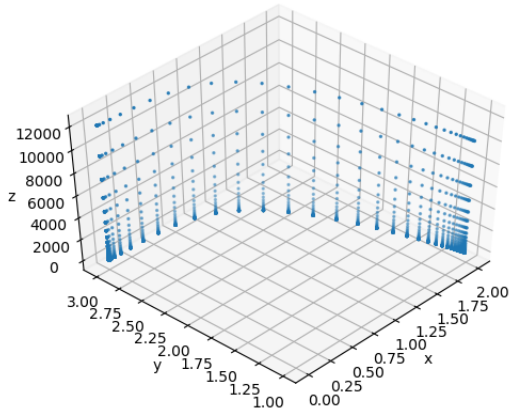


Figure 3.2 A plot of $A \xrightarrow{k_1} B \xrightarrow{k_2} B + C$'s model manifold. Notice that, though a and b remain finite, c (displayed on the z-axis) goes to infinity.

Especially when it comes to nonlinear simple path models, this is something to watch for whenever you are trying to derive the Hasse diagram from the network: **anytime the reactants of a specific reaction are produced at the same or greater rate than they are consumed by that specific reaction, I conjecture that one of the sides of the model manifold will be unbounded**. Note that producing less of the species than is consumed doesn't lead to unbounded manifolds. For example, though it may look like it would be unbounded, the model manifold of $2A \rightarrow A + B$ is a line segment.

This brings me to another of the tools I've developed to study the connection between the model manifold and the network. It's called the equivalent model.

3.3 The Equivalent Model: Another Tool for Grouping Similar Networks

The equivalent model is equivalent in the sense that, though it has different reaction rates, the steady states reached after the reaction has run its course are usually the same steady states as the more complex, original model. To find it, you start by translating the reaction into a set of disconnected reactions. For example, let's consider the network $2A + B \xrightarrow{k_1} A + 2B \xrightarrow{k_2} 2B + C$ (see Figure 3.3a). After breaking it into a set of disconnected reactions, it becomes

$$2A + B \xrightarrow{k_1} A + 2B \tag{3.25}$$

$$A + 2B \xrightarrow{k_2} 2B + C \tag{3.26}$$

(see Figure 3.3b).

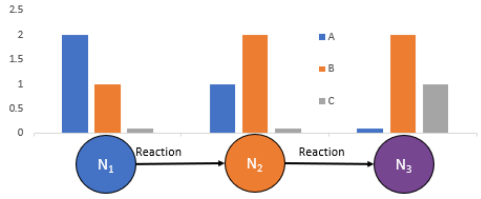
From there, the next steps are to find what the net result of each reaction is then rewrite the equations to reflect the net result of each reaction. In this example, the first reaction does the equivalent of turning A into B and the second reaction does the equivalent of turning A into C. Rewriting the reactions to account for this gives the following:

$$A \xrightarrow{k_1} B \tag{3.27}$$

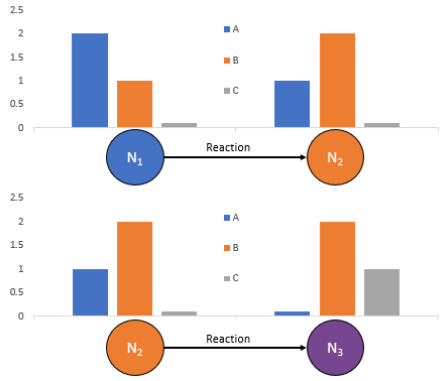
$$A \xrightarrow{k_2} C \tag{3.28}$$

(see Figure 3.3c).

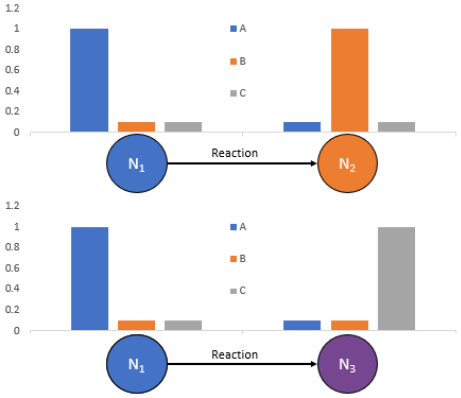
Seeing this equivalent network this way reveals that what seemed to be a simple path model actually behaves more like a branching model, with two reactions coming



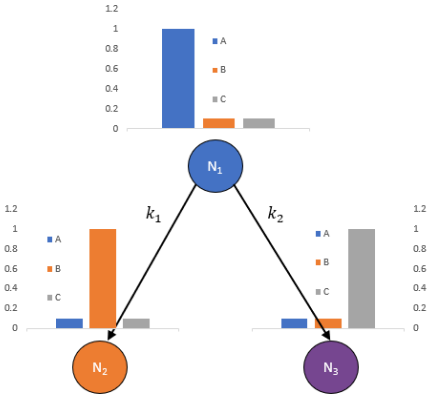
(a) $2A + B \xrightarrow{k_1} A + 2B \xrightarrow{k_2} 2B + C$



(b) $2A + B \xrightarrow{k_1} A + 2B,\ A + 2B \xrightarrow{k_2} 2B + C$



(c) $2A + B \xrightarrow{k_1} A + 2B,\ A + 2B \xrightarrow{k_2} 2B + C$ Equivalent Network



(d) $A \xrightarrow{k_1} B,\ A \xrightarrow{k_2} C$ Branching Network

Figure 3.3 Four models that all have the same manifold structure to illustrate how the equivalent model works.

from the same node (see Figure 3.3d). Note that this is different from any of the simple path networks we have seen so far. We'll talk about why later, but it turns out this deviant behavior leads this simple path network to have a triangular model manifold rather than the previously expected square (unbounded or not) model manifold.

Rounding back to unbounded models, the equivalent model of $A \xrightarrow{k_1} A + B$ is the following:

$$\_\_\_ \xrightarrow{k_1} B \tag{3.29}$$

In other words, it becomes very easy to see that something is being generated from nothing. The equivalent model is especially helpful for unveiling this behavior in more complicated models such $4A + 6B + C \xrightarrow{k_1} 2A + D + C \xrightarrow{k_2} 3A + D + 2C$, which has the following equivalent model:

$$2A + 6B \xrightarrow{k_1} D \tag{3.30}$$

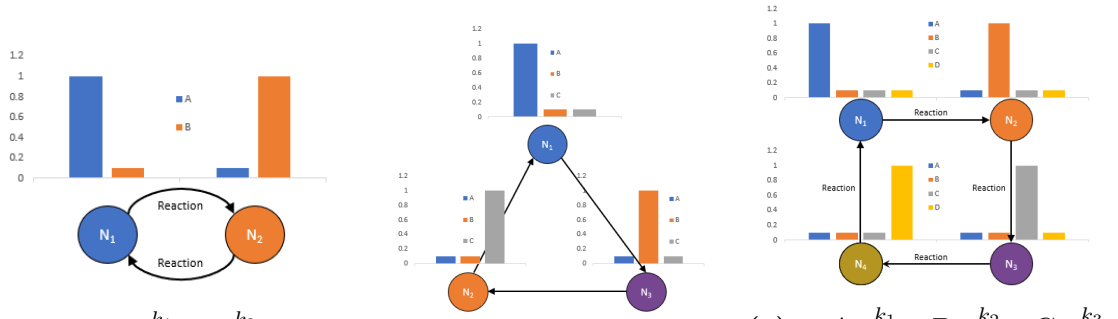
$$\_\_\_ \xrightarrow{k_2} A + C \tag{3.31}$$

There are times, though, when the equivalent model doesn't predict that something will come from nothing, even when it does happen. The most prominent examples of this typically involve cycles.

3.4 Linear Cycles and Cyclic Behavior in Simple Path Models

Before talking about what cycles have to do with simple path models, lets talk about cycles in general. First off, a reminder: cyclic networks are networks which have the same first and last vertex. Just as linear simple path models are the most straightforward simple path models, linear cyclic models are the most straightforward cyclic models. The three linear cycles with the fewest parameters are shown in Figure 3.4.

Note that nonlinear cyclic models, such as $A + B \xrightarrow{k_1} C \xrightarrow{k_2} A + B$ or the one in Figure 1.10a, will not be discussed in this report. Even ignoring nonlinear cycles, though, there are some interesting things that happen to the model manifold when cycles become involved. To start, we'll see how the two-parameter cycle $A \xrightarrow{k_1} B \xrightarrow{k_2} A$ ends up with a triangular model manifold rather than a manifold with four sides. Here is the network's mass action model:



(a) $A \xrightarrow{k_1} B \xrightarrow{k_2} A$ **(b)** $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} A$ **(c)** $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D \xrightarrow{k_4} A$

Figure 3.4 Linear Cycles

$$(\tfrac{\mathrm{d}a}{\mathrm{d}t}, \tfrac{\mathrm{d}b}{\mathrm{d}t}) = (-k_1 a + k_2 b, k_1 a - k_2 b) \tag{3.32}$$

The solution to these equations is a fairly complex set of equations for a and b.[3] Once we have them, taking k_1 or k_2 to zero results in the linear simple path networks $B \xrightarrow{k_2} A$ or $A \xrightarrow{k_1} B$, respectively, so we know that there exists at least two sides to the model manifold which are line segments. However, problems start happening when you take either parameter to infinity:

$$k_1 \to \infty : (a, b) = (0, a_0 + b_0) \tag{3.35}$$

$$k_2 \to \infty : (a, b) = (a_0 + b_0, 0) \tag{3.36}$$

In both cases, taking the parameter to infinity jumps the model straight to a zero-dimensional model, rather than passing through a one-dimensional model like

[3]

$$a = \frac{a_0 e^{t(-k_1-k_2)} k_1 + a_0 k_2 + b_0 k_2 - b_0 e^{t(-k_1-k_2)} k_2}{k_1 + k_2} \tag{3.33}$$

$$b = -\frac{-a_0 k_1 - b_0 k_1 + a_0 e^{t(-k_1-k_2)} k_1 - b_0 e^{t(-k_1-k_2)} k_2}{k_1 + k_2} \tag{3.34}$$

normal, which is like jumping straight to a corner of the model manifold instead of an edge. After all, taking $k_1 \to \infty$ makes the network pretty much identical to $B \xrightarrow{k_2} B$: every tiny bit of B that turns into A is immediately turned back into B. The same sort of thing happens when you take $k_2 \to \infty$, and the remaining model is essentially $A \xrightarrow{k_1} A$. In other words, taking either parameter to infinity practically wipes out the ability of the other parameter to change predictions, reducing what would be a side of the manifold to a single point.

