2018-07-01

An Algorithm for Symbolic Computing of Singular Limits of Dynamical Systems

Dane Jordan Bjork
Brigham Young University

An Algorithm for Symbolic Computing of Singular Limits of Dynamical Systems

Dane Jordan Bjork

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Sean C. Warnick, Chair
Mark K. Transtrum
Michael Jones

Department of Computer Science

Brigham Young University

ABSTRACT

An Algorithm for Symbolic Computing of Singular Limits of Dynamical Systems

Dane Jordan Bjork
Department of Computer Science, BYU
Master of Science

The manifold boundary approximation method, *MBAM* is a new technique used in approximating systems of equations using parameter reduction. This method and other approximation methods are introduced and described. Several current issues in performing MBAM are discussed in further detail. These issues significantly slow down the process of MBAM and create a barrier of entry for those wishing to use the method without a strong background in mathematics. A solution is proposed to automatically reparameterize models and evaluate specific types of variables approaching limits – significantly speeding up the process of MBAM. An implementation of the solution is discussed.

Keywords: Approximation, Automation, Parameter Reduction, MBAM

ACKNOWLEDGMENTS

I would like to recognize the dedicated support of Dr. Mark Transtrum and Dr. Sean Warnick. Not only have they spent many hours helping me understand and work through difficult problems, but they have always been warm and welcoming when I would come back with even more questions. I would also like to recognize the sacrifices my parents made to help me get to where I am today.

Table of Contents

List of Figures

Chapter 1

Introduction

Models and computational simulations are growing in their capabilities to emulate real world behaviors [16]. These systems can take various shapes and sizes. For example, systems biology research to understand the dynamics of the musculoskeletal system [9] or cellular pathways [12, 13] is being used to understand and improve treatments for various disabilities. Other examples include behaviors as detailed as molecular dynamics [10, 22], or as broad as economics [2], human behavior [4], and climate changes [17]. These models are often built by combining many microscopic details. While building models with this constructionist approach can yield effective results, complications can arise such as the cost of computational power, difficulty fitting the model to data [20], or even numerical instabilities [1].

Typically, model classes of complex phenomena are described by parameterized sets of equations. Often, these are differential equations and the complexity of the set is characterized by the number of parameters used to distinguish one model in the set from another. Varying the values of these parameters lead to different error values on the observed data. Having too few parameters leads to higher error costs, while too many parameters leads to inaccurate predictions. An accurate model, therefore, requires a sufficient number of parameters to best fit the data and properly predict the behavior. Moreover, deriving smaller, simpler systems grants a researcher the ability to understand key aspects to different behaviors of her nominal system [7].

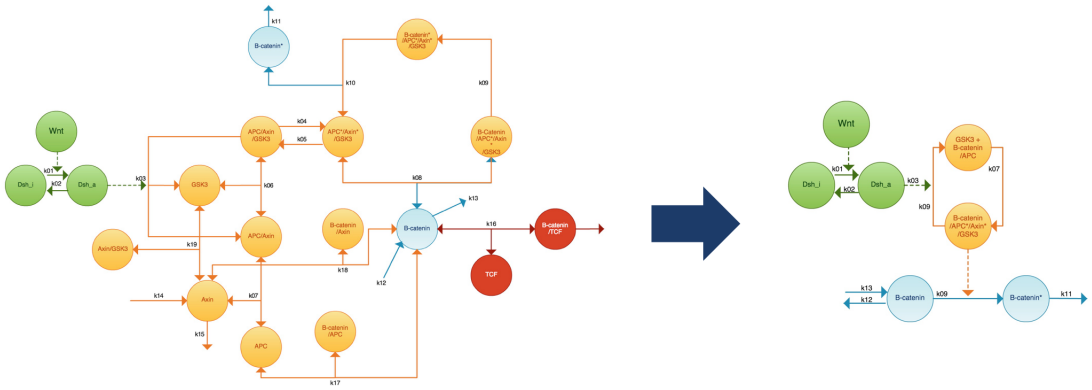


Figure 1.1: The Manifold Boundary Approximation Method has been used previously to reduce a complex systems biology network, the Wnt pathway. As seen above, the final results reduced a complex network to a fairly simple model still capable of capturing the observed dynamics of the complex system.

1.1 The Manifold Boundary Approximation Method

The Manifold Boundary Approximation Method, *MBAM*, is a new approximation method based on information geometry. Note that a set of systems, in this work, is defined as a set of equations with unspecified values for the parameters. A single system is defined by a choice of parameter values within a set of systems. This method is capable of iteratively reducing a set of systems with each iteration returning a simplified set of systems containing one less parameter than the previous. These new set of systems reveal the emergent behavior that can be hidden in the larger sets of systems [18]. The more parameters a system has, the more useful MBAM can be. As the number of parameters increase, the model often becomes more *sloppy*. Sloppiness is a characteristic of a model to have an insensitivity to large variations of certain parameters and extreme sensitivity to other *stiff* parameter combinations. Sloppiness of models is a strong driving force for the methodology of MBAM. The sloppy behavior of parameters can be shown in the eigenvalues of the Hessian (second derivative) matrix of the observation function. The eigenvectors of the Hessian with small eigenvalues are the sloppy directions, and the parameters to remove first from the model.

There are three requirements which need to be met in order to preform MBAM on a system. First, the model must be formulated as a parameterized statistical model. Second, the system must be twice differentiable with respect to its parameters. Third, the effective dimensionality of the modeled behavior must be much less than the number of parameters [19]. These requirements often occur naturally when creating models, making MBAM a very versatile tool in approximation. The fundamentals of MBAM are discussed in further detail in Sec. 2, with an example of MBAM's workflow found in Sec. 2.4.

Chapter 2

An In-Depth Overview of MBAM

MBAM uses information geometry to approximate sets of systems. This section goes into a deeper background on the information geometry terms necessary to understand MBAM. This includes the model manifold, what the boundaries of the model manifold represent, and how those boundaries are approached using geodesic calculations. Finally, the difficulties in using MBAM are discussed.

2.1 The Model Manifold

The objective of a learning problem is to find a system, out of a set of systems, that best fits data according to a norm. Each of the individual systems is chosen from the set of systems by using different values for its parameters. The model manifold characterizes this entire set of systems. In both sections of the problem formulation it is assumed that the model manifold is given:

$$f : \mathbb{R}^N \to \mathbb{R}^M,$$

where f is a map from parameters, $\theta \in \mathbb{R}^N$, to a point in "data space", $\mathbb{R}^M$. To better understand what is encompassed in the model manifold, two examples are given.

Example: Exponential Decay

The exponential decay model leads to a natural understanding of the model manifold. Given an original set of systems following the form in Fig. 2.1, a model manifold can be created. To find the manifold, the solution of the system must first be calculated.

Original Set of Systems

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -\theta_1 & 0 \\ 0 & -\theta_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Solution

$$y(t) = e^{-\theta_1 t} x_1(0) + e^{-\theta_2 t} x_2(0)$$

with $x_1(0) = x_1^*, x_2(0) = x_2^*$

System Solutions for Values of θ

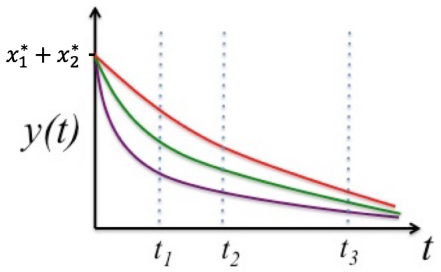


Figure 2.1: This solution represents an entire set of systems. To select a single system, one must choose values for θ_1 and θ_2. These individual systems can then be characterized by their values at specific times, t_1, t_2, and t_3. This characterization can be found for any choice of θ_1 and θ_2. Finding and plotting this characterization for *every* choice of θ_1 and θ_2 creates the model manifold.

The model manifold for $y(t) = e^{-\theta_1 t} x_1^* + e^{-\theta_2 t} x_2^*$

or, equivalently the set of systems in at $t_1 = 1, t_2 = 2, t_3 = 3$.

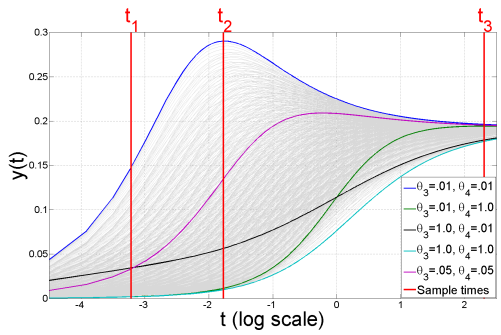
Figure 2.2: This manifold is a mapping $f : \mathbb{R}^N \to \mathbb{R}^M$, where N is the number of parameters in the system, and M the number of time points used to characterize the system. These time points are often the time values for which the user has data. The model manifold's shape is independent from the data points used, although, there must be a sufficient number of data points. Each system solution for any value of $\theta \in \mathbb{R}^N$ is represented by an individual point on the model manifold – giving a geometric representation of the entire set of systems.

Example: Enzyme Reaction

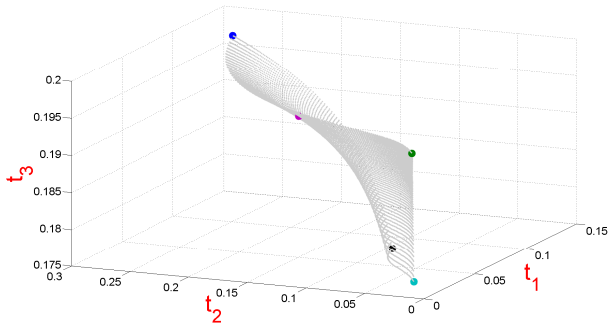
Using the nonlinear dynamics of an enzyme reaction, an explicit solution can be acquired of the form:

$$y(t, \theta) = \frac{\theta_1(t^2 + \theta_2 t)}{t^2 + \theta_3 t + \theta_4} \tag{2.1}$$

Again, it can be seen that this represents a set of systems, parameterized by the values $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$. In this example, all four parameter values could be varied to create a four-dimensional manifold, however, for visualization purposes only two values will be varied. The same principles of the four-dimensional manifold apply to this two-dimensional manifold.



An original set of models parameterized by θ. Each curve is a system defined by various values for θ_3 and θ_4. With M data points, t_1, t_2, and t_3, each curve can be characterized as a mapping from the two parameter values to its values at the three times.



There are two parameters varied at three times which will create a two-dimensional manifold embedded in a three-dimensional space, $f : \mathbb{R}^2 \to \mathbb{R}^3$. Notice how the colored curves in the first image map to individual points on the manifold.

Figure 2.3: These figures show the transition from a common view of a set of systems, i.e. a set of curves in a two dimensional plane, to a model manifold mapped into data space.

2.2 Manifold Boundaries

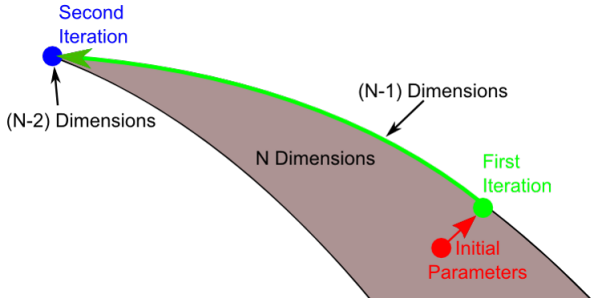


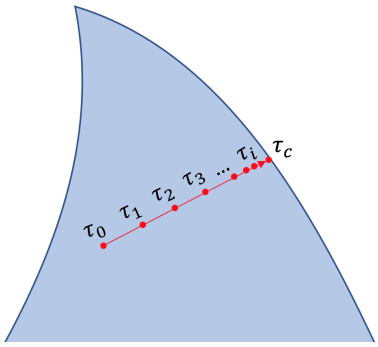
Figure 2.4: Close-up view of a corner on an example model manifold [19]. Starting at some initial N parameter values, as seen in red, the model can be pushed toward a boundary. This boundary, as seen in green, characterizes a new set of systems where one of the N parameters has reached a value of zero, and now contains $N-1$ parameters. Furthermore, another set of systems containing $N-2$ parameters can be defined by pushing the new set of systems to a corner. Thus each boundary represents a single parameter reaching zero.

When using MBAM, it is assumed only bounded model manifolds will be used, (i.e. the entire model manifold fits inside a large enough, but finite, sphere). Notice that parameter values corresponding to points on the boundaries of the model manifold are approaching a limit at zero or infinity. This can be seen in the blue curve with θ_3 and θ_4 approaching zero as seen in Fig. 2.3, and the blue point representing that curve is found on a corner – or the intersection of two boundaries – of the model manifold. These boundaries can be found starting at any point on the model manifold through the calculation of a geodesic. Calculating the geodesic is a solved problem that returns the values of the parameters approaching a boundary and the direction those parameter values are moving. These results allow the user to identify which parameters are approaching limiting values. Once the those parameters and the corresponding limits are understood, the system must be reorganized in such a way that when these limits are evaluated the system dynamics are not lost, yet the system contains one less parameter. This new set of systems with $N-1$ parameters now corresponds only to the boundary of the original model manifold.

2.3 The Geodesic

Traversing the model manifold in an efficient manner is a solved problem which uses the calculation of a *geodesic*. Geodesics are a generalization of straight lines for curved surfaces and can be specified with an initial point on the model manifold and a direction [19]. Parameter values can be calculated from the points along a geodesic. Therefore, as a geodesic is calculated towards a boundary, the parameter values moving towards that boundary can be returned. The geodesic may never reach the boundary of the manifold, however, the boundary it is heading towards can be inferred by evaluating the parameter values and observing their evolution along the geodesic.