This isn't like before, where there were four sides with one or more at infinity. Instead, this manifold simply doesn't have four sides, or at least not in the same way as the linear simple path networks we've been looking at. However, we know that, topologically, there needs to be at least one more side to connect the corners (see Figure 3.5 for the partial Hasse diagram and Figure 3.6 for the reduced models so far). We'll just need to find the reduced model for the remaining side another way, and in this case that other way involves ratios.

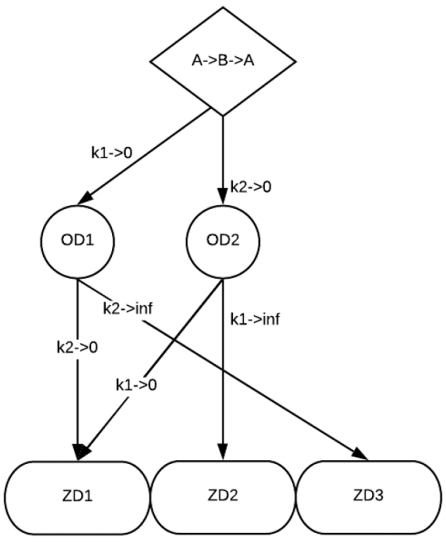


Figure 3.5 Incomplete Hasse diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} A$

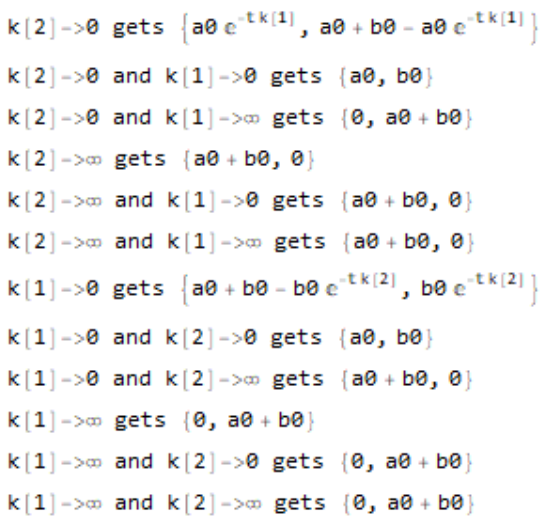


Figure 3.6 Taking parameters to zero or infinity in the model $A \xrightarrow{k_1} B \xrightarrow{k_2} A$.

If we reparameterize the model so that $k = k_1$ and $K = k_2/k_1$, we end up with this new set of equations:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = k(-a + Kb) \tag{3.37}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = k(a - Kb). \tag{3.38}$$

The solved version of these equations is included in the footnote below.[4]Now, when we take k to infinity, we end up with a model that still depends on the parameter K. In other words, the final concentrations depend on the ratio between the two initial parameters:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = (a_0 + b_0) * \frac{K}{1+K} \tag{3.39}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = (a_0 + b_0) * \frac{1}{1+K} \tag{3.40}$$

[4]

$$a = -\frac{-a_0 e^{tk(-1-K)} - a_0 K - b_0 K + b_0 e^{tk(-1-K)} K}{1+K} \tag{3.41}$$

$$b = \frac{a_0 + b_0 - a_0 e^{tk(-1-K)} + b_0 e^{tk(-1-K)} K}{1+K} \tag{3.42}$$

Taking K to infinity or zero shows that this side connects with the other sides at their endpoints:

$k \to \infty$, then $K \to 0$

$$a_f = 0 \tag{3.43}$$

$$b_f = a_0 + b_0 \tag{3.44}$$

$k_1 \to 0$, then $K \to \infty$

$$a_f = a_0 + b_0 \tag{3.45}$$

$$b_f = 0 \tag{3.46}$$

Therefore, we predict that the manifold will have three one-dimensional boundaries, two of which appear when a parameter goes to zero, and a third which appears when the two parameters go to infinity in ratio with each other. Those three sides are then connected via one of three zero-dimensional models, the final states when both of the parameters are at their limits (you can see these predicted connections in Figure 3.7). In other words, we predict we will see a triangle, which is, in fact, what we see in Figure 3.8.

The $A \xrightarrow{k_1} B \xrightarrow{k_2} A$ network has provided us an example of how cycles can lead to losing a side of the manifold and how that side disappears because taking parameters to infinity feeds a node into itself. However, sometimes cycles can lead to an extra side appearing, such as with the network $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} A$ (shown in Figure 3.4b).

Taking k_1, k_2, or k_3 to zero leaves behind a two-parameter network identical to a linear simple path network with no reactant in the product, so there will be three sides of this manifold that are squares (with each square side corresponding to one of the two-parameter models left behind when one of the three parameters goes to

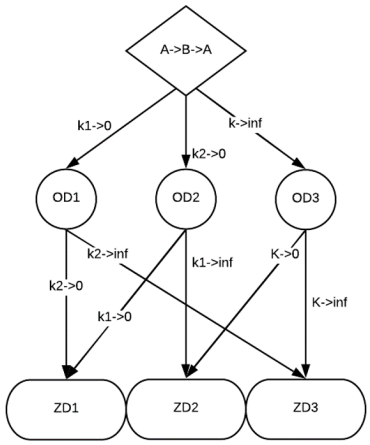


Figure 3.7 Complete Hasse diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} A$

zero first). Meanwhile, taking k_1, k_2, or k_3 to infinity results in a two-parameter linear cycle, the model we just studied. As such, there will be three sides of this manifold that are triangles, each corresponding with the model resulting from taking a parameter to infinity. This means there are six sides predicted so far. This might seem to be all of the sides, but, if you were to actually fill out the Hasse diagram, you would discover that there must be at least one other side to allow everything to fit together topologically. It turns out that we can find the missing side using ratios again.

In this case, instead of taking one parameter to infinity, take all of the parameters to infinity in ratio to each other. To do this, reparameterize the model so that $k = k_1$, $K_2 = k_2/k_1$, and $K_3 = k_3/k_1$. Once k is taken to infinity, K_2 and K_3 can be taken to zero individually to match up with two of the remaining sides, or taken to infinity in ratio to each other to match up with the final side. This leaves behind a strange, three-dimensional shape that is made up of three squares and four triangles, for seven sides total.

Now that we've seen how cycles can lead to more or fewer sides than might be

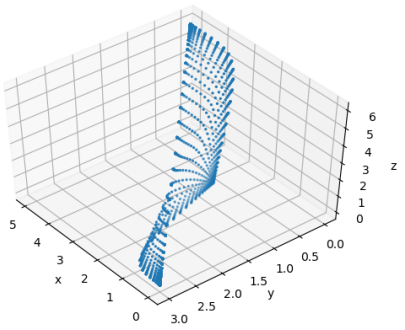


Figure 3.8 Model manifold of $A \xrightarrow{k_1} B \xrightarrow{k_2} A$.[a]

[a] In order the break a time symmetry and an initial condition symmetry in this model, I had the x-axis be a at one time with one set of initial conditions, the y-axis be b at another time with a different set of initial conditions, and finally the z-axis be a at yet another time with a third set of initial conditions. The reasoning and method behind doing this is beyond the scope of this paper; it is simply important to know that it produced the expected triangular model manifold. The program used to plot this and other two-parameter model manifolds can be found in the appendix under the section "Two-Parameter Plotting Code."

expected, let's see how this knowledge can be applied to simple path models. The most straightforward example of how they relate can be demonstrated by the network $A \xrightarrow{k_1} B \xrightarrow{k_2} A + C$. Though this seems very much like the cycle $A \xrightarrow{k_1} B \xrightarrow{k_2} A$, the addition of C makes it a simple path network (see Figure 3.9). Some interesting behavior results because this network behaves like a linear simple path network in some ways and like a linear cycle in other ways.

To start our analysis, taking either k_1 or k_2 to zero results in a linear simple path model with no reactants in the final product. Therefore, unlike in the network $A \xrightarrow{k_1} B \xrightarrow{k_2} B + C$, both one-dimensional models correspond to boundaries of the model manifold that are line segments. Taking k_1 or k_2 to infinity also results in perfectly valid one-dimensional models. So far, this sounds a lot like the models

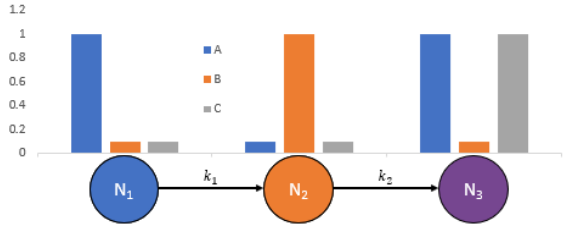
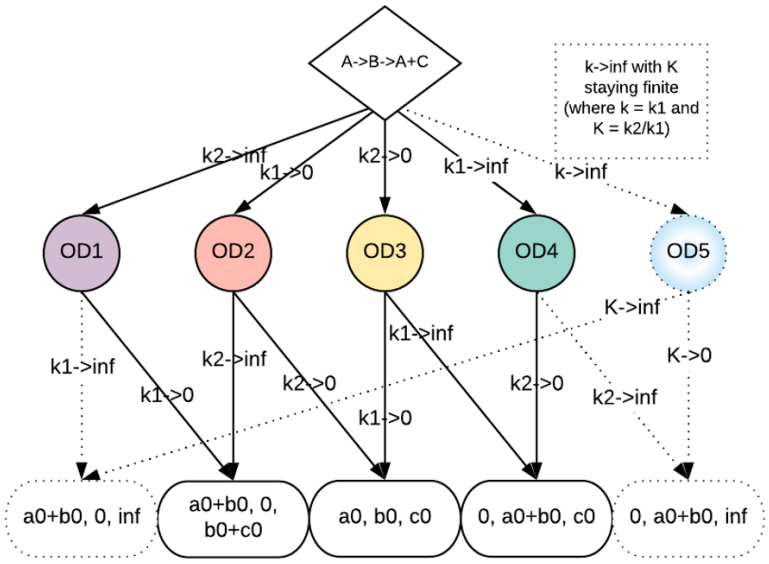


Figure 3.9 Linear simple path network $A \xrightarrow{k_1} B \xrightarrow{k_2} A + C$

of the other linear simple path networks we've analyzed. However, taking the limit $k_1 \to \infty$ then $k_2 \to \infty$ results in a corner that is unbounded. Even more unexpectedly, taking the limit $k_2 \to \infty$ then $k_1 \to \infty$ results in a *different* unbounded corner. This means there needs to be another side to connect the two corners.

It turns out that taking the parameters to infinity together in ratio provides that remaining side, although that side is exclusively at infinity. You can see how this all plays out in Figure 3.10.

We only recently found this model, so we haven't been able to do much analysis regarding how it might help us make predictions regarding other models. However, the main takeaway from this section is to **be careful anytime a reactant shows up in multiple nodes of the network because additional sides may appear and the manifold may be unbounded on some sides**. Having fewer sides happens when taking a parameter to infinity causes the remaining parameter to simply funnel a node back into itself, becoming effectively useless. Extra sides, however, can come because cycles allow for steady states in which reactions still occur (as opposed to in linear simple path networks that don't have reactants as a final product, where steady states only appear after all reactions have stopped).



(a) Hasse diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} A + C$

$k[2] \to 0$ gets $\{a0\, e^{-t\,k[1]},\ a0 + b0 - a0\, e^{-t\,k[1]},\ c0\}$

$k[2] \to 0$ and $k[1] \to 0$ gets $\{a0, b0, c0\}$

$k[2] \to 0$ and $k[1] \to \infty$ gets $\{0, a0 + b0, c0\}$

$k[2] \to \infty$ gets $\{a0 + b0, 0, b0 + c0 + a0\, t\, k[1] + b0\, t\, k[1]\}$

$k[2] \to \infty$ and $k[1] \to 0$ gets $\{a0 + b0, 0, b0 + c0\}$

$k[2] \to \infty$ and $k[1] \to \infty$ gets $\{a0 + b0, 0, (a0 + b0)\, \infty\}$

$k[1] \to 0$ gets $\left\{\frac{a0\,k[2] + b0\,k[2] - b0\,e^{-t\,k[2]}\,k[2]}{k[2]},\ b0\,e^{-t\,k[2]},\ \frac{b0\,k[2]^2 + c0\,k[2]^2 - b0\,e^{-t\,k[2]}\,k[2]^2}{k[2]^2}\right\}$

$k[1] \to 0$ and $k[2] \to 0$ gets $\{a0, b0, c0\}$

$k[1] \to 0$ and $k[2] \to \infty$ gets $\{a0 + b0, 0, b0 + c0\}$

$k[1] \to \infty$ gets $\{0, a0 + b0, c0 + (a0 + b0)\, t\, k[2]\}$

$k[1] \to \infty$ and $k[2] \to 0$ gets $\{0, a0 + b0, c0\}$

$k[1] \to \infty$ and $k[2] \to \infty$ gets $\{0, a0 + b0, (a0 + b0)\, t\, \infty\}$

$k \to \infty$ gets $\left\{\frac{(a0 + b0)\ K}{1 + K},\ \frac{a0 + b0}{1 + K},\ \infty\ K\right\}$

$k \to \infty$ and $K \to 0$ gets $\{0, a0 + b0, \text{Indeterminate}\}$

$k \to \infty$ and $K \to \infty$ gets $\{a0 + b0, 0, \infty\}$

(b) Limits taken to get the Hasse diagram of $A \xrightarrow{k_1} B \xrightarrow{k_2} A + C$

Figure 3.10 (a) To interpret the Hasse diagram, there are four bounded one-dimensional sides, one unbounded one-dimensional side, three bounded corners, and two unbounded corners. They are connected together to form a pentagon that has one of its sides at infinity. (b) The one- and zero-dimensional models that correspond to the different nodes in the Hasse diagram. Note that the one-dimensional models have been color coded.

3.5 Linear Branching Models and Branching like behavior in Simple Path Models

In the last section, we learned that taking parameters to infinity in cycles can frequently lead to a node being its own product, effectively reducing the model to a zero-parameter model and facilitating the need for a ratio to describe the remaining side. However, there is another reason ratios may be required that is best exemplified by linear branching models. In these models, taking a parameter to infinity results in multiple reactions running out of at least one reactant, again reducing the number of effective parameters by more than one. As such, we need to use ratios to find the last side.

The simplest linear branching network was encountered earlier in Figure 3.3d and is shown again in Figure 3.11. Rather than its model manifold having four sides like linear simple path models that have no reactants as a final product, we will see that the model manifold of this model is actually triangular.

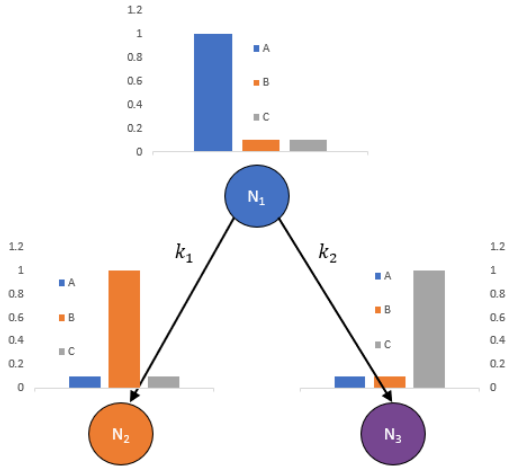


Figure 3.11 $A \xrightarrow{k_1} B$, $A \xrightarrow{k_2} C$, meaning reaction 1 consumes A and turns it into B while reaction 2 consumes A and turns it into C. This is the same figure shown in Figure 3.3d.

Let's analyze this network to see why its model manifold is shaped like a triangle, starting with writing out the mass action model describing the network:

$$\frac{da}{dt} = -k_1 a - k_2 a \tag{3.47}$$

$$\frac{db}{dt} = k_1 a \tag{3.48}$$

$$\frac{dc}{dt} = k_2 a \tag{3.49}$$

$$a(0) = a_0 \tag{3.50}$$

$$b(0) = b_0 \tag{3.51}$$

$$c(0) = c_0 \tag{3.52}$$

After solving these equations and taking limits of k_1 and k_2 to infinity and/or zero, I end up with the collection of equations found in Figure 3.12. Notice that, though taking k_1 or k_2 to zero produce acceptable one-dimensional models, taking either parameter to infinity produces a zero-dimensional model. In other words, what we see so far is a Hasse diagram that looks exactly like the one in Figure 3.5.

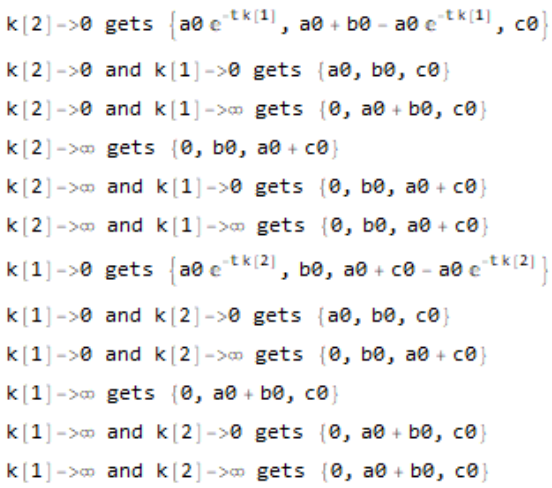


Figure 3.12 The one- and zero-dimensional models that result from taking k_1 or k_2 to infinity or zero in the model describing the network $A \xrightarrow{k_1} B$, $A \xrightarrow{k_2} C$.

It turns out that taking either parameter to infinity results in a zero-dimensional model because, as $k_1 \to \infty$ or $k_2 \to \infty$, $a \to 0$. Because both reactions require $a \neq 0$ to continue, they both cease instantly. Instead of getting a new, one-dimensional side to our two-dimensional manifold, a side that would have been described by a one-dimensional (one-parameter) model, we get just a single point described by a zero-parameter model.[5] As mentioned earlier, the way to get the last side of this model is by taking both parameters to infinity at once in ratio to each other.