The parameter values can be calculated as the geodesic is traversed. These parameter values lead the user to understand which parameters are approaching which limits. In Fig. 2.5, it can be inferred that θ_2 is approaching infinity. In practice, moving along the



$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix} = \begin{cases} \begin{bmatrix}.5\\ 2.5\end{bmatrix} \text{ at } \tau_0 \\ \begin{bmatrix}.49\\ 4.02\times 10^{10}\end{bmatrix} \text{ at } \tau_i \\ \begin{bmatrix}.49\\ \infty\end{bmatrix} \text{ at } \tau_c \end{cases}$$

Figure 2.5: The geodesic uses a different coordination system than simply varying parameter values. However, the parameter values can be found as the geodesic is calculated. As the initial point of the geodesic moves towards a boundary at τ_c, at least one parameter value will approach a limit.

geodesic returns a larger set of parameter values in which case the inference of the parameters approaching their limits is fairly obvious. Therefore, as τ approaches τ_c, some parameter values will approach limits. Upon evaluating those limits, τ_c is reached and the manifold boundary is revealed.

2.4 Using MBAM

The focus of this thesis is increasing the speed and usability of the manifold boundary approximation method. This will be done by developing an algorithm for automating the pieces of MBAM that remain arduous in the current workflow. MBAM's workflow is found in Fig. 2.6 and is shown as a loop, where each iteration through this loop outputs a new set systems containing one less parameter than the previous system. By removing one parameter from the set of systems with each iteration, reduction on large sets of systems can grow tedious or impossible to perform by hand. The following example further illustrates the flow of MBAM.

To perform one cycle of MBAM, the following steps must be followed as shown in Fig. 2.6:

1. Initialize a model.

2. Calculate the geodesic on the model.

3. Infer the parameter limits from the geodesic.

4. Define a new set of parameters from the original parameters and the inferred geodesic limits.

5. Plug the new parameters into the system.

6. Symbolically evaluate the limits on the system to return a new system containing one less parameter.

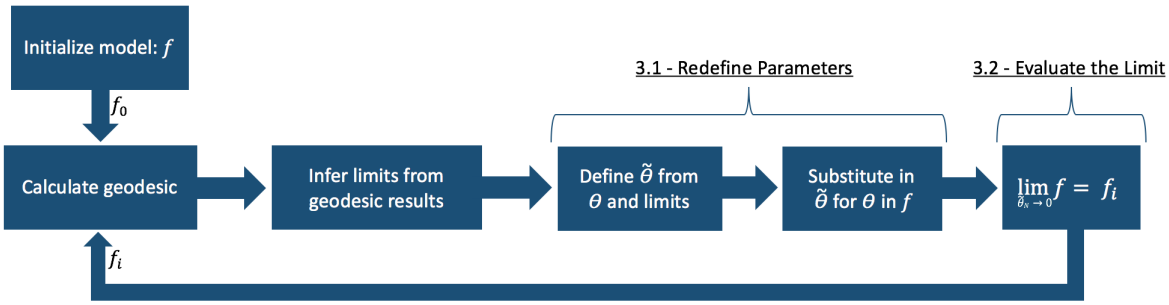


Figure 2.6: The major steps for approximating a computational model with MBAM. A new system f_i is created for each iteration, having one less parameter than the input system f_{i-1}. Systems are represented by f_i and the parameters are represented by θ.

2.4.1 MBAM Flow Example

Step 1 - Initialize Model

To begin approximation using MBAM, the initial system f_0 must be defined. This system must meet three requirements [19]:

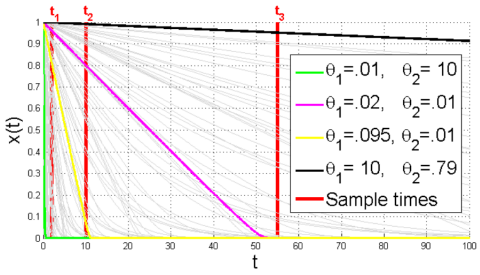
1. The model must be formulated as a parameterized statistical model.

2. The system must be twice differentiable with respect to its parameters.

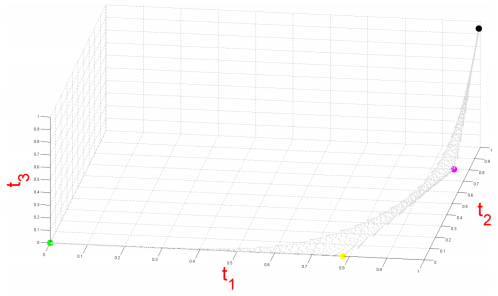
3. The effective dimensionality of the modeled behavior must be much less than the number of parameters.

In this example, the set of systems Eq. 2.2 are given from the dynamics of the Michaelis Menten Reaction. While this system may not have a much higher dimensionality of the modeled behavior with respect to the number of parameters, its simplicity aids in understanding this process.

$$\dot{x} = -\frac{\theta_1 x}{\theta_2 + x} \tag{2.2}$$



(a) This initial set of equations corresponding to Eq. 2.2. Each curve represents an individual system with different value choices for the parameters θ_1 and θ_2.



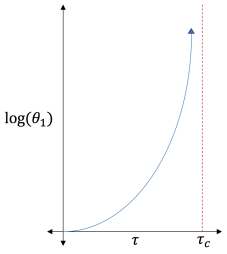
(b) The model manifold corresponding to Eq. 2.2. For each curve in Fig. 2.7a there is a single point on this manifold. This is highlighted with a colored curve and its corresponding point on the model manifold highlighted the same color.

Figure 2.7: It is often the case that the user of MBAM does not create figures similar to Fig. 2.7a & 2.7b. However, they are helpful for understanding the underlying mechanisms that drive MBAM.

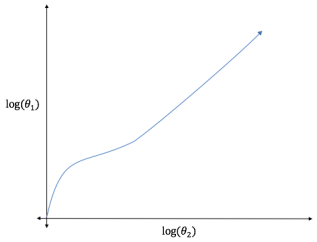
Now that the model manifold, $f : \mathbb{R}^N \to \mathbb{R}^M$, has been created, a geodesic can be calculated to understand which boundary will be used to approximate the system. In practice the time stamps, t_1, t_2, and t_3 will be associated with the time points at which data exists for the model. Because the number of time points will be greater than the number of parameters, the collected data point will always be a higher dimensionality than the model manifold. This difference in dimensionality often causes the point in data space representing the data to lie off the model manifold. This can easily be seen if the model is incapable of fitting the data perfectly – showing that no parameter combination in the set of systems can reach the data. Therefore, better fits in parameter values will naturally approach boundaries.

Step 2 - Calculate the Geodesic

In order to approach boundaries on the model manifold the geodesic will be calculated. The work for computing geodesics is a previously solved problem [21]. Parameter values along the geodesic can be calculated while the geodesic is traversing the model manifold towards the boundary. Once calculated, graphs similar to those found in Fig. 2.8 would be created. In which case, the user infers which parameter values are approaching which limits.



(a) As τ approaches τ_c, the value for θ_1 grows towards infinity.



(b) Plotting the values of θ_1 vs θ_2 reveal both approaching infinity at an equal rate, showing that the ratio of these two parameter values become a new finite parameter value.

Figure 2.8: Example results from calculating the geodesic on Eq. 2.2.

Step 3 - Infer Parameter Limits

When the data from the geodesic has returned, it becomes the user's responsibility to determine which parameters are approaching which values. This is a fairly trivial step in MBAM. As seen in Figures 2.8a & 2.8b, the data clearly shows that both θ_1 and θ_2 are approaching their limits at infinity. In the case of this example, the set of parameters $\theta_{lim} = \{\theta_1 \to \infty, \theta_2 \to \infty\}$. While inferring parameter values approaching a limit from a geodesic is currently left to the user, this process will be a fairly trivial process to automate.

Step 4 - Redefine the Parameters

Now the user has the original set of systems, Eq. 2.2, the original parameters, $\theta = \{\theta_1, \theta_2\}$, and the set of parameters approaching a limit, $\theta_{lim} = \{\theta_1 \to \infty, \theta_2 \to \infty\}$. With these three

inputs the new set of parameters, $\tilde{\theta} \in \mathbb{R}^N$, must be defined from θ such that one parameter value is approaching zero while all other parameter values are finite.

For this example, one must understand that both θ_1 and θ_2 are approaching infinity at the same rate. This allows for the user to first infer that a new parameter can be defined as the ratio of those two values:

$$\tilde{\theta}_{1/2} = \frac{\theta_1}{\theta_2}.$$

Thus, $\tilde{\theta}$ can be initialized with $\tilde{\theta} = \{\tilde{\theta}_{1/2}\}$. Because $|\tilde{\theta}| \neq |\theta|$ another parameter is necessary for $\tilde{\theta}$ to substitute θ in the original system. Knowing that the final parameter value must be approaching its limit at zero, the parameter can be represented by the inverse of one of the parameters approaching infinity. In this case

$$\varepsilon = \frac{1}{\theta_1}.$$

This gives the new set of parameters

$$\tilde{\theta} = \{\tilde{\theta}_{1/2},\ \varepsilon\} = \left\{\frac{\theta_1}{\theta_2}, \frac{1}{\theta_1}\right\},$$

which is bijective with the original set of parameters, θ,

$$[\theta_1,\ \theta_2]^T = \theta \leftrightarrow \tilde{\theta} = [\tilde{\theta}_{1/2},\ \varepsilon]^T.$$

While this example is fairly trivial, as a set of systems increases in size, this process quickly becomes non-trivial.

Step 5 - Evaluate $\tilde{f} = f(f_{\tilde{\theta}}^{-1}(\tilde{\theta}))$

Now that $\tilde{\theta}$ has a definition, the next step is to plug those parameter values in for the original parameter values. To begin, solve for the original θ values using the values for $\tilde{\theta}$:

$$\varepsilon = \frac{1}{\theta_1} \Rightarrow \theta_1 = \frac{1}{\varepsilon}$$

$$\theta_{1,2} = \frac{\theta_1}{\theta_2} \Rightarrow \theta_2 = \frac{\theta_1}{\theta_{1/2}} = \frac{1}{\varepsilon\,\theta_{1/2}}$$

With these equivalences, the parameterization on the original set of systems, $\tilde{f}$, can be defined through simple substitution.

$$\tilde{f} = f(f_{\tilde{\theta}}^{-1}(\tilde{\theta})) = \frac{-(\frac{1}{\varepsilon})x}{(\frac{1}{\varepsilon\,\theta_{1/2}}) + x}$$

Step 6 - Evaluate the Limit for ε

Evaluating $\lim_{\varepsilon^+ \to 0} \tilde{f}$ is often a non-trivial process. In the case of this example, tricky algebraic manipulation must be used in order to prevent division by zero. The trick used here is simply multiplying the system by $\frac{\varepsilon\,\theta_{1/2}}{\varepsilon\,\theta_{1/2}} = 1$. This manipulation removes the error, without having changed any of the system's dynamics.

$$\left(\frac{\varepsilon\,\theta_{1/2}}{\varepsilon\,\theta_{1/2}}\right)\frac{-(\frac{1}{\varepsilon})x}{(\frac{1}{\varepsilon\,\theta_{1/2}}) + x} = \frac{-\theta_{1/2}x}{1 + \varepsilon\,\theta_{1/2}x}$$

Finally, the limit can be evaluated as $\varepsilon \to 0$. This results in an approximated system which contains one fewer parameters than the original

$$\lim_{\varepsilon \to 0} \frac{-\theta_{1/2}x}{1 + \varepsilon\,\theta_{1/2}x} = -\theta_{1/2}x = f_1.$$

It is often the case that evaluating $\lim_{\varepsilon \to 0} \tilde{f}$ causes algebraic errors which will need symbolic manipulation to overcome. Thus, automating this process is also non-trivial.

Chapter 3

Problem Formulation

MBAM is a very powerful tool in the world of approximation. Models with higher numbers of parameters are typically more sloppy and can therefore use MBAM more effectively. However, the methods used to preform MBAM are highly specialized. This creates a barrier for those without a background in information geometry to preform MBAM. Sloppy models are frequently found in the field of systems biology and were, in fact, developed by the field [5, 6, 8]. With this barrier in place, few may be able to perform MBAM on their systems. Therefore, in order for MBAM to be applied to various fields and be used to its fullest potential, this barrier must be removed.

Even when the underlying methods for MBAM are understood, reducing a complicated model takes a significant amount of time. Attempting to take singular limits by hand is very time intensive due to the algebraic manipulations that must be performed with a large set of equations such that the dynamics of the system are preserved. Solving these limits often has a puzzle-like nature. As a set of systems grows into hundreds of differential equations and thousands of parameters, or even larger, it becomes clear that taking these limits are troublesome and even impossible for the user to preform by hand. The problem of singular limits in MBAM is further described in this section and consists of redefining parameters and evaluating the potential singular limit created by the parameter redefinition. By automating the solution to this problem, MBAM can be more widely used and used more efficiently.

3.1 Evaluating Singular Limits Automatically

3.1.1 Redefine Parameters

Given a model manifold (Sec. 2.1), $f : \mathbb{R}^N \to \mathbb{R}^M$, and a specified boundary on that manifold (Sec. 2.2) as often given through the results of a geodesic calculation (Sec. 2.3), find a change of coordinates:

$$f_{\tilde{\theta}} : \mathbb{R}^N \to \mathbb{R}^N$$

such that the N^{th} element of the new coordinate,

$$\tilde{\theta}_N = f_{\tilde{\theta}}(\theta)_N = 0 \text{ and } ||\tilde{\theta}|| < \infty$$

for all points in the specified boundary.

3.1.2 Evaluate the Limit

Given a model manifold (Sec. 2.1), $f : \mathbb{R}^N \to \mathbb{R}^M$, and $f_{\tilde{\theta}}$ as defined in Sec. 3.1.1, find an analytic expression characterizing the boundary:

$$f(f_{\tilde{\theta}}^{-1}(\{\tilde{\theta}|\tilde{\theta}_{N^+} \to 0\})).$$

*Note that simply substituting in the $\tilde{\theta}$ values for θ into the set of equations can lead to division by zero or other intractable algebraic problems which must be resolved.