If we reparameterize the model so that $k = k_1$ and $K = k_2/k_1$, we end up with this new set of equations:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -k(a - Ka) \tag{3.53}$$

$$\frac{\mathrm{d}b}{\mathrm{d}t} = ka \tag{3.54}$$

$$\frac{\mathrm{d}c}{\mathrm{d}t} = kKa \tag{3.55}$$

$$(a(0), b(0), c(0)) = (a_0, b_0, c_0) \tag{3.56}$$

Now, if we solve this set of equations and take k to infinity, we end up with a model that still depends on the parameter K. In other words, the final concentrations depend yet again on the ratio between the two initial parameters:

$$a = 0 \tag{3.57}$$

$$b = \frac{a_0 + b_0 + b_0 K}{1 + K} \tag{3.58}$$

$$c = \frac{a_0 K + c_0 + c_0 K}{1 + K} \tag{3.59}$$

Taking K to infinity or zero shows that this side connects with the other sides at their endpoints:

[5] The zero-dimensional model would either be $(a_f, b_f, c_f) = (0, b_0 + a_0, c_0)$ or $(a_f, b_f, c_f) = (0, b_0, c_0 + a_0)$, depending on which parameter you take to infinity.

$k \to \infty$, then $K \to 0$

$$a_f = 0 \tag{3.60}$$

$$b_f = a_0 + b_0 \tag{3.61}$$

$$c_f = c_0 \tag{3.62}$$

$k \to 0$, then $K \to \infty$

$$a_f = 0 \tag{3.63}$$

$$b_f = b_0 \tag{3.64}$$

$$c_f = a_0 + c_0 \tag{3.65}$$

Therefore, we predict that the manifold will have three one-dimensional boundaries: two for when a parameter goes to zero, and a third for when the two parameters go to infinity in ratio with each other. Those three sides are then connected via one of three zero-dimensional models that represent the final concentrations when all parameters are at their limits. In other words, we predict we will see a triangle, which is exactly what we see in Figure 3.13.

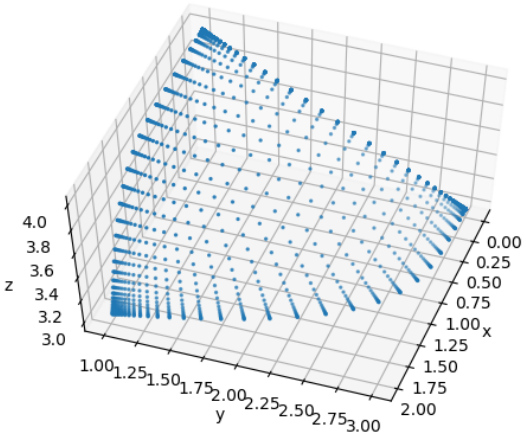


Figure 3.13 Model manifold of $A \xrightarrow{k_1} B$, $A \xrightarrow{k_2} C$. The Hasse diagram is the same as in Figure 3.7.

Now to show how branch-like[6] ratios apply to simple path models. It turns out that, like linear branching models, there exist simple path models in which taking any parameter to infinity reduces the model by multiple dimensions. For example, let's analyze the network $2A \xrightarrow{k_1} A+B \xrightarrow{k_2} B+C$. Using the equivalent model, we can see that it has the same steady states as the linear branching model we just observed:

$$A \xrightarrow{k_1} B \tag{3.66}$$

$$A \xrightarrow{k_2} C \tag{3.67}$$

Taking either parameter to infinity results in the concentration of A going to zero, abruptly ending both reactions. This is just like with the model $2A+B \xrightarrow{k_1} A+2B \xrightarrow{k_2} 2B+C$, which we discussed earlier in the section "The Equivalent Model: Another Tool for Grouping Similar Networks." Therefore, we might guess that this network would have a model manifold that is topologically a triangle[7], which is exactly what we see when we analyze it (not discussed here) and plot it (see Figure 3.14).

In the last two examples, taking either parameter to infinity resulted in a zero-dimensional model. However, there exist two-parameter networks which reduce to a zero-parameter network when one parameter is taken to infinity but not when the other parameter is taken to infinity. The simplest such network is probably $A+B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$.

In the network $A+B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$, taking either parameter to zero results in

[6]Note: this was a fairly simple example of a branching network; after all, not only was it linear, but it also had none of the reactants as a final product. The manifolds of nonlinear branching networks or branching networks with reactants as products can be complicated in many of the ways discussed throughout this report. As such, many of the techniques discussed in this report can be applied to branching networks, but such studies are outside the scope of this report.

[7]Just like the branching model has a manifold that is topologically a triangle

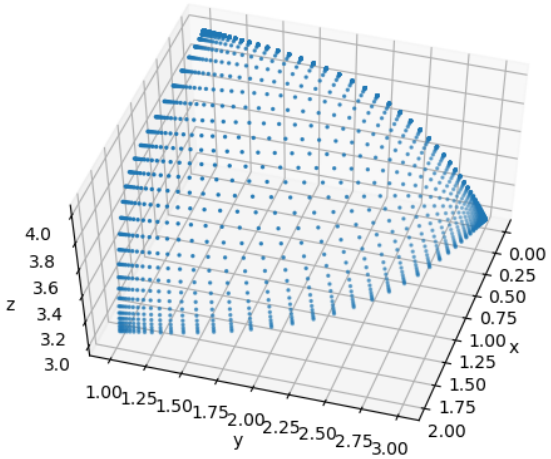


Figure 3.14 Model manifold of $2A \xrightarrow{k_1} A+B \xrightarrow{k_2} B+C$. The Hasse diagram is the same as in Figure 3.7.

a bounded one-dimensional model. Furthermore, taking k_1 to infinity also results in a bounded one-dimensional model. However, taking k_2 to infinity cause b to go to zero, ending both reactions. So far we can describe these connections using the Hasse diagram seen in Figure 3.15. I suspect that it's possible to use ratios to create the missing side, but how that might work is beyond the scope of this report.

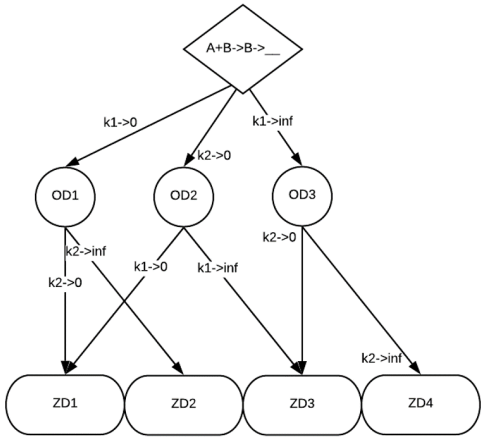


Figure 3.15 Incomplete Hasse diagram of $A + B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$.

Note, networks that behave like $A + B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$ provide the only counterex-

ample I've found to categorizing models according to their equivalent model. They are a counterexample because both $A + B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$ and $A \xrightarrow{k_1} \_\_\_$, $B \xrightarrow{k_2} \_\_\_$ have the same equivalent model,[8] even though $A \xrightarrow{k_1} \_\_\_$, $B \xrightarrow{k_2} \_\_\_$ doesn't have problems when $k_2 \to \infty$.

In summary, **another reason to be cautious whenever reactants show up in more than one stoichiometry is because taking one parameter to infinity may cause multiple reactions to run out of reactant. If this happens, taking that parameter to infinity by itself is no longer a valid way to find a side of the model manifold, so you may need to resort to ratios or other parameter combinations.**

3.6 Dependency on Initial Conditions

I will just briefly touch on how the model manifolds of a mass action model may be dependent on the initial conditions. More specifically, I will discuss how the initial conditions which we choose in making our cross section may affect whether the cross section we see is a good representative of the full model manifold.

To start off, the specific dimensions of the manifold cross section are obviously dependent on the initial conditions, especially since boundaries are frequently along where a concentration stays constant. However, as discussed in the introduction, it is not very often that the initial conditions cause the structure of the manifold cross section to be different than that of the full model manifold, at least so long as we are

[8] Here is the equivalent model of both $A + B \xrightarrow{k_1} B \xrightarrow{k_2} \_\_\_$ and $A \xrightarrow{k_1} \_\_\_$, $B \xrightarrow{k_2} \_\_\_$:

$$A \xrightarrow{k_1} \_\_\_ \tag{3.68}$$

$$B \xrightarrow{k_2} \_\_\_ \tag{3.69}$$

using generic initial conditions. There are two common ways that initial conditions don't affect the manifold cross section's structure and at least one way that I've seen in which they do affect its structure.

First, because we use generic initial conditions, the final manifold structure isn't based on any particular set of initial conditions. This was discussed earlier with a model of radioactive decay, and the principle carries over to here nicely. For example, whether a_0 is set to one, ten, or ten thousand, the mass action model describing the network $A \xrightarrow{k_1} B$ would still have the same limits to which we take k_1 and we would still get a one-dimensional bounded model. Setting $a_0 = 0$ in the network $A \xrightarrow{k_1} B$ *would* change the manifold structure of that particular model,[9] but we already discussed that zero often can't be considered an initial condition. After all, if you were to pick set of generic initial conditions, statistically you would almost certainly choose numbers other than zero.

Secondly, sometimes the difference between two or more initial conditions affect the final outcome of a normal network, such as in the network $A + B \xrightarrow{k_1} C$. To elaborate, lets examine what happens as k_1 is taken to infinity. If $a_0 > b_0$, the final concentration of B, denoted by b_f, would go to zero, and the final concentration of A, denoted by a_f, would go to $a_0 - b_0$. On the other hand, $a_0 < b_0$ would lead to a_f going to zero and b_f going to $b_0 - a_0$. However, the limits to which we take the parameters don't change even when the final values change, so Hasse diagrams of this model and models like this one would still represent line segments. As such, picking any set of initial conditions to use in our cross section should still give us a good representation of the full model manifold.