3.2 Solution Algorithm

The solution to problems found in Sec. 3.1.1 & 3.1.2 deal with evaluating the results of a geodesic calculation on a given model. This chapter presents an algorithm capable of evaluating geodesic results. Following chapters analyze that algorithm and its various sub-algorithms.

Algorithm 1 Evaluate A Given Limit on a Model

Input: L, a mapping of K parameters to their limits (Geodesic Results).
Input: M_N, a full model containing N total parameters.
Output: M_{N-1}, a full model containing $N-1$ total parameters.

```
θ̃_list ⇐ Reparameterize(L, M)
for θ̃ in θ̃_list do
    M_θ̃ ⇐ Substitute θ̃ into M_N.
    M_{N-1} ⇐ EvaluateEpsilon(M_θ̃)
    if IsValid(M_{N-1}) then
        return M_{N-1}
    else
        M_VL ⇐ VariableLimit(M_θ̃)
        M_{N-1} ⇐ EvaluateEpsilon(M_VL)
        if IsValid(M_{N-1}) then
            return M_{N-1}
        end if
    end if
end for
return M
```

Chapter 4 discusses the first sub-algorithm dealing with redefining parameters and solving the problem defined in Sec. 3.1.1. The second sub-algorithm is found in Chapter 5 and deals with solving the problem defined in Sec. 3.1.2 in which evaluating $\tilde{\theta}_{N^+} \to 0$ is non-trivial. An implementation of this algorithm is defined in Appendix A, with that implementation's documentation found in Appendix B. Results returned from this algorithm are found in Appendix C.

Note: the parameter referred to as $\tilde{\theta}_N$ in the problem statement will be called ε in the following chapters.

Chapter 4

Redefining Parameters

Algorithm 2 Reparameterize

Input: L, a mapping of K parameters to their limits.
Input: M_N, a full model containing N total parameters.
Output: $\tilde{\theta}_{list}$, a list of all potential valid reparameterizations.

$L_{\text{key}} \Leftarrow \text{DefineLimitKey}(L)$
$T \Leftarrow \text{GetAllTemplates}()$
$T_F \Leftarrow \text{FilterTemplates}(L_{\text{key}}, T)$
$\varepsilon_T, F_T \Leftarrow \text{FillTemplates}(T_F, L)$
$\tilde{\theta}_{list} \Leftarrow \text{CreateFTildeList}(\varepsilon_T, F_T, L)$
return $\tilde{\theta}_{list}$

4.1 Defining a Limit Key

A limit key is generated to filter out unnecessary templates. To generate this limit key, a key legend is defined. The key legend is stored in the database and is a mapping from limit type to key index. Because new limit types can be found through the process of MBAM having an extendable method for filtering templates is necessary. Thus, the key legend must not be hard coded and should be user-defined.

Key legends are best understood with an example. Given results from a geodesic

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

and a key legend

$$\{\text{“zero”} : 0, \;\; \text{“infinity”} : 1\},$$

the limit key can be defined

$$[\text{Number of Zeros, Number of Infinities}] = [0,\ 2].$$

However, if the key legend were

$$\{\text{“zero”} : 1,\ \ \text{“infinity”} : 0\},$$

the corresponding limit key for this geodesic result would be

$$[\text{Number of Infinities, Number of Zeros}] = [2,\ 0].$$

4.2 Getting Templates From Database

4.2.1 Motivation

As one works with MBAM, it becomes clear that the definition of $f_{\tilde{\theta}}$ has a repetitive nature, leading to the idea that limits can be evaluated using templates. For example, when using Michaelis-Menten models there are two commonly used pieces of the model. The first piece is where enzyme A promotes enzyme B,

$$\dot{B} = \frac{(B_{\text{total}} - B)Ak_1}{(B_{\text{total}} - B) + k_2}. \tag{4.1}$$

The second piece is when enzyme A inhibits enzyme B,

$$\dot{B} = -\frac{ABk_1}{B + k_2}. \tag{4.2}$$

For Eq. 4.1, a valid $f_{\tilde{\theta}}$ for the limit $k_1, k_2 \to \infty$ is:

$$\varepsilon = \frac{1}{k_1},$$

$$K_{12} = \frac{k_1}{k_2}.$$

For Eq. 4.2, a valid $f_{\tilde{\theta}}$ for the limit $k_1, k_2 \to \infty$ is:

$$\varepsilon = \frac{1}{k_1},$$

$$K_{12} = \frac{k_1}{k_2}.$$

While Eq. 4.1 and Eq. 4.2 have different behaviors, the valid reparameterizations for these two equations are the same for the given limit. In fact, these valid reparameterizations could work in many other models types as well. While there are valid reparameterizations that are repeated frequently, new valid reparameterizations can be found as new model types are approximated with MBAM. Understanding this repetition leads to the idea of using a template to represent $f_{\tilde{\theta}}$. These templates are stored alongside additional information in one object refered to as a **??**.

4.2.2 Template Objects

To automate the reparameterization process, valid reparameterizations are stored as template objects. As the user finds new ways of defining reparameterizations, new template objects are created and stored in the database. Template objects are composed of four pieces:

- Label
- Key
- Model Class
- Template

The *label* in the template object defines how the resulting parameter will be referenced in the resulting model. If the template evaluates to a finite value, this label will be user defined. If the template evaluates to zero, the label will be automatically recorded as 'epsilon'.

The *key* of the template object is defined using the template key found in Sec. 4.1. The number of limit types in the given template is counted, and following the template key, a key for the specific template object is created.

Model class refers to the type of model being reduced. As the number of template objects in the database increases, this can act as an additional filter – preventing the algorithm from using any template objects the user knows will not result in a valid model.

Finally, the template object contains the *template*. The template is a representation of $f_{\tilde{\theta}}$ with place-holders for the parameters. Each place-holder corresponds to a limit type. A template must contain at least one place-holder. A few template object examples follow (infinity place-holders are represented by ∞_i and zero place-holders are represented by 0_i):

$$\{\text{label} : \text{`epsilon'},\ \text{key} : [0,1],\ \text{model class} : \text{`michaelis-menten'},\ \text{template} : 1/\infty_1\}$$

$$\{\text{label} : \text{`epsilon'},\ \text{key} : [1,0],\ \text{model class} : \text{`michaelis-menten'},\ \text{template} : 0_1\}$$

$$\{\text{label} : \text{`inf1 over inf2'},\ \text{key} : [0,2],\ \text{model class} : \text{`michaelis-menten'},\ \text{template} : \infty_1/\infty_2\}$$

4.3 Filtering Templates

Algorithm 3 Filter Templates

Input: L_{key}, they key corresponding to the geodesic results.
Input: M_{class}, the class of the current model.
Input: T, a list of template objects.
Output: T_F, a list of all potential valid reparameterizations.

```
T_F ⇐ Initialize an empty list.
for Template in T do
  if Template Model Class == M_class then
    Valid ⇐ true
    for Index in L_key do
      if Template Key > L_key at the given Index then
        Valid ⇐ false
      end if
    end for
    if Valid then
      Append Template to T_F
    end if
  end if
end for
return T_F
```

To speed up the reparameterization process, the algorithm automatically filters the templates. Each template has a key corresponding to the limit key. If L_K is the key for the current limit and T_K is the key for the template to be filtered, then T_K will only be used if $T_{Ki} \leq L_{Ki} \; \forall \; i$.

4.4 Filling Templates

Algorithm 4 Fill Templates

Input: T_F, A list of filtered templates.
Output: ε_T, a set of filled templates that evaluate to zero.
Output: F_T, a set of filled templates that evaluate to a finite value.
  for Template **in** T_F **do**
    Fill Template with All Possible Permutations of L.
    if Template Name == 'epsilon' **then**
      Add Filled Templates to ε_T.
    else
      Add Filled Templates to F_T.
    end if
  end for

The parameter values fill the filtered templates. A single template can be filled by different permutations of the parameters approaching their limit. These permutations are then added to their corresponding set based on the template object label. There are two groupings for filled templates: templates that evaluate to zero as the manifold boundary is reached, ε_T, and templates that evaluate to a finite value as the manifold boundary is reached, F_T.

For the case where the geodesic returns the limits $k_1 \to \infty,\ k_2 \to \infty$, some valid filled template sets would be:

$$\varepsilon_T = \left\{\frac{1}{k_1}, \frac{1}{k_2}\right\},$$

$$F_T = \left\{\frac{k_1}{k_2}, \frac{k_2}{k_1}\right\}.$$

4.5 Defining a List of Potentially Valid $f_{\tilde{\theta}}$

To have a valid reparameterization with K parameters approaching their limit, one parameter must be chosen from the group of ε_T, and $K-1$ parameters must be chosen from the F_T. Because different models can have varying structures, each combination following this form is created from the filled template sets. Following along with the sets of filled templates ε_T and F_T defined in the previous section, each potentially valid $f_{\tilde{\theta}}$ would be:

$$\text{A: } \varepsilon = \frac{1}{k_1},\ k_{1/2} = \frac{k_1}{k_2}$$

$$\text{B: } \varepsilon = \frac{1}{k_1},\ k_{2/1} = \frac{k_2}{k_1}$$

$$\text{C: } \varepsilon = \frac{1}{k_2},\ k_{1/2} = \frac{k_1}{k_2}$$

$$\text{D: } \varepsilon = \frac{1}{k_2},\ k_{2/1} = \frac{k_2}{k_1}$$

4.6 Substituting $f_{\tilde{\theta}}$ Into the Model

To substitute $f_{\tilde{\theta}}$ into the model, the limit parameters will need to be solved for within $f_{\tilde{\theta}}$. This evaluation is results in $f_{\tilde{\theta}}^{-1}$, which only contains the new parameters. Each limit parameter will correspond one to one with a value in $f_{\tilde{\theta}}^{-1}$. Using A from the example in Sec. 4.5, the $f_{\tilde{\theta}}^{-1}$ can be defined as

$$\text{A: } k_1 = \frac{1}{\varepsilon},\ k_2 = \frac{1}{\varepsilon k_{1/2}}.$$

Once in this form, simply substituting these values for the parameters currently in the model prepares the model to be evaluated as the limit of $\varepsilon \to 0$.

4.7 Algorithm Analysis

Each template will have $T_i \geq 1$ parameter spaces to be filled. Those spaces are filled with every possible permutation of limit parameters $K \geq 1$. The total possible permutations of filled templates is calculated with

$$T_f = \frac{K!}{(K - T_i)!}$$

The templates are sorted into two groups, ε-templates and finite-templates. Each of those groups are then filled creating a total number of filled templates for each group. E_f represents the total number of filled ε-templates. E_i represents the number of parameter spaces to be

filled for a given ε-template. E_N is the number of ε-templates to be filled. T_f represents the total number of filled finite-templates. T_i represents the number of parameter spaces to be filled for a given finite-template. T_N is the number of finite-templates to be filled.

$$E_f = \sum_{i=1}^{E_N} \frac{K!}{(K - E_i)!}$$

$$T_f = \sum_{i=1}^{T_N} \frac{K!}{(K - T_i)!}.$$

A potential reparameterization is composed of one filled ε-template and $K - 1$ filled finite-templates. This gives a total of potential reparameterizations of

$$E_f \left(\frac{T_f!}{(K-1)!(T_f - (K-1))!} \right),$$

after substituting in T_f and E_f into the above equation, it is clear that the worst-case complexity bound is $O(K!)$.

4.7.1 Analysis of Real Use

While the worst-case complexity bound of this algorithm is very inefficient, there are several reasons why this algorithm is still able to be computed quickly. Those reasons include:

- The geodesic often results in only a handful of parameters approaching limits, which places a constraint on K.
- The number of known templates is currently small. Even as this number grows, the templates are filtered by model type. Therefore, E_N and T_N are often constrained.
- Most templates contain a small amount of parameter spaces to be filled. Thus, E_i and T_i are often constrained.
- The algorithm is terminated early if one potential reparameterization successfully approximates the model.

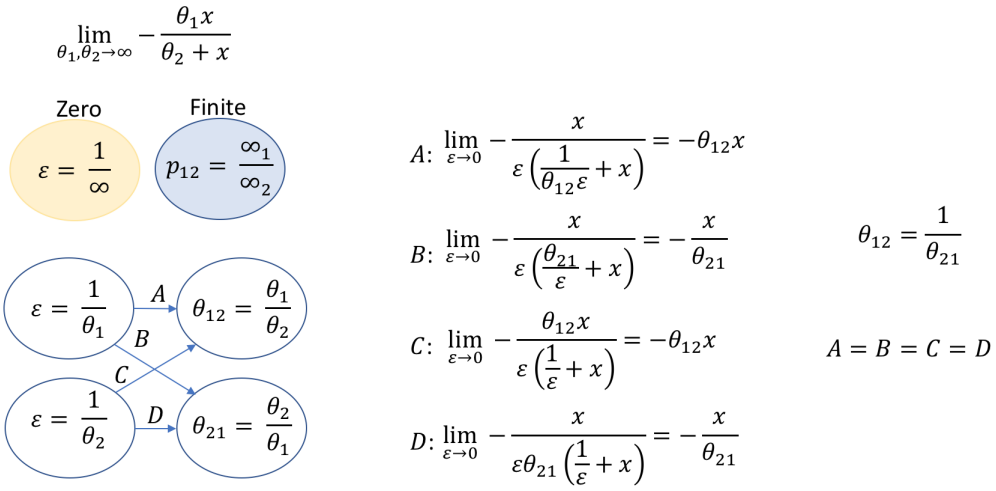


Figure 4.1: This algorithm terminates early if any potential reparameterization successfully approximates the model. This figure shows how one limit and two templates can create four possible reparameterizations (A, B, C, D). Each of these reparameterizations can evaluate to different models. However, notice how these models are all equivalent. Noting this equivalency, the algorithm would terminate after a single attempt, resulting in a model that would be equivalent to using any of the other reparameterizations.