With those two ideas in mind, we can feel confident that we have the right manifold structure of all the models we've examined in this paper so far. There is, however,

[9]The reaction can't ever start, so the manifold is just a point.

at least one network that I have found in which a cross section with generic initial conditions is not a good representation of the full model manifold. I didn't find this network until the last few days of my internship, so I haven't had time to analyze it in great detail. However, I am reasonably sure that it is different enough from the other networks that I have discussed in this report that it can be discussed separately from them. Specifically, let us consider the network $A + B \xrightarrow{k_1} B + C \xrightarrow{k_2} \_\_\_$.

In this network, if you start with $b_0 > a_0$ and $b_0 > c_0$, then everything behave exactly like a linear simple path network with no reactants as final products. In other words, there are four limits that can initially be taken to find a side of the model manifold: $k_1 \to 0$, $k_1 \to \infty$, $k_2 \to 0$, and $k_2 \to \infty$. However, if $b_0 < a_0$ and $b_0 < c_0$, we end up with the something like a linear branching model, where taking either parameter to infinity reduces the model to a zero-dimensional model. As such, the limits that can be taken are $k_1 \to 0$, $k_2 \to 0$, and $k \to \infty$ with K kept finite (where $k = k_1$ and $K = k_2/k_1$). Alternatively, if b_0 was greater than one of a_0 or c_0 but not the other, we would end up with a situation where one parameter can be taken to infinity normally but the other cannot.

Because there are five different limits which, depending on the initial conditions, you can take the parameters to,[10] this network has a bounded pentagon-shaped model manifold. However, it is possible that no cross section of the model manifold will be shaped like a pentagon since I'm not sure how to take into account that different limits are needed for different sets of initial conditions. Regardless, it is important to note that **two parameter networks do have the potential of having more than four different bounded sides, and that some of those sides may only be revealed with certain initial conditions**.

[10] $k_1 \to 0$, $k_1 \to \infty$, $k_2 \to 0$, $k_2 \to \infty$, and $k \to \infty$ with K kept finite (where $k = k_1$ and $K = k_2/k_1$).

Chapter 4

Conclusion

Instead of showing that certain network structures consistently lead to certain manifold structures like we had initially endeavored to do, we found that the manifold structures of simple path networks are very dependent on the specific stoichiometries in that network, and sometimes even the initial concentrations of the species. In the process of discovering this, we explored new methods of categorizing models (the equivalent model, independent species) and some aspects of the stoichiometries that may lead to unexpected manifold structures (unboundedness, cycles, running out of reactants).

Regarding future research, the goal to find a way to connect the network structure to the manifold structure has not yet been achieved. I still believe that it should be possible to prove some relationships, perhaps starting with my conjecture that all linear simple path models with no reactants as a final product will have a hypercubic manifold structure. I further conjecture that the independent species principle holds for all networks, and proving it to be true should allow for further breakthroughs.

This research was supported by NSF Grant# 1757998

Bibliography

[1] M. K. Transtrum and P. Qiu, "Model Reduction by Manifold Boundaries," Physical Review Letters (2014).

[2] M. K. Transtrum, B. B. Machta, K. S. Brown, B. C. Daniels, C. R. Myers, and J. P. Sethna, "Perspective: Sloppiness and emergent theories in physics, biology, and beyond," The Journal of Chemical Physics (2015).

[3] M. K. Transtrum, G. Hart, and P. Qiu, "Information topology identifies emergent model classes," (2016). 1409.6203.

[4] K. S. Brown, C. C. Hill, G. A. Calero, C. R. Myers, K. H. Lee, J. P. Sethna, and R. A. Cerione, "The statistical mechanics of complex signaling networks: nerve growth factor signaling," Physical Biology **1**(3), 184–195 (2004). URL https://doi.org/10.1088/1478-3967/1/3/006.

[5] L. W. Tu, *An Introduction to Manifolds* (Springer-Verlag, New York, 2011).

[6] "Chemical reaction network theory," (2020). URL https://en.wikipedia.org/wiki/Chemical_reaction_network_theory#Overview.

[7] C. R. Bagshaw, *Law of Mass Action*, pp. 1233–1234 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013). URL https://doi.org/10.1007/978-3-642-16712-6_574.

Appendix A

Code

A.1 Two-Parameter Plotting Code

This is the code which I used to plot the model manifolds of two-parameter mass action models. It is written in Julia 1.4.1. Note: you can choose which concentration is displayed on each axis. In addition, you can also choose which time point and with what set of initial conditions each axis uses to make predictions.

In order to run the program, you will need to first install all the needed packages (including some custom made packages) using the program in the section "Package Installer" found on page 92 of this report. You will also need another file, the one in the section "File of Models" (page 86), in the same folder as this plotting code to run the code properly.

When running the plotting code, use the code "mymodel = " on line 83 to choose which model in your file of models to use when plotting.

```
1 # Print time and program name. Make it clear the program has started
2 using Dates
3 println("\n\n\n\nTime ",
4     now())
```

```
5
6  using DelimitedFiles
7  import Models
8  import ModularLM # used for doing fitting
9  import ParametricModels # used for models like the ODE model
10 using Parameters
11 println("Halfway through packages")
12 using LinearAlgebra
13 import Random
14 using PyPlot
15 import Sundials # a package for solving ODEs
16
17 ### Setup
18 docostcont = true # Do you want a contour plot
19 domanifold = true # Do you want a plot of the model manifold
20 scattering = true # Get a scatter plot of the model manifold...
21 surfing = true # ...and/or a surface plot of the model manifold
22 # After the first time running this program, feel free to move the "
       mymodel" line here
23
24
25 myobs = [1,2,3] # Which variables do you want to observe
26 mytimes = [1,2,3] # Which times do you want to observe
27
28 # Initial Concentrations:
29 A0 = [2.0,2.0,1.0]
30 B0 = [3.0,1.0,5.0]
31 C0 = [1.0,3.0,3.0]
32 t = [.7, 3, 5]# Quantities of interest (expandable)
33
34 num = 50 # Number of points in each parameter (or you can customize
```

```
   it \|/)
parRange = 10 # Parameter range (or you can customize it \|/)
x1 = range(-parRange, parRange, length=num) |> collect
x2 = range(-parRange, parRange, length=num) |> collect

# Cost contour stuff
xi=log.([0.7, 2.0]) # Best fit parameters
sigma = 0.01 # standard deviation (for creating the data)
# How many colors the plot uses to distinguish costs
ContourLevels = 50
# If wanting to narrow the color range, use something like
#ContourLevels = range(11.9,stop=12.1,length=100)
# instead


### Create a Julia type that holds parameter values
# @with_kw allows you to set default values (such as k1 = exp(xi[1])
   ).
#      /|\ Comes from the parameters package, I think
# @deftype T sets the default type inside this structure to be T. In
    other words,
#      because k1 and k2 don't have a type specified, they are of
   type T.
#  /|\ Comes from the parameters package, I think
# Wnt{T<:Real} <: ParametricModels.AbstractParameterSpace{T} means
   that we
#      declare Wnt{T} to be a subtype ParametricModels.
   AbstractParameterSpace{T}
#      for every T that is a subtype of Real, I think
# One reason to do this is you can have parameters with different
   names
```

```
@with_kw mutable struct Wnt3p{T<:Real} <: ParametricModels.
    AbstractParameterSpace{T} @deftype T
     k1   = exp(xi[1])
     k2   = exp(xi[2])
     k3   = exp(xi[3])
end
@with_kw mutable struct Wnt2p{T<:Real} <: ParametricModels.
    AbstractParameterSpace{T} @deftype T
     k1   = exp(xi[1])
     k2   = exp(xi[2])
end

# Connect this program to the file with the different models
# Note: the "include" line needs to be after the Wnt lines
include("my MAM models.jl")
# mymodel lets you choose which model to use from the file "my MAM
    models."
# For example, if mymodel = A1B1C, then you will use the model for
    the network
# A->B->C. If, on the other hand, mymodel = AB2C, the plotting
    program would
# use the model for the network A+B<->C. Currently it is set to the
    most useful
# model, asN1N1N13s, which stands for "Arbitratray stoichiometeries:
     Node 1 goes
# to Node 2 goes to Node 3 with three species involved at most." See
     the file
# "my MAM models" for details of how that works. Note that this
    plotting program
# only works for models that have two parameters.