Chapter 5

Evaluating the Limit

5.1 Evaluation

Once the model has been reparameterized, evaluating the limit as $\varepsilon \to 0$ is trivial. Evaluating this limit does not mean that the resulting model fully characterizes the boundary of the model manifold designated by the limits. Checking that the resulting model characterizes this boundary cannot currently be proven and lies outside of the scope of this thesis.

In this algorithm several conditions must be met after evaluating the limit as $\varepsilon \to 0$. Meeting these conditions give a high confidence that the resulting model fully characterizes boundary. The conditions are:

- The model contains $N - 1$ parameters.
- All the K limit parameters are no longer found in the model.
- All values are finite within the model.
- There are no ε values remaining in the model.

These conditions were defined using previous experiences with MBAM and correctly characterizes the manifold boundary for a majority of the limits tested. If any of these conditions are violated it is known that the model is not valid. However, some invalid models may pass all of these conditions. When reparameterization of the model fails any of these conditions, it is often the case that one of the variables is also approaching a limit. These limits depend on the context of the reparameterization and are not determined or inferred by the geodesic. In this solution, several types of variable limits are evaluated. The first limit occurs when

the variable is approaching zero. These are often referred to as *singular limits*. The second limit occurs when the variable is approaching infinity. In either case, the variable and ε are combined to create some new finite and non-zero variable at the limit. In a third case, equations are combined to isolate the singular limit behavior within a variable. Each of these variable limits will be further examined as they appear in the evaluation algorithm.

Algorithm 5 Evaluate Variable Limits

Input: $M_{\tilde{\theta}}$, a full model with N parameters substituted with $\tilde{\theta}$.
Output: M_{VL}, a model with the variable limit evaluated.
$M_{DAE} \Leftarrow \text{ToDAE}(M_{\tilde{\theta}})$
$\varepsilon_{\text{terms}} \Leftarrow \text{FindEpsilon}(M_{DAE})$
$\tilde{M}_{DAE} \Leftarrow \text{CheckLimitZero}(\varepsilon_{\text{terms}}, M_{DAE})$
$\hat{M}_{DAE} \Leftarrow \text{CheckLimitInfty}(\varepsilon_{\text{terms}}, \tilde{M}_{DAE})$
$M_{VL} \Leftarrow \text{CombineVariables}(\varepsilon_{\text{terms}}, \hat{M}_{DAE})$
return M_{VL}

5.1.1 Example

To illustrate the issue of singular limits, consider a simple system of two chemical reactions where chemical concentration for species 1, x_1, is converted into the concentration of species 2, x_2, at a rate k_f, and x_2 is converted back into x_1 at a rate k_r giving the equations:

$$\dot{x_1} = -k_f x_1 + k_r x_2,$$

$$\dot{x_2} = k_f x_1 - k_r x_2.$$

In this model, it is possible for the geodesic to return the limits $k_f, k_r \to \infty$. With the templates:

$$\varepsilon = \frac{1}{\infty_1} = \frac{1}{k_f},$$

$$\text{finite} = \frac{\infty_1}{\infty_2} = \frac{k_r}{k_f} = K.$$

The reparameterization for the model would be:

$$\dot{x_1} = -\frac{x_1}{\varepsilon} + \frac{Kx_2}{\varepsilon},$$
$$\dot{x_2} = \frac{x_1}{\varepsilon} - \frac{Kx_2}{\varepsilon}.$$

When the limit $\varepsilon \to 0$ is evaluated, the final model is:

$$\dot{x_1} = -\infty,$$
$$\dot{x_2} = \infty.$$

This evaluation is not successful because the new model violates two conditions: not all values are finite, and the new model contains less than $N - 1$ parameters. To evaluate this limit successfully, the variable limit evaluation algorithm as found in Algorithm 5 must be applied before evaluating the limit $\varepsilon \to 0$. The following sections explain each step of this algorithm.

5.2 Transforming an ODE to a DAE

It is possible that these singular limits occur in either an ordinary differential equations model, *ODE*, or a differential algebraic equations model, *DAE*. Combining equations with a DAE model is significantly easier. Different functions are used when solving either a DAE or an ODE model. Those functions include:

Functions Found in an ODE

- Inputs
- Observations
- Right-Hand Side Equations
- Initial Conditions

Functions Found in a DAE

- Inputs
- Observations
- Residual Equations
- Initial Conditions
- Initial Condition Derivatives

Converting from ODE to DAE is a trivial process. The right-hand side functions in the ODE are of the form:

$$\dot{x} = f(x, \theta).$$

DAE residual functions are very similar. They are of the form:

$$0 = f(x, \theta, \dot{x}).$$

The DAE's residual functions can be found by subtracting the derivative from both sides of the ODE right-hand side function. The initial condition derivatives for the DAE model are created from an ODE by substituting the variables in the right-hand side functions with its initial conditions. They follow the form:

$$\text{icd} = f(x_0, \theta).$$

After defining the residual functions and the initial condition derivatives, an ODE is successfully converted to a DAE. To proceed in evaluating the singular limit example, Eq. 5.1.1 will be converted to a DAE of the form:

$$\begin{aligned} 0 &= -\frac{x_1}{\varepsilon} + \frac{Kx_2}{\varepsilon} - \dot{x_1}, \\ 0 &= \frac{x_1}{\varepsilon} - \frac{Kx_2}{\varepsilon} - \dot{x_2}. \end{aligned}$$

5.3 Finding Epsilon

The residual equations found within the DAE only need to be manipulated if they contain ε. In this section an algorithm is defined to find the equations ε exists in. It also returns a list of the terms within that equation where ε is found. Terms are defined as parts of the equation separated by linear operators. Algorithm 9 describes the process of creating the mapping from equation index to ε terms. The mapping for Eq. 5.2 would be:

$$\left\{0 : \left[-\frac{x_1}{\varepsilon}, \frac{Kx_2}{\varepsilon}\right],\ 1 : \left[\frac{x_1}{\varepsilon}, -\frac{Kx_2}{\varepsilon}\right]\right\}. \tag{5.0}$$

Algorithm 6 Find Epsilon

Input: M_{DAE}, A model composed of differential algebraic equations.
Output: $\varepsilon_{\text{terms}}$, A mapping from Equation Index to a List of its Terms which Contain ε.

```
for Index, Equation in M_DAE do
  ε_terms ⇐ Initialize the Mapping.
  if ε in Equation then
    Append Index to ε_terms.
    for all Terms in Equation do
      if ε in Term then
        Append Term to ε_terms at key: Index.
      end if
    end for
  end if
end for
```

5.4 Variable Limit at Zero

The algorithm checks first for variables approaching their limit at zero. In this case, the variable is being divided by ε. As both of these values approach zero, they become a new finite algebraic variable. Algebraic variables do not have derivatives and can be solved for using an algebraic equation. This is the first instance of evaluating *singular limits* in this project.

Checking for these limits require the algorithm to check every instance of ε. It seems that in most cases where the variable approaches zero, all epsilon values can be found in the denominator of that variable. However, not all instances of the variable need to be found with ε. When evaluating this limit, the variable instances that have a denominator containing ε are combined into a new, finite, algebraic variable, the variables derivative is lost, and all other instances of the variable are set to zero. This variable redefinition is very similar to the redefinition found when the variable is approaching infinity.

Algorithm 7 Variable Limit to Zero

Input: M_{DAE}, A model composed of differential algebraic equations.
Input: $\varepsilon_{\text{terms}}$, A mapping from Equation Index to a List of its Terms which Contain ε.
Output: $\tilde{M}_{DAE}$

```
for Index, Term List in ε_terms do
  In Denominator ⇐ false
  for Term in Term List do
    if ε in Denominator of Term then
      In Denominator ⇐ true
    end if
  end for
  if not In Denominator then
    Divide Model Equation by ε
  end if
end for
ε̄_terms ⇐ FindEpsilon(M_DAE)
All Variables ⇐ A set of all variables in the model
for Index, Term List in ε_terms do
  for Term in Term List do
    All Variables ⇐ All Variables and Set of Variables in Term
  end for
end for
if All Variables is Left with only one variable then
  Merge that variable with ε
  Set that variable's derivative to 0
  Update other parts of the model
end if
```

5.5 Variable Limit at Infinity

The algorithm then checks if any variable is approaching infinity. In this situation, every instance of a single variable and its derivative is being multiplied by ε. However, not all instances of ε need to occur with an instance of the limit variable. Evaluating this limit is done by simply merging the limit variable and epsilon, as well as merging the limit variable's derivatives and epsilon, then evaluating the limit $\varepsilon \to 0$.

Algorithm 8 Variable Limit to Infinity

Input: M_{DAE}, A model composed of differential algebraic equations.
Input: $\varepsilon_{\text{terms}}$, A mapping from Equation Index to a List of its Terms which Contain ε.
Output: $\tilde{M}_{DAE}$

```
for Index, Term List in ε_terms do
  In Numerator ⇐ false
  for Term in Term List do
    if ε in Numerator of Term then
      In Numerator ⇐ true
    end if
  end for
  if not In Numerator then
    Multiply Model Equation by ε
  end if
end for
for Variable in All Variables do
  V_terms ⇐ FindVariable(M_DAE, Variable)
  for Index, Term List in V_terms do
    All ε ⇐ true
    for Term in Term List do
      if ε not in Term then
        All ε ⇐ false
      end if
    end for
    if All ε then
      Merge variable with ε
      Update other parts of the model
    end if
  end for
end for
```

5.5.1 Example

To illustrate this, take an approximated version of a Michaelis-Menten negative feedback loop as follows:

$$\begin{aligned} 0 &= -\frac{A}{k_1} + \dot{A} - k_2, \\ 0 &= -Bk_3 - \dot{B} + Ck_4, \\ 0 &= \frac{ACk_5 + Ak_7k_5 - BCk_6 - C\dot{C} - \dot{C}k_7}{C + k_7}. \end{aligned}$$

Evaluating the limit

$$k_2 \to \infty, k_5 \to 0$$

on the above model, with the reparameterization of

$$\varepsilon = k_5, \;\; K_{25} = k_2k_5$$

results in the model

$$\begin{aligned} 0 &= -\frac{A}{k_1} + \dot{A} - \frac{1}{\varepsilon}, \\ 0 &= -Bk_3 - \dot{B} + Ck_4, \\ 0 &= \frac{AC\varepsilon + Ak_7\varepsilon - BCk_6 - C\dot{C} - \dot{C}k_7}{C + k_7}. \end{aligned}$$

Multiplying the first equation by ε removes the potential non-finite values, giving the model:

$$\begin{aligned} 0 &= -\frac{A\varepsilon}{k_1} + \dot{A}\varepsilon - K_{25}, \\ 0 &= -Bk_3 - \dot{B} + Ck_4, \\ 0 &= \frac{AC\varepsilon + Ak_7\varepsilon - BCk_6 - C\dot{C} - \dot{C}k_7}{C + k_7}. \end{aligned}$$

Notice how every instance of the variable A, as well as its derivative $\dot{A}$ is being multiplied by ε. Also, note that ε only occurs with the variable A. Knowing this, a new variable and its derivative can be created

$$\tilde{A} = A\varepsilon,$$

$$\dot{\tilde{A}} = \dot{A}\varepsilon.$$

Substituting in this new variable and its derivative gives the final solution:

$$\begin{aligned} 0 &= -\frac{\tilde{A}}{k_1} + \dot{\tilde{A}} - K_{25}, \\ 0 &= -Bk_3 - \dot{B} + Ck_4, \\ 0 &= \frac{\tilde{A}C + \tilde{A}k_7 - BCk_6 - C\dot{C} - \dot{C}k_7}{C + k_7}. \end{aligned}$$

Note that ε is no longer contained within the model, therefore evaluating the limit is no longer needed. Also, note that the new model is a valid model containing $N - 1$ parameters.

5.6 Variable Limit Shared Between Variables

The final limit checked is similar to that of the variable approaching zero. The difference in this situation is that the limit is *shared* between several variables. The method used for identifying these limits check the terms containing ε within each equation. If any equation's ε-terms match another equation's ε-terms either exactly, or the negatives exactly, those two equations are combined to remove the like ε-terms.

Assuming that each equation contains at most one derivative, combining equations can cause multiple derivatives to occur in a single equation. Each instance of multiple derivatives in a single equation are merged into a single derivative. This merging defines a new derivative which corresponds to a new variable. That derivative that occurs most frequent corresponds to the fast variable's derivative. The fast variable's derivative is then set to zero making the fast variable algebraic.

Algorithm 9 Variable Limit Shared to Zero

Input: M_{DAE}, A model composed of differential algebraic equations.
Input: $\varepsilon_{\text{terms}}$, A mapping from Equation Index to a List of its Terms which Contain ε.
Output: $\tilde{M}_{DAE}$

```
for Index One, Term List One in ε_terms do
  for Index Two, Term List Two in ε_terms do
    if Term List One == Term List Two then
      Subtract Equation One from Equation Two
    else if Term List One == −Term List Two then
      Add Equation One to Equation Two
    end if
  end for
end for
for Equation in M_DAE do
  if Number of Derivatives in Equation > 1 then
    Merge the derivatives into a new derivative
    Apply the merging to corresponding variables
    Update the model
  end if
end for
```

Once all the like ε-terms have been combined, each equation is checked if ε is contained in any denominator.

If the algorithm finds ε in the denominator it is removed by multiplying the entire equation by ε. Finally, the limit $\varepsilon \to 0$ is realized and the singular limit has been evaluated.