```

```
81 # The "mymodel" line must be beneath the "include" line during  1st
       run;
82 # afterwards, you can move it up to where the initial conditions are
       defined
83 mymodel = asN1N1N3s
84
85 println("Model file included")
86 ### Initial condition function
87 # ic accepts only arguments of type Wnt{T} where T is a subtype of
       Real
88 function ic1(ps::Wnt2p{T}) where T<:Real
89     # Convert A0,B0, and C0 to Type T, while making sure they
90     # (the parameters? The ics?) are type T
91     return T[A0[1], B0[1], C0[1]]
92 end
93 function ic2(ps::Wnt2p{T}) where T<:Real
94     # Convert A0,B0, and C0 to Type T, while making sure they
95     # (the parameters? The ics?) are type T
96     return T[A0[2], B0[2], C0[2]]
97 end
98 function ic3(ps::Wnt2p{T}) where T<:Real
99     # Convert A0,B0, and C0 to Type T, while making sure they
100    # (the parameters? The ics?) are type T
101    return T[A0[3], B0[3], C0[3]]
102 end
103
104 ### Define observation function
105 function obs(ps::Wnt2p{T}, t, y) where T<:Real
106     ans = zeros(length(myobs))
107     for i in 1:length(myobs)
108         ans[i] = y[myobs[i]]
```

```
109     end
110     return ans
111 end
112
113
114 ### Specify data object to specify what time point to evaluate this
    at
115 ### and specify the data
116 Random.seed!(0) # I don't think I'm going to be able to get cost
    contours to work right with this
117 parametricmodel1 = @ParametricModels.ODEModel(ParametricModels.
    OLSData(
118     "TempData",zeros(length(t)*length(myobs))), Wnt2p, ic1, mymodel,
     obs, t,
119     (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:
    reltol, 1e-6)])
120 ydata1 = ParametricModels.model_predictions(parametricmodel1) +
121     Random.randn((length(t),length(myobs)))*sigma
122 data1 = ParametricModels.OLSData("MyData1", ydata1) # Might need to
    change to MyData
123 #
124 parametricmodel2 = @ParametricModels.ODEModel(ParametricModels.
    OLSData(
125     "TempData",zeros(length(t)*length(myobs))), Wnt2p, ic2, mymodel,
     obs, t,
126     (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:
    reltol, 1e-6)])
127 ydata2 = ParametricModels.model_predictions(parametricmodel2) +
128     Random.randn((length(t),length(myobs)))*sigma
129 data2 = ParametricModels.OLSData("MyData2", ydata2)
130 #
```

```
parametricmodel3 = @ParametricModels.ODEModel(ParametricModels.OLSData(
    "TempData",zeros(length(t)*length(myobs))), Wnt2p, ic3, mymodel, obs, t,
    (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:reltol, 1e-6)])
ydata3 = ParametricModels.model_predictions(parametricmodel3) +
    Random.randn((length(t),length(myobs)))*sigma
data3 = ParametricModels.OLSData("MyData3", ydata3)


### Define the parametric model
parametricmodel1 = @ParametricModels.ODEModel(data1, Wnt2p, ic1, mymodel, obs, t,
    (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:reltol, 1e-6)])
parametricmodel2 = @ParametricModels.ODEModel(data2, Wnt2p, ic2, mymodel, obs, t,
    (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:reltol, 1e-6)])
parametricmodel3 = @ParametricModels.ODEModel(data3, Wnt2p, ic3, mymodel, obs, t,
    (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:reltol, 1e-6)])
println("Parametric models defined")

### Transform parameters that need to be positive
# Tells the code how to transform the parameters so that the inputs to the model
# can be all reals. So long as worrying about MAMs, probably don't have to worry
```

```
151 # about changing
152 for i in 1:length(xi)
153     parametricmodel1.parameters[i].t = exp
154     parametricmodel1.parameters[i].invt = log
155 end
156 for i in 1:length(xi)
157     parametricmodel2.parameters[i].t = exp
158     parametricmodel2.parameters[i].invt = log
159 end
160 for i in 1:length(xi)
161     parametricmodel3.parameters[i].t = exp
162     parametricmodel3.parameters[i].invt = log
163 end
164 println("Parameter inverses defined")
165
166 ### Take parametric model and make it a normal model
167 # The parameteric model is useful for being solvable using Dr.
      Transtrum's packages
168 # The normal model structure is useful for other reasons the will be
      used
169 # throughout this program
170 model1 = Models.Model(parametricmodel1)
171 model2 = Models.Model(parametricmodel2)
172 model3 = Models.Model(parametricmodel3)
173
174
175 function Cost1(x)
176     return 0.5*sum(model1.r(x).^2)/sigma^2
177 end
178 function Cost2(x)
179     return 0.5*sum(model2.r(x).^2)/sigma^2
```

```
end
function Cost3(x)
    return 0.5*sum(model3.r(x).^2)/sigma^2
end


if docostcont || domanifold
    num1 = length(x1)
    num2 = length(x2)
    cost = zeros(num2, num1)
    manif = []
    println("Beginning loop for filling out the graphs")
    for i in 1:num1
        for j in 1:num2
            params = [x1[i], x2[j]]
            if docostcont
                cost[j,i] = (Cost1(params)+Cost2(params)+Cost3(
params))
            end
            if domanifold
                appendable = []
                append!(appendable,[(ydata1-reshape(model1.r(params)
,
                    (length(t),length(myobs))))[mytimes[1],myobs
[1]]])
                append!(appendable,[(ydata2-reshape(model2.r(params)
,
                    (length(t),length(myobs))))[mytimes[2],myobs
[2]]])
                append!(appendable,[(ydata3-reshape(model3.r(params)
,
```

```
205                     (length(t),length(myobs))))[mytimes[3],myobs
    [3]]])
206                 append!(manif, [appendable])
207             end
208         end
209         print("*")
210     end
211
212
213     println("\nDone with filling out the graphs")
214
215     if docostcont
216         figure()
217         contourf(x1, x2, log.(cost), levels=ContourLevels )
218         colorbar(label="log(cost)")
219         scatter([xi[1]], [xi[2]], c="red", marker="x") # Plot the
    best fit point
220         xlabel("log(k1)")
221         ylabel("log(k2)")
222     end
223
224     if domanifold
225         # manif contains the predictions specified at the top.
226         # Take all of the first predictions and store them in p1,
227         # then the second predictions in p2, etc.
228         p1 = [el[1] for el in manif]
229         p2 = [el[2] for el in manif]
230         p3 = [el[3] for el in manif]
231         if scattering
232             figure()
233             scatter3D(p1, p2, p3, s=2)
```

```
            xlabel("x")
            ylabel("y")
            zlabel("z")
        end
        if surfing
            figure()
            surf(p1,p2,p3)
            xlabel("x")
            ylabel("y")
            zlabel("z")
        end
    end
end


###### model.r returns the data - observable
###### m.model.r([1.0,0.1,3.0]) |> plot # to plot stuff
###### m.ydata - m.model.r([2.0,0.0,3.0]) returns just the
    observable (KE)
PyPlot.display_figs() # required to see the graph in Atom
println("Finished")
```

A.2 Three-Parameter Plotting Code

This program (also written in Julia) has all the same functionality and requirements of the two-parameter plotting code except for two things: first, it works for models that require three parameters instead of two parameters; secondly, each axis has to use the same initial conditions.

```
# Print time and program name. Make it clear the program has started
```

```
using Dates
println("\n\n\n\nTime ",
    now())

using DelimitedFiles
import Models
import ModularLM # used for doing fitting
import ParametricModels # used for models like the ODE model
using Parameters
println("Halfway through packages")
using LinearAlgebra
import Random
using PyPlot
import Sundials # a package for solving ODEs

### Setup
docostcont = false # Do you want a contour plot
domanifold = true # Do you want a plot of the model manifold
scattering = true # Get a scatter plot of the model manifold...
surfing = true # ...and/or a surface plot of the model manifold
# After the first time running this program, feel free to move the "
   mymodel" line here


myobs = [1,2,3] # Which variables do you want to observe
mytimes = [1,2,3] # Which times do you want to observe

# Initial Concentrations:
A0 = 1.0
B0 = 2.0
C0 = 5.0
```

```
32 t = [.7, 3, 5] # Quantities of interest (expandable)
33
34 num = 25 # Number of points in each parameter (or you can customize
       it \|/)
35 parRange = 10 # Parameter range (or you can customize it \|/)
36 x1 = range(-parRange, parRange, length=num) |> collect
37 x2 = range(-parRange, parRange, length=num) |> collect
38 x3 = range(-parRange, parRange, length=num) |> collect
39
40 # Cost contour stuff
41 xi=log.([0.7, 2.0, 3.0]) # Best fit parameters
42 sigma = 0.01 # standard deviation (for creating the data)
43 # How many colors the plot uses to distinguish costs
44 ContourLevels = 50
45 # If wanting to narrow the color range, use something like
46 #ContourLevels = range(11.9,stop=12.1,length=100)
47 # instead
48
49
50 ### Create a Julia type that holds parameter values
51 # @with_kw allows you to set default values (such as k1 = exp(xi[1])
       ).
52 #      /|\ Comes from the parameters package, I think
53 # @deftype T sets the default type inside this structure to be T. In
       other words,
54 #      because k1 and k2 don't have a type specified, they are of
       type T.
55 #  /|\ Comes from the parameters package, I think
56 # Wnt{T<:Real} <: ParametricModels.AbstractParameterSpace{T} means
       that we
```

```
#       declare Wnt{T} to be a subtype ParametricModels.AbstractParameterSpace{T}
#       for every T that is a subtype of Real, I think
# One reason to do this is you can have parameters with different names
@with_kw mutable struct Wnt3p{T<:Real} <: ParametricModels.AbstractParameterSpace{T} @deftype T
    k1  = exp(xi[1])
    k2  = exp(xi[2])
    k3  = exp(xi[3])
end
@with_kw mutable struct Wnt2p{T<:Real} <: ParametricModels.AbstractParameterSpace{T} @deftype T
    k1  = exp(xi[1])
    k2  = exp(xi[2])
end

# Connect this program to the file with the different models
# Note: the "include" line needs to be after the Wnt lines
include("my MAM models.jl")
# mymodel lets you choose which model to use from the file "my MAM models."
# For example, if mymodel = A1B1C1, then you will use the model for the network
# A->B->C->__. If, on the other hand, mymodel = A1B1_andA1C, the plotting
# program would use the model for the network A->B->__ and A->C. Note that this
# plotting program only works for models that have three parameters.

```

```
# The "mymodel" line must be beneath the "include" line during 1st run;
# afterwards, you can move it up to where the initial conditions are defined
mymodel = A1B1C1#A1B1_andA1C

println("Model file included")
### Initial condition function
# ic accepts only arguments of type Wnt{T} where T is a subtype of Real
function ic(ps::Wnt3p{T}) where T<:Real
    #Convert A0,B0, and C0 to Type T, while making sure it is type T
    return T[A0, B0, C0]
end

### Define observation function
function obs(ps::Wnt3p{T}, t, y) where T<:Real
    ans = zeros(length(myobs))
    for i in 1:length(myobs)
        ans[i] = y[myobs[i]]
    end
    return ans
end


### Specify data object to specify what time point to evaluate this at
### and specify the data
Random.seed!(0)
parametricmodel = @ParametricModels.ODEModel(ParametricModels.OLSData(
```

```
105     "TempData",zeros(length(t)*length(myobs))), Wnt3p, ic, mymodel,
    obs, t,
106     (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:
    reltol, 1e-6)])
107 ydata = ParametricModels.model_predictions(parametricmodel) +
108     Random.randn((length(t),length(myobs)))*sigma
109 data = ParametricModels.OLSData("MyData", ydata)
110 println("Parametric models defined")
111
112 ### Define the parametric model
113 parametricmodel = @ParametricModels.ODEModel(data, Wnt3p, ic,
    mymodel, obs, t,
114     (Sundials.CVODE_BDF(),), Tuple{Symbol, Any}[(:abstol, 1e-6), (:
    reltol, 1e-6)])
115
116
117 ### Transform parameters that need to be positive
118 # Tells the code how to transform the parameters so that the inputs
    to the model
119 # can be all reals. So long as worrying about MAMs, probably don't
    have to worry
120 # about changing
121 for i in 1:length(xi)
122     parametricmodel.parameters[i].t = exp
123     parametricmodel.parameters[i].invt = log
124 end
125 println("Parameter inverses defined")
126
127 ### Take parametric model and make it a normal model
128 # The parameteric model is useful for being solvable using Dr.
    Transtrum's packages
```

```
129 # The normal model structure is useful for other reasons the will be
        used
130 # throughout this program
131 model = Models.Model(parametricmodel)
132
133 function Cost(x)
134     return 0.5*sum(model.r(x).^2)/sigma^2
135 end
136
137
138 if docostcont || domanifold
139     num1 = length(x1)
140     num2 = length(x2)
141     num3 = length(x3)
142     cost = zeros(num2, num1)
143     manif = []
144     println("Beginning loop for filling out the graphs")
145     for i in 1:num1
146         for j in 1:num2
147             for k in 1:num3
148                 params = [x1[i], x2[j], x3[k]]
149                 if docostcont # NOT ADAPTED FOR 3 Parameters
150                     cost[j,i] = Cost(params)
151                 end
152                 if domanifold
153                     appendable = []
154                     for cntr in 1:3
155                         append!(appendable,[(ydata-reshape(model.r(
    params),
156                             (length(t),length(myobs))))[mytimes[cntr
    ],myobs[cntr]]])
```

```
157                     end
158                     append!(manif, [appendable])
159                 end
160             end
161         end
162         print("*")
163     end
164
165
166     println("\nDone with filling out the graphs")
167     if docostcont #NOT ADAPTED FOR 3 Parameters
168         figure()
169         contourf(x1, x2, log.(cost), levels=ContourLevels )
170         colorbar(label="log(cost)")
171         scatter([xi[1]], [xi[2]], c="red", marker="x") # Plot the
    best fit point
172         xlabel("log(k1)")
173         ylabel("log(k2)")
174     end
175
176     if domanifold
177         # manif contains the predictions specified at the top.
178         # Take all of the first predictions and store them in p1,
179         # then the second predictions in p2, etc.
180         p1 = [el[1] for el in manif]
181         p2 = [el[2] for el in manif]
182         p3 = [el[3] for el in manif]
183         if scattering
184             figure()
185             scatter3D(p1, p2, p3, s=2)
186             xlabel("x")
```

```
187             ylabel("y")
188             zlabel("z")
189         end
190         if surfing
191             figure()
192             surf(p1,p2,p3)
193             xlabel("x")
194             ylabel("y")
195             zlabel("z")
196         end
197     end
198 end
199
200
201 ###### model.r returns the data - observable
202 ###### m.model.r([1.0,0.1,3.0]) |> plot # to plot stuff
203 ###### m.ydata - m.model.r([2.0,0.0,3.0]) returns just the
    observable (KE)
204 PyPlot.display_figs() # required to see the graph in Atom
205 println("Finished")
```

A.3 File of Models

This program needs to be named "my MAM models" and be in the same folder as the plotting programs in order for those programs to work. It has many ready made models that can be run, or you can create your own. You can either create your model from scratch or, if it is a simple path network with three species and two parameters, there is a model called asN1N1N13s that can be easily edited to work with your network. There is also one for four species and two parameters called asN1N1N14s.

```
### Name this file "my MAM models" and put it in the same folder as your
# plotting code in order to use the models in this file to create plots

### Define the mass action models of the networks you want to model
# Wnt2p only works with the program that focuses on two parameter models,
# Wnt3p only works with the program that focuses on three parameter models
#
# y[1] is the concentration of the first substance, y[2] is the concentration
# of the second substance, etc. dy[1] is the change in concentration of the
# first substance, etc.
#
# ps.k1 is the first parameter, ps.k2 is the second parameter, etc.


#A+B<->C
function AB2C(ps::Wnt2p{T}, t, y, dy) where T<:Real
    dy[1]   = -ps.k1*y[1]*y[2] + ps.k2*y[3]
    dy[2]   = -ps.k1*y[1]*y[2] + ps.k2*y[3]
    dy[3]   =  ps.k1*y[1]*y[2] - ps.k2*y[3]
    nothing
end

#A->B->C
function A1B1C(ps::Wnt2p{T}, t, y, dy) where T<:Real
    dy[1]   = -ps.k1*y[1]
```

```
26      dy[2]    = ps.k1*y[1] - ps.k2*y[2]
27      dy[3]    = ps.k2*y[2]
28      nothing
29  end
30
31  #A->B, doesn't work on two-parameter or three-parameter program
32  function A1B(ps::Wnt2p{T}, t, y, dy) where T<:Real
33      dy[1]    = -ps.k1*y[1]
34      dy[2]    = ps.k1*y[1]
35      nothing
36  end
37
38  #A->B, C->D
39  function A1B0C1D(ps::Wnt2p{T}, t, y, dy) where T<:Real
40      dy[1]    = -ps.k1*y[1]
41      dy[2]    = ps.k1*y[1]
42
43      dy[3]    = -ps.k2*y[3]
44      dy[4]    = ps.k2*y[3]
45      nothing
46  end
47
48  # A goes to B or C
49  function A1BoC(ps::Wnt2p{T}, t, y, dy) where T<:Real
50      dy[1]    = -ps.k1*y[1] - ps.k2*y[1]
51      dy[2]    = ps.k1*y[1]
52      dy[3]    = ps.k2*y[1]
53
54      nothing
55  end
56
```

```
# Arbitrary stoichiometry, node one -> node 2 -> node 3, with 3
    species
function asN1N1N3s(ps::Wnt2p{T}, t, y, dy) where T<:Real
    #=A+B->B->__=#
    a = [1,0,0]
    b = [1,1,0]
    c = [0,0,0]
    #=3A+1B+C->1A+2B->2B+C
    a = [3,1,0]
    b = [1,3,2]
    c = [1,0,1]=#
    v1 = ps.k1 * y[1]^a[1] * y[2]^b[1] * y[3]^c[1]
    v2 = ps.k2 * y[1]^a[2] * y[2]^b[2] * y[3]^c[2]

    dy[1]   = -(a[1]-a[2])*v1 - (a[2]-a[3])*v2
    dy[2]   = -(b[1]-b[2])*v1 - (b[2]-b[3])*v2
    dy[3]   = -(c[1]-c[2])*v1 - (c[2]-c[3])*v2
    nothing
end

# A<->B
function A2B(ps::Wnt2p{T}, t, y, dy) where T<:Real
    dy[1]   = -ps.k1*y[1] + ps.k2*y[2]
    dy[2]   =  ps.k1*y[1] - ps.k2*y[2]
    dy[3]   = 0
    nothing
end


# A->B->C->A
function A1B1C1A(ps::Wnt3p{T}, t, y, dy) where T<:Real
```

```
87      dy[1]    = -ps.k1*y[1] + ps.k3*y[3]
88      dy[2]    =  ps.k1*y[1] - ps.k2*y[2]
89      dy[3]    =  ps.k2*y[2] - ps.k3*y[3]
90      nothing
91  end
92
93  # A goes to B which goes to __ while A also goes to __
94  function A1B1_andA1C(ps::Wnt3p{T}, t, y, dy) where T<:Real
95      dy[1]    = -ps.k1*y[1] - ps.k2*y[1]
96      dy[2]    =  ps.k1*y[1] - ps.k3*y[2]
97      dy[3]    =  ps.k2*y[1]
98      nothing
99  end
100
101 # A goes to B or A goes to C or A goes to __
102 function A1BorCor__(ps::Wnt3p{T}, t, y, dy) where T<:Real
103     dy[1]    = -ps.k1*y[1] - ps.k2*y[1] - ps.k3*y[1]
104     dy[2]    =  ps.k1*y[1]
105     dy[3]    =  ps.k2*y[1]
106     nothing
107 end
108
109
110 # A goes to B which goes to __ or C
111 function A1B1_orC(ps::Wnt3p{T}, t, y, dy) where T<:Real
112     dy[1]    = -ps.k1*y[1]
113     dy[2]    =  ps.k1*y[1] - ps.k2*y[2] - ps.k3*y[2]
114     dy[3]    =  ps.k2*y[2]
115     nothing
116 end
117
```

```
118
119 # Arbitrary stoichiometry, node one -> node 2 -> node 3, with 4
        species
120 function asN1N1N4s(ps::Wnt2p{T}, t, y, dy) where T<:Real
121     #=A+2B->C+2D->2A+D=#
122     a = [1,0,2]
123     b = [2,0,0]
124     c = [0,1,0]
125     d = [0,2,1]
126
127     v1 = ps.k1 * y[1]^a[1] * y[2]^b[1] * y[3]^c[1] * y[4]^d[1]
128     v2 = ps.k2 * y[1]^a[2] * y[2]^b[2] * y[3]^c[2] * y[4]^d[2]
129
130     dy[1]   = -(a[1]-a[2])*v1 - (a[2]-a[3])*v2
131     dy[2]   = -(b[1]-b[2])*v1 - (b[2]-b[3])*v2
132     dy[3]   = -(c[1]-c[2])*v1 - (c[2]-c[3])*v2
133     dy[4]   = -(d[1]-d[2])*v1 - (d[2]-d[3])*v2
134     nothing
135 end
136
137
138 # A goes to B which goes to A which goes to B... Also, B can go to C
139 function A2B1C(ps::Wnt3p{T}, t, y, dy) where T<:Real
140     dy[1]   = -ps.k1*y[1] + ps.k2*y[2]
141     dy[2]   =  ps.k1*y[1] - ps.k2*y[2] - ps.k3*y[2]
142     dy[3]   =  ps.k3*y[2]
143     nothing
144 end
145
146 # A -> B -> C -> __
147 function A1B1C1(ps::Wnt3p{T}, t, y, dy) where T<:Real
```

```
148     dy[1]    = -ps.k1*y[1]
149     dy[2]    =  ps.k1*y[1] - ps.k2*y[2]
150     dy[3]    =  ps.k2*y[2] - ps.k3*y[3]
151     nothing
152 end
```