5.6.1 Example

With the model defined in Eq. 5.2 and the mapping given in Eq. 5.3, it is clear that these two equations can be added together to give:

$$0 = -\frac{x_1}{\varepsilon} + \frac{Kx_2}{\varepsilon} - \dot{x_1},$$
$$0 = -\dot{x_2} - \dot{x_1}.$$

In Eq. 5.6.1, ε can be found in the denominator of the first equation. Multiplying that equation by ε gives:

$$0 = -x_1 + Kx_2 - \dot{x_1}\varepsilon,$$
$$0 = -\dot{x_2} - \dot{x_1}.$$

With ε removed from the denominator, Eq. 5.6.1 can now evaluate the limit $\varepsilon \to 0$ successfully:

$$0 = -x_1 + Kx_2,$$
$$0 = -\dot{x_2} - \dot{x_1}.$$

In this limit, parts of the variable space are becoming algebraic and is shared among variables. The objective of updating the model is to isolate this algebraic behavior. This redefinition is a bijective function, $f_{\tilde{x}}$, that transforms x to $\tilde{x}$, where at least one of the variables in $\tilde{x}$ now contains all newly found algebraic behavior in the model.

$$f_{\tilde{x}}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 + \tilde{x}_1 \end{bmatrix}$$

This new mapping results in the model:

$$0 = -\tilde{x}_1 + K(\tilde{x}_2 + \tilde{x}_1),$$
$$0 = -\dot{\tilde{x}}_2.$$

Finally, the limit has been evaluated and all the algebraic behavior has been isolated.

5.7 Algorithm Analysis

Algorithm 8, which evaluates a variable at infinity, is the most expensive of all the sub-algorithms for variable limits. Knowing that Algorithm 8 is the most expensive, the complexity

bound of the variable limits algorithm will depend on the complexity bound of solving variables approaching infinity..

For Algorithm 8, if V is the number of variables in the model, E_Q is the number of equations in the model, and E_{QT} is the average number of terms in each equation, then the run time is determined by

$$VE_QE_{QT}.$$

Therefore, the worst-case complexity bound for evaluating variable limits is $O(VE_QE_{QT})$. Unlike the reparameterization algorithm, this algorithm is unconstrained in the variables and will be scaled depending on the model used.

Chapter 6

Conclusion

The algorithms proposed in this work have been implemented and tested using different systems. The description of this implementation can be found in Appendix A. The results from this implementation are reported in Appendix C. The implementation was able to significantly reduce the amount of time required for each step of MBAM. If the templates necessary to approximate the system were found in the database, a single approximation step took less than one second for each step reported. Each automated result corresponded with previous results solved by hand. Implementing these algorithms laid the foundation for even further improvements on the usability of MBAM. These improvements includes a standardized formatting for models, automated script creation for solving the geodesic, live reporting of geodesic results, automated inferring of geodesic limits, a standardized and automatic storage of models and data, a simplified approach to manipulating models, and more. While this work was successful in automating the manifold boundary approximation method, it is only laying the foundation for further improvements and new features.

One such improvement deals with variable limits. The solution presented here only begins to implement variable limits. These limits have not been fully explored. They may benefit from a similar solution to that of the reparameterization limit templates. Yet, much is still unknown about definitively saying if a variable is approaching a limit and how that limit should be evaluated. Further automation of variable limits is outside the scope of this project. However, to aid in this exploration of further variable limits, a user interface for

MBAM was created. This user interface is used when the solution presented fails. Further reading on it can be found in Appendix A.2.

Appendices

Appendix A

Implementation

A.1 Code Structure

Several classes were built to implement these algorithms effectively. All of the classes were coded using Python 3.6. SymPy [15] was used for symbolic algebraic manipulations on the model.

The code for this implementation can be found on GitHub at `https://github.com/danebjork/AutomatedMBAM`. The documentation for this implementation can be found online as well at `https://AutomatedMBAM.readthedocs.io`. The documentation at the time of writing this paper can be found in Appendix B.

A.1.1 Models

Models are the foundation of the manifold boundary approximation method. Models consist of parameters, variables, and various types of equations. Thus, each of these is broken up into several classes as found in Fig. A.2. Both ModelParam and ModelVar deal with the instance of having a single symbol representing a value. ModelParams and ModelVars are classes that provide a simple interface for counting and manipulating these symbols. ModelEq is used for a single equation. An equation can contain two pieces: a *symbol* and an *equation*. The *symbol* is anything to the left of the equals sign, while the *equation* is anything to the right of the equals sign. If no equals sign exists, everything is considered to be a part of the *equation*. ModelEqs stores and manipulates a list of equations. ModelEqFull stores two ModelEqs, a list of equations called *substitutions*, and a list of equations called *equations*. Substitutions

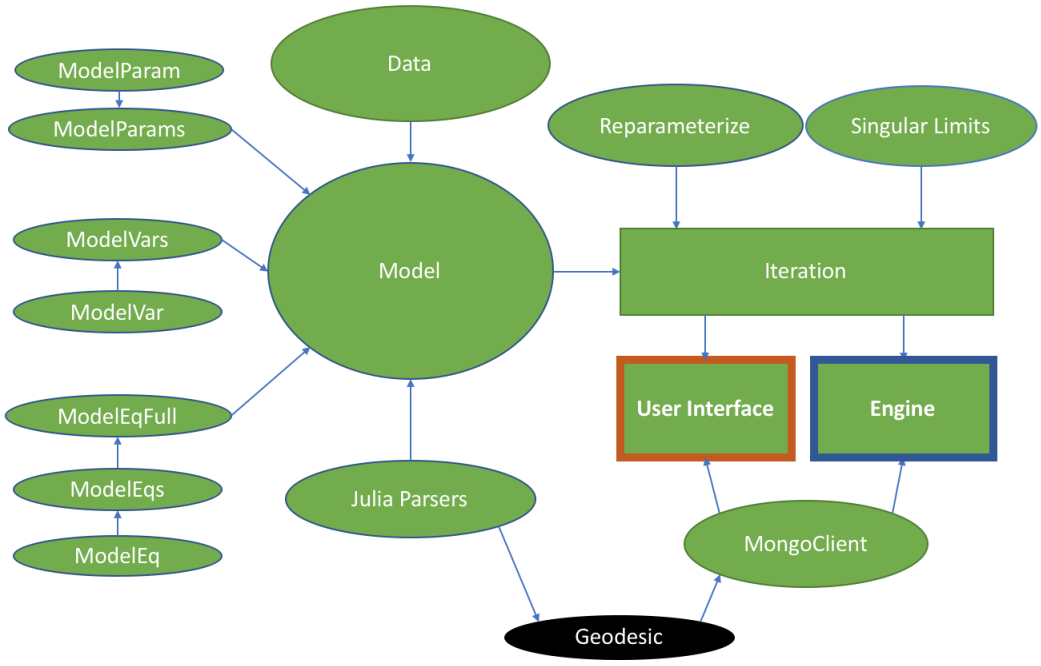


Figure A.1: The class structure for the implementation. The arrows designate dependencies within classes. The green shapes are those built in Python. The black object represents the Geodesic and is run using Julia. The outlined objects designate the final classes. The blue outlined object labeled Engine is to be used within the terminal. The red outlined object labeled User Interface is used to manage the connection for the JavaScript used in the user interface.

are those terms that appear frequently in the list of equations, but the user wishes to only evaluate those terms once. Therefore, the substitutions are evaluated prior to evaluating the equations and are substituted in as variables.

All of these objects come together to be a *model*. The model also contains data. This is the observation data used to determine model fits. Julia parser classes are then able to parse the model and its data into specific Julia scripts corresponding to the model type.

A.1.2 Julia

The packages used to calculate the geodesic are all in Julia. Julia is a recently developed programming language [3] and can be hard to integrate with other programming languages and tools. Because of this, Python generates the Julia scripts and runs them with sub-processes. These sub-processes then connect with Python through ZeroMQ, a messaging

library, to push the geodesic data to MongoDB. Python is then capable of querying that data from the database to infer geodesic limits. Fig. A.2 portrays this flow.

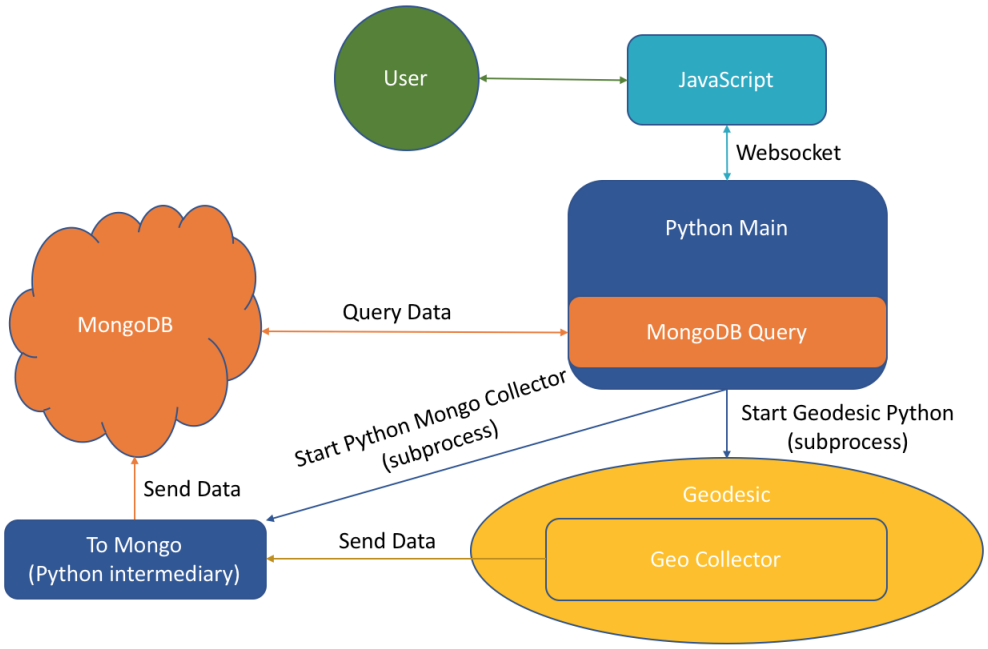


Figure A.2: The flow of the software. Each color represents a different medium for data. Orange is MongoDB, Blue is Python, Green is User Interaction, Yellow is Julia, and Teal is JavaScript.

A.1.3 MBAM

Each iteration of MBAM is contained within the *Iteration* class. This class manages everything necessary to take a model of N parameters to a model containing $N-1$ parameters. Included in this class are the two classes used to evaluate the limits. *Reparameterize* deals with taking in the limits inferred from the geodesic results and creates all possible combinations of solutions from the limit templates found in the database. Each of these is applied until successful. If failed, *Singular Limits* is then used to attempt the evaluation of any variable approaching its limit.

The *Engine* and *User Interface* classes are built on iterations of MBAM. The Engine class is to built for use in a terminal or in a Jupyter Notebook. This will attempt approximate

every parameter within the model automatically creating a new iteration for each reduction. These iterations can either attempt to calculate the limits through the geodesic or implement given limits. The User Interface class is a wrapper for the Iteration class to be used in the user interface defined in Sec. A.2.

A.1.4 Data Storage

All of the data used and created by these classes are stored in MongoDB. This database was chosen for several reasons. Those reasons are

- it is open-sourced,
- there is a large community supporting it,
- everything is converted to JSON which flows well with websockets,
- it is simple to use,
- it has a very powerful Python library, and
- MapReduce can be used for further model analysis.

The structure of the data stored follows the pattern of Hasse Diagrams. In these diagrams each model would be considered a node, and each iteration of MBAM resulting in a new model would be represented as an edge. Storing models and iterations with links to each other will allow the user to create Hasse Diagrams using MapReduce techniques inside MongoDB. Other things stored in the database include the geodesic data, the limit templates, as well as the template key.

A.2 User Interface

In this implementation the user interface was created using JavaScript, CSS and HTML. The JavaScript then connects with Python through a websocket as seen in Fig. A.2. This framework was designed to be updated and changed. By using these common languages with

a simple connection to Python, making changes will be a fairly simple process. Creating a web browser-based user interface allows for easy installation and a large set of tools specifically built for user interfaces.

A.2.1 The Model

The model is the foundation for approximation. The user interface contains inputs for everything necessary for a model to be created. The model can also be loaded from a JSON file or using an ID found in the database. Within the model section are various tabs for each different aspect of the model. Each of these aspects can be edited within their respective tabs.

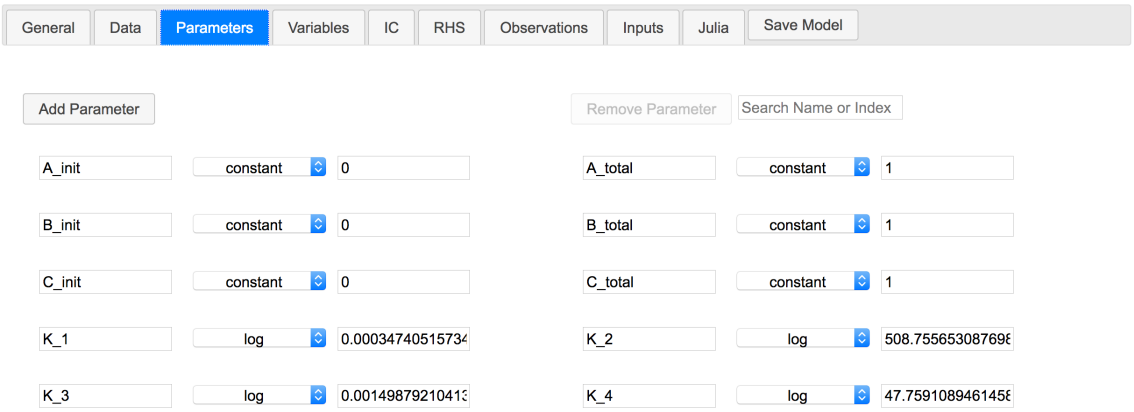


Figure A.3: On this page, parameters can be edited, added, searched for, and removed. Note the individual tabs for other parts of the model.

A.2.2 Julia Editing

The scripts used to run the geodesic are written in Julia. These scripts may need to be changed to help with some of the numerical issues when running the geodesic. A Julia IDE, ACE, was implemented with JavaScript to edit these scripts directly in the software. These scripts are generated after the model and model data have been saved in the database. After editing, they can be saved using the Update File button found above the IDE.