A.4 Package Installer

This program installs the various packages needed to run the plotting programs.

```
1  # Code to get most of the important packages installed for members
       of
2  # Dr. Transtrum's team
3
4  # Initialize Package Path
5  import Pkg
6  # Uncomment next line to put packages into "Modeling" environment
7  # Pkg.activate("Modeling")
8  # Import registered packages
9  Pkg.add("PyCall")
10 Pkg.add("PyPlot")
11 Pkg.add("DifferentialEquations")
12 Pkg.add("Sundials") # Not required, but useful
13 Pkg.add("Dierckx")  # Not required, but useful
14 # On ubuntu, install hdft-tools first using ``sudo apt-get install
       hdf5-tools``
15 Pkg.add("MAT") # Not required, but useful
16 Pkg.add("ForwardDiff")
17 Pkg.add(Pkg.PackageSpec(url="https://github.com/mktranstrum/
       NumDiffTools.jl"))
18 Pkg.add("Logging")
```

```
19 Pkg.add("Parameters")
20 Pkg.add("JSON")
21 #Install Main Modeling Package
22 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       Models.jl.git"))
23 # Install Geometry Package
24 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       Geometry.jl.git"))
25 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       SmoothApproximations.jl.git"))
26 # Install Fitting Packages
27 # Note that GeodesicLM will need to compile some FORTRAN source
       before working.
28 # ModularLM is a good pure-julia alternative for now that will
       eventually replace GeodesicLM, so we leave this commented out for
        now.
29 # Pkg.add("https://git.physics.byu.edu/Modeling/GeodesicLM.jl")
30 # Install Additional Modeling Packages.
31 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       ModularLM.jl.git"))
32 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       ParametricModels.jl.git"))
33 Pkg.add(Pkg.PackageSpec(url="https://git.physics.byu.edu/Modeling/
       ExampleModels.jl.git"))
34 # Test installation
35 # If this test works, then all the other dependencies should be
       working, too
36
37 # Uncomment the line below this comment to let the program test to
       see if all
```

```
# the custom packes were installed correctly. Note, it may take
    several hours
# for the tests to run.
#Pkg.test("ExampleModels")
```

A.5 Mathematica Based Model Solver

This program (written in Mathematica 12.1) analytically solves the mass action models of a simple path networks with three or fewer species and one or two parameters, then finds the models that describe the boundaries of the model manifold by taking parameters to their limits. The reaction rates can be adjusted to take ratios into account. Unfortunately, many, if not most, models cannot be solved by this or perhaps any solver. That said, many of the models in this report were successfully and easily analyzed using this solver, so it can be very useful when it does work.

```
In[93]:= (* N1\[Rule]N2 2 Species *)
(* Two nodes, two species *)
(*2A\[Rule]B*)
Clear["`*"]
aa = {2,0}
bb = {0,1}
v[1] :=k[1]*a[t]^aa[[1]]*b[t]^bb[[1]]
sol=DSolve[{
a'[t]==-v[1]*(aa[[1]]-aa[[2]]),
b'[t] == -v[1]*(bb[[1]]-bb[[2]]),
a[0]==a0,b[0]==b0},
{a[t],b[t]},t];
{a,b}={sol[[1]][[1]][[2]],sol[[1]][[2]][[2]]}

p = 1; (* Number of parameters *)
```

```
dim1 = Array[0,{p,2}];

lims = {0,Infinity};
For[ki=1,ki<= p,ki++,
For[i = 1, i <=  Length[lims],i++,
dim1[[Mod[ki,p]+1,i]]=Limit[{a,b},k[Mod[ki,p]+1]->lims[[i]],
   Assumptions->{t>0}];
Print["Position ",Mod[ki,p]+1,i,"  ",k[Mod[ki,p]+1],"->",lims[[i]],"
    gets ",dim1[[Mod[ki,p]+1,i]]]]];

dim1;









(* N1\[Rule]N2\[Rule]N3 3 Species *)
(* A\[Rule]B\[Rule]C *)
Clear["`*"]

(*Coefficient vectors*)
aa = {1,0,0};
bb = {0,1,0};
cc = {0,0,1};
Print["aa = ",aa]
Print["bb = ",bb]
Print["cc = ",cc]

```

```
(*Reaction rates*)
v[1] :=k[1]*a[t]^aa[[1]]*b[t]^bb[[1]]*c[t]^cc[[1]]
v[2] :=k[2]*a[t]^aa[[2]]*b[t]^bb[[2]]*c[t]^cc[[2]]

(*Solve the mass action model*)
sol=DSolve[{
a'[t]==-v[1]*(aa[[1]]-aa[[2]])-v[2]*(aa[[2]]-aa[[3]]),
b'[t] == -v[1]*(bb[[1]]-bb[[2]])-v[2]*(bb[[2]]-bb[[3]]),
c'[t] == -v[1]*(cc[[1]]-cc[[2]])-v[2]*(cc[[2]]-cc[[3]]),
a[0]==a0,b[0]==b0,c[0]==c0},
{a[t],b[t],c[t]},t];
{a,b,c}={sol[[1]][[1]][[2]],sol[[1]][[2]][[2]],sol[[1]][[3]][[2]]}


(* Number of parameters *)
p = 2;
(* Arrays to store the new equations as they are found *)
dim1 = Array[0,{p,2}] ;
dim0=Array[0,{p,2,2}] ;
(* The limits that you want to take the parameters to *)
lims = {0,Infinity};

(* For every parameter that exists... *)
For[ki=1,ki<= p,ki++,
(* ...and for every limit that you want to take... *)
For[i = 1, i <=  Length[lims],i++,
(* ...take parameter Mod[ki,p]+1 (the Mod is to help switch which
   parameter has the limits taken first) to limit i and store it
   accordingly...*)
dim1[[Mod[ki,p]+1,i]]=Limit[{a,b,c},k[Mod[ki,p]+1]->lims[[i]],
   Assumptions->{t>0}];
```

```
(* ...then print the result... *)
Print["Position ",Mod[ki,p]+1,i,"  ",k[Mod[ki,p]+1],"->",lims[[i]],"
    gets ",dim1[[Mod[ki,p]+1,i]]];
(* ...then, for every limit that you want to take... *)
For[j = 1, j<= Length[lims], j++,
(* ... take the other parameter to limit j, storing the result
   accoringly... *)
dim0[[Mod[ki,p]+1,i,j]]=Limit[dim1[[Mod[ki,p]+1,i]],k[Mod[ki+1,p
   ]+1]->lims[[j]],Assumptions->{t>0}];
(* ...and print out the the result. *)
Print["Position ",Mod[ki,p]+1,i,j," ",k[Mod[ki,p]+1],"->",lims[[i]],
   " and ",k[Mod[ki+1,p]+1],"->",lims[[j]]," gets ",dim0[[Mod[ki,p
   ]+1,i,j]]]]]]
dim1;
dim0;
```

A.6 Hasse Diagrams and Network Structures

The Hasse diagrams in this report, with the exception of the Hasse diagram for $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$ and the Hasse diagrams in the introduction, were created using a free account on the website Lucidchart.com. The website is targeted towards people who want to make professional looking flowcharts. The company running the website gave me a temporary premium account as part of a promotion, and I created the Hasse diagram for $A \xrightarrow{k_1} B \xrightarrow{k_2} C \xrightarrow{k_3} D$ during that time. Without a premium account, I would have exceeded the permitted number of objects in my flowchart while making such a large Hasse diagram.

The network structures in this report and the Hasse diagrams in the introduction were created using a licensed copy of Microsoft PowerPoint for Microsoft 365.