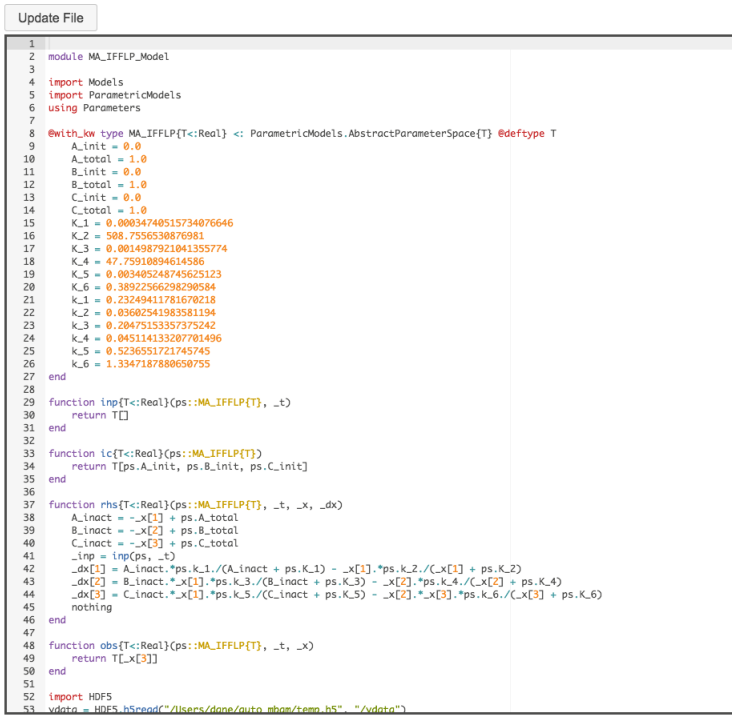


Figure A.4: A Julia integrated development environment, ACE, is used for editing any of the parsed Julia scripts. One IDE exists for editing the Julia model script, another for editing the Julia geodesic script.

A.2.3 Geodesic

The geodesic section contains three tabs: geodesic options, the Julia script, and graphs to report the geodesic results. The graphs update as the geodesic is calculated. The graphs were created using Plotly which provide a sleek and interactive look with options to save graphs as a PNG file or upload and edit in the cloud.

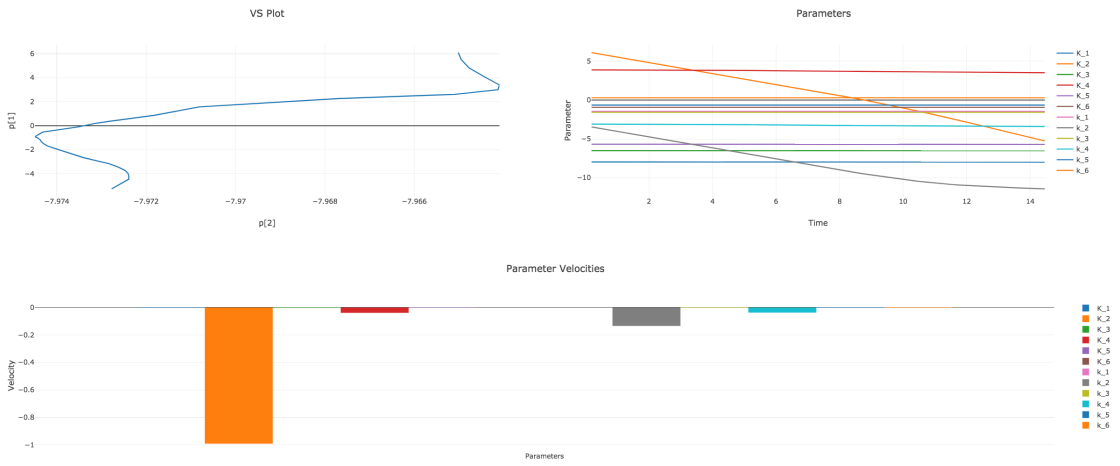


Figure A.5: A screenshot from the results of running the geodesic on a model. The top left graph reports parameter vs parameter velocities. The top right graph reports each parameter's independent velocity as the manifold is traversed. The bottom graph reports the most current velocity for each of the parameters. In this screenshot, the parameter K_2 is approaching its limit at zero.

A.2.4 Limit Evaluation

After geodesic reports the limit the Evaluation section is used to evaluate the limit. The tab Parameter Substitution inside Evaluation is automatically filled with the inferred limits of the geodesic. These results are free to edit as needed. Filling in $\tilde{\theta}$ and $f_{\tilde{\theta}}(\theta)$ for each parameter allow for θ to be substituted in the model equations with a single button.

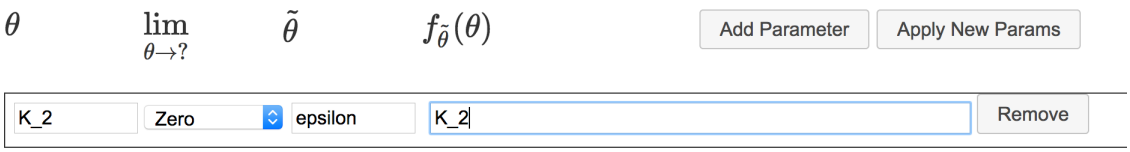


Figure A.6: This is the input to define the reparameterization for the limit. Clicking Apply New Params solves for and substitutes the new parameters into the model, and it attempts to evaluate $\varepsilon \to 0$. If the evaluation is successful, templates are created from the solution, the templates are saved to the database, the model is saved to the database, and the UI iterates to the $N-1$ model.

Once the reparameterization has been defined, the user can update variables, and update the equations in the model. A workspace was created to help with equation manipu-

lations. Equations can be imported from the model with N parameters, and exported to the model with $N-1$ parameters. The user would then import equations into the workspace, manipulate them as necessary, and export the manipulated equations into the $N-1$ model. When the user clicks on the Sub New Params button, the reparameterization defined in the previous tab will solve for and substitute the new parameters into the current workspace. Other buttons were built for convenience in viewing and manipulating these equations.

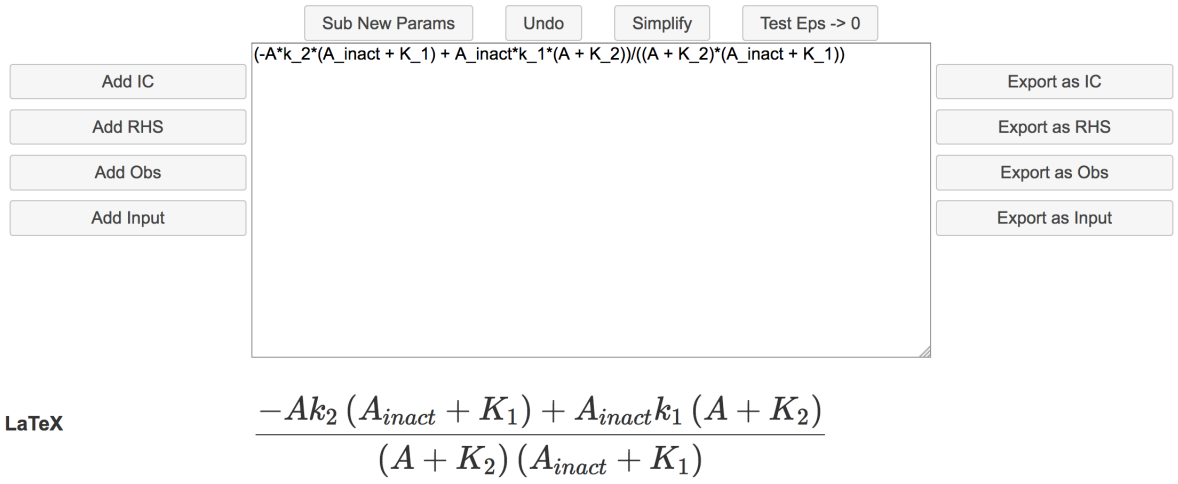


Figure A.7: The workspace is used to update the model equations. The buttons on the left side of the workspace import equations from the model into the workspace. The buttons on the right export the current workspace into the $N-1$ model. Below is a LaTeX representation of the equation to help the user more clearly see the changes to the equations.

Once all the manipulated equations have been exported into the $N-1$ equations, the user will then define a name for the $N-1$ model. Finally, this new model will be saved in the database and the entire UI will be refreshed with the $N-1$ model replacing the N model.

Appendix B

Code Documentation

This chapter outlines the documentation of the implementation at the time of writing this paper. This documentation will change as the implementation updated and improved. Please refer to the most recent version found at `https://AutomatedMBAM.readthedocs.io`.

CONTENTS:

CHAPTER

ONE

MBAM.MODELING PACKAGE

1.1 Submodules

1.2 mbam.modeling.data module

Manages the HDF5 files and reading of HDF5 byte strings from the UI. HDF5 file types are somewhat tricky to work with over websockets and cannot be read using javascript, so this module contains some workarounds for that.

class mbam.modeling.data.**MData**

Bases: object

create_zero_data(*num_vars*, *t_start*, *t_end*, *t_step*)

Creates data full of zeros with the given lengths, and saves the data to an hdf5 file.

Parameters

- **num_vars** (int) – The number of variables being observed.
- **t_start** (float) – The starting point for the time axis.
- **t_end** (float) – The final point for the time axis.
- **t_step** (float) – The step size taken for each time point.

load_bytes(*byte_str*)

Saves the string of raw bytes to a file, then reads that file using the hdf5 library.

Parameters **byte_str** (byte_str) – The hdf5 data in byte_str format.

load_from_id(*data_id*)

Loads the data from the database with the matching id. Saves it to an hdf5 file, and saves the id.

Parameters **data_id** (str) – The id of the data object to be loaded.

load_hdf5(*file_path*)

Loads an hdf5 file.

Parameters **file_path** (str) – The full path to the hdf5 file.

save_data()

Saves the hdf5 data to the database.

save_to_hdf5(*full_path='temp.h5'*)

Saves the data to an hdf5 file.

Parameters **full_path** (str) – The path to where the file will be saved. Defaults to "./temp.h5".

1.3 mbam.modeling.elements module

All pieces of a model. This contains Param, Params, Var, Vars, Eq, Eqs, and EqFull.

Any model dictionary will be parsed into these individual pieces.

class mbam.modeling.elements.**Eq**(*str_eq*, *equals=False*)
Bases: object

The Base Equation Class. May contain two parts, a symbol and an equation.

Example

a = b + c => symbol: a, equation: b + c

If the input as an '=', the left side will be the symbol, the right side will be the equation.

atoms
set – The set of all SymPy.Symbol objects in the equation.

change_to_DAE(*var*)
Subtracts the given variable from the right hand side.

Used to move derivatives to the right hand side.

Parameters **var** (SymPy.Symbol or str) – The variable to be subtracted from the right hand side, often a derivative.

dict
dict – Equation dictionary containing SymPy objects.

divide_epsilon()
Divide the right hand side by 'epsilon'.

eval_epsilon()
Take the limit as the parameter epsilon goes to zero.

latex
str – The latex string of the equation.

multiply_epsilon()
Multiply the right hand side by 'epsilon'.

str_dict
dict – a = b + c => {"sym" – "a", "eq": "b+c"}.

substitute(*substitutions*)
Substitutes a list of SymPy substitutions into the equation, including the symbol.

Parameters **substitutions** (list) – A list of tuples used to subtitute into the equation.

sympy
SymPy Symbol – The equation as a SymPy object.

class mbam.modeling.elements.**EqFull**(*eq_type*, ***kwargs*)
Bases: object

This class contains both Substitutions and Equations for a given equation types.

e.g. OBS, IC, RHS, RES, etc.

atoms
set – A set of all SymPy Symbols found in all equations.

check_subs()
Looks for the 'sym' in each substitution to be contained in the equations. If it is not contained in the equations, it is removed.

dict
dict – {"eq_type" – {"sbs": [sym_sb, ...], "eqs": [sym_eq, ...]}}

eq_symbols
list – A list of symbolic left hand side symbols.

eqs_sym_list
list – A list of symbolic eqs.

eval_epsilon()
Evaluate epsilon for all sbs and eqs.

latex
dict – {"eq_type" – {"sbs": [latex_sb, ...], "eqs": [latex_eq, ...]}}

sbs_sym_list
list – A list of symbolic sbs.

set_eq_type(*t*)
Updates the equation type.

Parameters **t** (str) – The type of equations: 'obs', 'ic', 'rhs', 'res', etc.

str_dict
dict – {"eq_type" – {"sbs": [str_sb, ...], "eqs": [str_eq, ...]}}

substitute(*substitutions*)
Substitute a list of SymPy substitutions into all sbs and eqs.

Parameters **substitutions** (list) – A list of tuples used for SymPy substitutions.

class mbam.modeling.elements.**Eqs**(*str_eq_list*, *equals=False*, *subs=False*)
Bases: object

A class containing a list of equations **or** substitutions as Eq.

atoms
set – A set of all SymPy Symbols in all equations.

dict
dict – eqs_dict = {"sbs" – [{'sym': 'a', 'eq': 'b + c'}]} as sympy equations.

divide_epsilon(*index*)
Divide the equation at *index* by 'epsilon'.

drop_equation(*symbol*)
Removes the equation with the given 'symbol'.

Example

a = b + c => [{'sym': a, 'eq': b + c}]

drop_equation(a) => []

eval_epsilon()
Evaluate the limit as epsilon goes to zero for each equation.

latex
list – A list of the equations in latex formatting.

left_hand_symbols
list – A list of equation symbols (the left hand side of '=').

list
list – A list of string representation of the equations.

multiply_epsilon(*index*)
Multiply the equation at *index* by 'epsilon'.

replace_eq(*index*, *eq*)
Replace the *eq* at the given *index*.

Parameters

- **index** (int) – The index of the equation to replace.
- **eq** (str) – The string representation of the new equation.

str_dict
dict – eqs_dict = {"sbs" – [{'sym': 'a', 'eq': 'b + c'}]} as strings.

substitute(*substitutions*)
Substitutes a list of SymPy *substitutions* into each equation.

sym_list
list – A list of dictionaries containing the SymPy symbolic equations.

class mbam.modeling.elements.**Param**(***kwargs*)
Bases: object

Base Parameter Class.

dict
dict – param = {'name' – 'p1', 'init_val', 1.0, 'transform': 'log'}

set_init_val(*init_val*)
Updates the initial value.

Parameters **init_val** (float) – The initial parameter value.

set_name(*name*)
Updates the name.

Parameters **name** (str) – The name of the parameter.

set_transform(*transform*)
Updates the transform value.

Parameters **transform** (str) – Transformation applied to the parmater. Valid transforms: 'identity','log', 'sinh', or 'constant'.

class mbam.modeling.elements.**Params**(*p_list*)
Bases: object

A class containing all model parameters as Param.

dict
dict – ps_dict = {"ps" – [{'name': 'p1', 'init_val', 1.0, 'transform': 'log'}]}

list
list – A list of parameter names.

non_constants
list – A list of *non-constant* parameter names.

remove_epsilon()
Removes any parameter named 'epsilon'.

symbols
list – A list of *non-constant* parameter SymPy symbols.

update_index_init(*index*, *init_val*)
Sets the parameter's initial value at *index* to *init_val*.

Parameters

- **index** (int) – The index of the parameter to be updated.
- **init_val** (float) – The initial value.

update_index_name(*index*, *name*)
Sets the parameter's name at *index* to *name*.

Parameters

- **index** (int) – The index of the parameter to be updated.
- **init_val** (float) – The initial value.

update_index_transform(*index*, *t*)
Sets the parameter's transformation at *index* to *t*.

Parameters

- **index** (int) – The index of the parameter to be updated.
- **t** (str) – The transformation to change the given parameter to.

update_params(*old_params*, *new_params*)
Renames each parameter found in *old_params* to the name in *new_params*.

Parameters

- **index** (int) – The index of the parameter to be updated.
- **init_val** (float) – The initial value.

class mbam.modeling.elements.**Var**(***kwargs*)
Bases: object

Base Variable Class.

dict
dict – var = {'name' – 'x1', 'type': 'dynamic'}

set_name(*name*)

Parameters **name** (str) – The name of the variable.

set_type(*t*)

Parameters **t** (str) – The variable type. Either 'algebraic' or 'dynamic'. Defaults to 'dynamic'.

class mbam.modeling.elements.**Vars**(*v_list*)
Bases: object

A class containing all model variables as Var.

dict
dict – vs_dict = {"vs" – [{'name': 'v1', 'type', 'dynamic'}]}

list
`list` – A list of variable names.

set_index_alg(*index*)
Updates the variable at the given index to 'algebraic'.

Parameters **index** (`int`) – The index of the variable to be updated.

set_index_dyn(*index*)
Updates the variable at the given index to 'dynamic'.

Parameters **index** (`int`) – The index of the variable to be updated.

set_var_alg(*var_name*)
Update the variable with *var_name* as name to 'algebraic'.

Parameters **var_name** (`str`) – The name of the variable to be updated.

update_name(*old_name*, *new_name*)
Renames a single variable from *old_name* to *new_name*.

Parameters

- **old_name** (`str`) – The name to be replaced.
- **new_name** (`str`) – The new name used in place of old_name.

var_index(*var_name*)
Return the index of the given variable name.

Parameters **var_name** (`str`) – The name of the variable to be found.

Returns **index** – The index of the variable with the given name.

Return type `int`

1.4 mbam.modeling.hdf5tojson module

HDF5 to JSON converter Author - Janu Verma j.verma5@gmail.com

Used to upload HDF5 files into python for graphing in plotly.

class `mbam.modeling.hdf5tojson.`**HDF5_JSON**(*file_name*)
Bases: `object`

Converts the contents of an HDF5 file into JSON. Also has methods to access the contents of a group directly without following the hierarchy.

Groups()
Returns all the groups in the HDF5 file. Helpful in exploring the file and getting an idea of the contents.

groupContents(*group*)
Returns the contents of a groups. You can access the contents of the group directly without following the hierarchy.

jsonOutput()
Returns a JSON document containing all the information stored in the HDF5 file. Creates a JSON file of the same name as the input HDF5 file with json extension. When decoded the file contains a nested dictionary. The primary key is the root group ''.

jsonString()
Returns a JSON document containing all the information stored in the HDF5 file. Creates a JSON file

of the same name as the input HDF5 file with json extension. When decoded the file contains a nested dictionary. The primary key is the root group ''.

subgroups(*group*)
Returns the subgroups of the group.

class mbam.modeling.hdf5tojson.**NumpyAwareJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

This class facilitates the JSON encoding of Numpy onjects. e.g. numpy arrays are not supported by the standard json encoder - dumps.

default(*obj*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

1.5 mbam.modeling.models module

A collection of model types: Base, Funciton, ODE, and DAE.

Base is the parent class for the other model types.

class mbam.modeling.models.**Base**(*model_dict*)
Bases: object

The Base Model Class.

all_params_in_eqs
bool – True if all non-constant parameters are found in the model.

atoms
set – All SymPy Symbols contained inside all of the model's equations.

check_eqs_finite
bool – True if there are no infinity or negative infinity values in the model.

check_subs()
Checks each equation type to make sure the substitutions listed are used in the equations. Deletes unused substitutions.

check_var_types()

dict
dict – The dictionary representation of the model with equations as symbolic equations.

eval_epsilon()
Evaluate the limit as 'epsilon' goes to zero for all equations in the model.

is_valid()
Checks if all parameters are in equation, and all values finite.

Returns **valid** – True if the model is a valid model.

Return type bool

latex
returns: **latex_str** – The latex string to be used for publications. :rtype: str

latex_eqs
returns: **latex_dict** – The latex version of each of the model's equations. :rtype: dict

str_dict
dict – The dictionary representation of the model with all values cast as strings.

substitute(*substitutions*)
Substitute the list of substitutions into all model equations.

Parameters **substitutions** (list) – A list of tuples for SymPy substitution.

update_name(*hasse_child_num*)
Creates a new name for a child off the base name inherited from the parent.

Parameters **hasse_child_num** (int) – The number of children the parent model has.

update_params(*old_p*, *new_p*)
Update the model's parameters from *old_p* to *new_p*.

Parameters

- **old_p** (list) – The list of old parameters.
- **new_p** (list) – The list of new parameters.

class mbam.modeling.models.**DAE**(*model_dict*)
Bases: *mbam.modeling.models.Base*

Differential Algebraic Equations Model Class.

icd_algebraic()

class mbam.modeling.models.**Function**(*model_dict*)
Bases: *mbam.modeling.models.Base*

A Class for models containing only algebraic functions.

Not really used...

class mbam.modeling.models.**ODE**(*model_dict*)
Bases: *mbam.modeling.models.Base*

Ordinary Differential Equations Model Class

dae_var_to_ic()
Create a list of substitutions for converting from ODE RHS functions to ICD's. This will be used to substitute each variable for its initial cond.

Example

subs = [(var[0], ic[0]), . . . , (var[n], ic[n])]

deriv_name(*var_name*)
Inserts 'dot' between characters and the first `int` found in the name.

Parameters **var_name** (`str`) – A variable name to altered to a derivative name.

Returns **new_var** – A derivative name with dot inserted.

Return type `str`

to_dae()
Converts the current ODE into a DAE.

Subtracts the derivatives from the RHS of the ODE to create RES. Creates ICD's based off the RHS equations. Then substitutes the variables in the ICD with its corresponding initial condition.

Returns **dae** – The equivalent DAE representation of the current ODE.

Return type `DAE`

1.6 Module contents

Everything needed to construct models for automatic approximation.

CHAPTER

TWO

MBAM.PARSING PACKAGE

2.1 Submodules

2.2 mbam.parsing.base module

class mbam.parsing.base.**BaseParser**(*mbam_model*, *data_path*)

Bases: object

Parent class for all model parsers.

create_default_options()

Used until options are updated with update_options

create_julia_swap()

Generates a list of sympy substitution tuples to sub out sympy vars with julia formating.

Example

Time update: t => _t.

Vars update: x1 => _x[1].

Params update: p1 => ps.p1.

Inputs update: u1 => _inp[1].

init_models_dir()

Generates a directory: julia_scripts/models/ to save the model file within.

list_out_param_type(*p_type*)

save_to_file(*script*)

Overwrites the current script with the given script string.

Parameters **script** (str) – A string representation of a full julia model script.

update_options(*options*)

Creates and saves a new script generated with the given options.

Parameters **options** (dict) – Must follow format: {"bare": bool, "weights": bool}

write_bare_model()

write_constants()

write_data()

write_end()

write_equation_list(*eq_list*)
Given a list of equations, write them out as a julia array.

Parameters **eq_list** (list) – A list containing equation dictionaries. Where each equation is only on the right-hand side. There should be no symbol for this function. The non-julia values will be subbed with julia values.

Example

x + b => {"sym": "", "eq": "a + b"}

write_equation_return(*eq_list*)
Given a list of equations, write them out as a julia array following the return statement.

Parameters **eq_list** (list) – A list containing equation dictionaries. Where each equation is only on the right-hand side. There should be no symbol for this function.

Example

a + b => {"sym": "", "eq": "a + b"}

write_imports()

write_inputs()

write_param_transforms(*bare=False*)

write_params()

write_substitutions(*sub_list*)
Given a list of substitutions, write them out prior to the return statement or the main equations.

Parameters **sub_list** (list) – A list containing equation dictionaries.

Example

c = a + b => {"sym": "c", "eq": "a + b"}

write_xi()

2.3 mbam.parsing.basediff module

class mbam.parsing.basediff.**BaseDiffParser**(*mbam_model*, *data_path*)
Bases: *mbam.parsing.base.BaseParser*

Parent class for all **differential** model parsers.

Adds observation parsing to the BaseParser class.

write_obs()

2.4 mbam.parsing.dae module

class mbam.parsing.dae.**DAEParser**(*mbam_model*, *data_path*)
Bases: *mbam.parsing.basediff.BaseDiffParser*

Used for parsing ODE models.

add_derivs_to_julia_swap()
Adding derivatives to the substitution list of julia vars for every variable in the model.

Example

x_1 => x_1dot => _dx[1]

create_icd_swap()
Substitute the variables in the ICD's with their corresponding initial condition.

Example

if IC: [x_1_init + p1], then ICD[x_1] => ICD[ps.x_1_init + ps.p1]

write_bare_model()

write_dvars()
Returnes a list of true/false for each variable, where true = dynamic false = algebraic.

write_ic()
Specialty function for writing the IC function.

write_ic_subs()

write_icd_list(*eq_list*)
Specialty funciton for writing ICD's in a list format.

write_model()

write_res()
Specialty funciton for writing RES function.

write_res_equations()
Specialty funciton for writing RES equations.

write_script()

2.5 mbam.parsing.function module

class mbam.parsing.function.**FunctionParser**(*mbam_model*, *data_path*)
Bases: *mbam.parsing.basediff.BaseDiffParser*

Used for parsing models composed of simple algebraic functions.

write_bare_model()

write_f()

write_model()

write_script()

2.6 mbam.parsing.geo module

class mbam.parsing.geo.**GeodesicParser**(*mbam_model*, *model_path*, *collector_path*)

Bases: object

default_options()

Used until options are updated with update_options

end_script()

geo_loop()

init_geos_dir()

init_script()

load_options()

save_to_file(*script*)

update_options(*options*)

Creates and saves a new script generated with the given options.

Parameters **options** (dict) – Must follow format:

{

"tmax": float,

"lambda": float,

"abstol": float,

"reltol": float,

"use_svd": bool,

"use_pinv": bool,

}

write_geo_script()

2.7 mbam.parsing.ode module

class mbam.parsing.ode.**ODEParser**(*mbam_model*, *data_path*)

Bases: *mbam.parsing.basediff.BaseDiffParser*

Used for parsing ODE models.

write_bare_model()

write_ic()

write_model()

write_rhs()

write_rhs_equations()

RHS function follow a different formatting for parsing equations. This is their specialized parsing function.

write_script()

2.8 Module contents

All the parsers necessary to convert an *mbammodel* and path to an hdf5 data file into the Julia model file and its corresponding Geodesic file.

CHAPTER

THREE

MBAM.LIMITS PACKAGE

3.1 Submodules

3.2 mbam.limits.reparameterize module

Given a set of limits, return all currently templated possibilities for a reparameterization. Reparameterization is the first tool to use when attempting to evaluate a model at a limit and remove a single parameter.

This module uses combinatorics to create possible reparameterizations using the current limit, and templates pulled from the database. While there may be lots of possibilities in the reparameterizations, only one valid model needs to be returned.

class mbam.limits.reparameterize.**Reparam**(*limits*, *key_legend*, *model_class=None*)

Bases: object

Creates all possible fthetas given the limit, the legend for the limit template key, and filtering with the model class.

add_to_ftheta(*subbed*, *epsilon*)

Once the templates are filled, fthetas need to be created.

Parameters

- **subbed** (SymPy symbolic equation) – The new parameter combination as an equation.
- **epsilon** (bool) – True if the parameter combination evaluates to zero.

count_limits()

Creates a mapping to see how many of each limit is being reached by the parameters. This is used as the *limit_key* and will filter templates.

fill_template(*templates*, *epsilon=False*)

Iterates through templates, and creates any possible manipulation of each template with the current limits.

Parameters

- **templates** (list) – The templates to be manipulated.
- **epsilon** (bool) – True if the templates evalaute to zero.

Returns **template_subs** – A list of templates filled by parameters approaching their limits.

Return type list

Example

limits: {"p1": "inf", "p2": "inf"},

template = "inf_1/inf_2",

returns ["p1/p2", "p2/p1"]

filter_fthetas()
Creates all possible ftheta combinations for the given limit, then filters out he invalid fthetas.

See in-line comments for details.

find_limit_subs()
Creates possible substitutions (template_name, param_name).

If two parameters are approaching infinity then the potential names for those parameters could be either 'inf_1' or 'inf_2'. This function finds a list of all possible parameter limit namings.

Returns **template_fill_list** – A list that contains all possible subs for the parameters to fill the limits.

Return type list

Example

{"a": "inf"} => the template needs one infinity:

returns [[('a', 'inf_1')], [(('b', 'inf_1)]]

{"a": "inf", "b": "inf"}=> the template needs two infinities:

returns [[('a', 'inf_1'), ('b', 'inf_2)], [('a', 'inf_2'), ('b', 'inf_1)]]

get_fthetas(*e_temp*, *f_temp*)
Finds all the potential ftheta combinations.

Parameters

- **e_temp** (list) – List of templates that evaluate to zero.
- **f_temp** (list) – List of templates that evaluate to a non-zero finite value.

Returns **valid_fthetas** – A list of ftheta dictionaries.

Return type list

Example

ftheta = {"theta": "p1", "limit": "inf", "tilde": "epsilon", "f": "1/p1"}

sub_label_name(*subbed_eq*, *epsilon*)
A simiple algorithm to rename new parameters.

Parameters

- **subbed_eq** (str) – The parameter equation to be renamed
- **epsilon** (bool) – Automatically renames parameter to 'epsilon' if True.

Returns **label** – The new label for the parameter combination.

Return type str

Example

a/b -> a_over_b

sub_params_in_template(*sub_list*, *template*, *epsilon*)
Replaces limit types in templates with actual parameters.

Parameters

- **sub_list** (list) – List of SymPy substitutions of the form (limit_type, parameter).
- **template** (dict) – Template dictionary to be substituted with parameters.
- **epsilon** (bool) – True if the template evaluates to zero.

3.3 mbam.limits.singular_limit module

Singular Limits are often found when using the Manifold Boundary Approximation Method. This module is built to evaluate these limits automatically. It works for specific cases e.g. var*epsilon, var/epsilon, and combining equations.

Further work can be done to extend this class to work on a more broad range of variable/singular limits.

class mbam.limits.singular_limit.**SingularLimit**(*dae_model*)
Bases: object

Parameters **dae_model** (DAEModel) – A differential algebraic equation model.

check_var_and_epsilon()
Checks every variable to see if one is found with every instance of epsilon. *May also need to add check for the derivative.*

Returns True if one variable is found with every instance of epsilon.

Return type bool

combine_eqs(*add_dict*)
Combines the given equations within the model.

Parameters **add_dict** (dict) – A dictionary mapping from operator to list of tuples each containg two values – the equations to be operated on.

Example

{"+": [(0, 1)]}: Add equation 0 to equation 1.

create_new_var_subs(*new_vars*, *new_var_eqs*)
Creates all substitutions necessary to update different functions as well as for use in the algorithm.

create_old_to_new_vars()
Creates the substitution from the old variables to those newly defined with the singular limit.

divide_epsilon()
Checks if the majority of any equation in the model is combosed of the term being multiplied by epsilon. If it is the majority and there are no epsilon terms found in the denominator, the equation will be divided by epsilon.

eval_limit()
Evaluates the potential singular limit.

extract_epsilon_terms(*eq*)
Extracts all epsilon terms from a given equation.

Parameters **eq** (Sympy Equation) – A symbolic form of an equation, potentially containin epsilon.

Returns

- **terms** (list) – The list of terms that contain epsilon found in the equation.
- **negative_terms** (list) – The list of terms that contain epsilon found in the equation, where each term has been multiplied by -1.

find_derivatives()
Searches the residuals for derivatives. Used to merge those derivatives where more than one derivative occurs in a given equation.

Returns A dictionary containing the equation index as the key, and a list of the derivatives contained in that equation as the value.

Return type dict

find_epsilon()
Searches the residuals function for all terms that epsilon is found in. Creates a dictionary mapping from equation index to list of terms containing epsilon found in that equation.

find_fast_var(*deriv_counts*)
Uses the counts of derivatives to assign the variable with the most frequent derivative as the fast variable.

Parameters **deriv_counts** (dict) – A dictionary mapping from derivative value to count.

find_like_terms()
Checks each list of terms to find equations that have exact (positive or negative) lists of terms.

Returns **to_combine** – A dictionary mapping from operator to a list of tuples directing which equations should be combined together.

Return type dict

find_new_var_names(*num_vars*)
Creates new variable names that don't already occur within the model.

Parameters **num_vars** (int) – The number of variable names to be returned.

Returns A list of new parameter names.

Return type list

merge_operator(*var*)
Finds the operator used between the variable and epsilon.

Parameters **var** (str) – The variable that occurs with every instance of epsilon.

Returns A string representation of the operator used between the variable and epsilon. Empty if the operators vary.

Return type str

merge_subs(*sym_vs*, *str_operator*)
Combines the variables in the list with epsilon if being operated on by the given operator.

Parameters

- **sym_vs** (Sympy.Symbol) – A list of symbolic variables (often just one variable).
- **str_operator** (str) – The operator acting on the variable and epsilon.

Returns **subs** – A list of tuples for substituting var(operator)epsilon to vartilde.

Return type `list`

multiply_epsilon()
Looks for epsilon in the denominator of the equation. If found, multiplies that equation by epsilon to remove instabilities.

new_var_equals(*new_v_dict*)
Counts the derivatives in each equation

Parameters **new_v_dict** (`dict`) – A dictionary containing the equation index as the key, and a list of the derivatives contained in that equation as the value.

Returns A list of combined derivatives. These combinations will be represented as a single variable after the limit is evaluated.

Return type `list`

set_fast_algebraic()
Sets the fast variable as algebraic. * Most likely not needed anymore.

sub_vars_in_ic()
Substitutes the new variables into the initial conditions.

sub_vars_in_icd()
Substitute the new variables into the initial condition derivatives function.

sub_vars_in_obs()
Substitute the new variables into the observation function.

sub_vars_in_res()
Substitutes the new variables into the residuals function.

transform_vars()
Creates all substitutions necessary for the model to be valid following the evaluation of a singular limit.

update_vars()
Updates the variables in the model with new names.

3.4 Module contents

The automatic limit evaluator module. Reparameterizes the model and attempts to evaluate any singular limits.

CHAPTER

FOUR

MBAM PACKAGE

4.1 Subpackages

4.2 Submodules

4.3 mbam.engine module

Manages many iteration of MBAM.

class mbam.engine.**Engine**(*model_dict*, *data_path*)

Bases: object

Parameters

- **model_dict** (dict) – A dictionary containing all the model information.
- **data_path** (str) – The full path to the hdf5 data file for the model.

apply_limits(*limit_list*, *printing=False*)

Applies a list of given limits. No geodesic will be run.

Parameters

- **limit_list** (list) – A list of dictionaries representing limits. Each dictionary maps from a parameter to a limit.
- **printing** (bool) – Prints out each new model in latex formating if True.

print_latex(*limit*)

Prints out a latex version of the current model and the limit that was evaluated.

Parameters **limit** (dict) – The limit dictionary that was just evaluated.

print_limit(*limit*)

Parses the given limit into a latex equation.

Parameters **limit** (dict) – A dictionary containing limits – a mapping from parameters to limits.

run(*printing=False*)

Runs the geodesic, infers the limit, attempts to evaluate the limit. Continues to repeat the process until a failure.

Parameters **printing** (bool) – Prints out each new model in latex formating if True.

4.4 mbam.geodesic module

A manager module for running and reporting on the geodesic.

class mbam.geodesic.**Geodesic**(*geo_parser*, *sender_file_path*)
Bases: object

check_engine_geo(*geo_data*)
Checks the current data for any potential limits used for engine.

check_limits(*recent_vs*)
Checks parameter velocities for any potential limits.

Parameters **recent_vs** (list) – A list of the most recent parameter velocities.

curr_data()
Queries the geodesic from the database and checks potential limits

Returns Contains geodesic data and any inferred limits.

Return type dict

find_threshold(*n*)
The threshold set to determine what velocities are defined as approaching a limit.

Parameters **n** (int) – The nth parameter being checked.

kill()
Kills the subprocesses used to run the geodesic and the data collector.

run_geo_auto()
Runs the geodesic until manually killed, or until the limits are found.

Returns **limits** – A dictionary of limits. e.g. {"p1": "inf", "p2", "zero"}

Return type dict

start()
Starts both subprocesses: the Julia geodesic and data collector.

update_limits(*geo_data*)
Checks the current data for any potential limits used for UI.

Parameters **geo_data** (dict) – The dictionary of geodesic data.

Returns The geodesic data and any limits inferred.

Return type dict

4.5 mbam.iteration module

This class manages everything necessary to go from a model with N parameter to a model with N-1 parameters. This includes running the geodesic, inferring the limits, reparameterizing the model, as well as evaluating singular perturbations.

class mbam.iteration.**Iteration**(*model*, *model_id*, *data_path*)
Bases: object

add_geo_options(*options*)
Update the options for the Julia geodesic parser.

Parameters **options** (dict) – options = {

"tmax": `float`,

"lambda": `float`,

"abstol": `float`,

"reltol": `float`,

"use_svd": `bool`,

"use_pinv": `bool`,

}

add_julia_options(*options*)
Update the options for the Julia model parser.

Parameters **options** (`dict`) – options = {"bare": `bool`, "weights" `bool`}

apply_ftilde(*exception=False*)
Attempts to evaluate epsilon approaching zero. If the model is valid, it returns True. If it is not valid, the function is called again and attempts singular perturbation before evaluating the limit.

Parameters **exception** (`bool`) – True if reparameterization has failed without an exception applied.

Returns True if the evaluation was successful.

Return type `bool`

apply_limits(*limits*)
Attempts to reparameterize the model. If not successful, exceptions are attempted with Singular Perturbations.

Parameters **limits** (`dict`) – Dictionary of limits to be applied to the model.

Returns True if the limits were applied successfully.

Return type `bool`

auto_run()
Finds the limits, applies the limits, and saves the new model if successful.

Returns True if the iteration was successful and saved.

Return type `bool`

check_ftildes(*solved*)
Make sure that f_inv functions in ftilde don't contain any of the parameters to be removed from the old model.

Parameters **solved** (`list`) – A list of ftildes.

Returns True if the f_inv doesn't contain any parameters to be removed.

Return type `bool`

create_tilde_subs(*ftildes*)
Using ftildes, create a list of tuples for SymPy substitutions.

Parameters **ftildes** (`list`) – A list of ftildes ready to be used for substitutions.

Returns A list of SymPy substitutions.

Return type `list`

dict
`dict` – Turn the iteration into a dictionary for saving.

find_limits()
Starts the geodesic and runs until a limit is found, or until the geodesic crashes.

init_geodesic()
Creates the Geodesic object, and retrieves its id.

init_parsers()
Assigns the parser for the model based off the model type, then it creates the geodesic parser.

kill_geodesic()
Kills the geodesic subprocess.

limit_keys_to_ps(*limits*)
Converts what the geodesic returns to a more useable format.

Parameters **limits** (dict) – A dictionary of limits returned by the geodesic.

Returns **limits** – A dictionary of limits converted to new format.

Return type dict

Example

Converts (p_index: limit) to (p_name: limit).

parse_templates()
If the iteration is successsful, the ftildes are parsed into templates to be saved in the database.

Example

ftilde = {"theta": "p1", "limit": "inf", "tilde": "epsilon", "f": "1/p1", "f_inv": "1/epsilon"}

realize_limit()
Checks if the model has been successfully applied.

Returns True if the new N-1 model is valid.

Return type bool

save_iteration()
Adds the iteration to the database, updates the parent and child models with the successful iteration id.

solve_ftildes(*ftheta*)
Using ftheta, solve for f_inv to substitute in place of the old parameters.

Parameters **ftheta** (list) – A list of fthetas to be solved.

Returns A list of solved fthetas, now known as ftildes.

Return type list

Example

ftheta = {"theta": "p1", "limit": "inf", "tilde": "epsilon", "f": "1/p1"}

ftilde = {"theta": "p1", "limit": "inf", "tilde": "epsilon", "f": "1/p1", "f_inv": "1/epsilon"}

try_singular_limit()
Converts an ODE to a DAE, then attempts combinging equations to solve the singular perturbation error.

unresolved_thetas(*ftildes*)
Substitutes out the old parameters in ftilde until no old paramters remain.

Parameters **ftildes** (list) – A list of ftildes to be solved.

Returns A list of solved ftildes, containing no old parameters

Return type list

Example

ftilde1 = {"theta": "p1", "limit": "inf", "tilde": "epsilon", "f": "1/p1", "f_inv": "1/epsilon"}

old_ftilde2 = {"theta": "p2", "limit": "inf", "tilde": "p1_over_p2", "f": "p1/p2", "f_inv": "p1/p1_over_p2"}

old_ftilde2 still contains an old paramter: p1.

new_ftilde2 = {"theta": "p2", "limit": "inf", "tilde": "p1_over_p2", "f": "p1/p2", "f_inv": "1/(epsilon*p1_over_p2"}

update_geo_script(*script*)
Overwrite the current julia geodesic script.

Parameters **script** (str) – The Julia geodesic file as a string.

update_julia_script(*script*)
Overwrite the current julia model script.

Parameters **script** (str) – The Julia Model file as a string.

update_params(*ftildes*)
Uses ftilde to change the parameter names from those currently in the model to those found in the ftildes.

4.6 mbam.mongo module

A custom client for connecting to MongoDB and managing the data flow for MBAM. Six collections are used in the database 'mbam':

geos: geodesic storage. models: model storage. model_data: the data read in the hdf5 files. Used in models. iters: successful MBAM iteration storage. temp_key: the key for navigating limit templates. temps: limit template storage.

class mbam.mongo.**MMongo**
Bases: object

finish_geodesic(*geo_id*, *exception=False*)
Marks the geodesic as 'done' inside MongoDB.

This allows the object waiting for the geodesic to finish to know it has completed.

get_hasse_children(*model_id*)
Returns the length of a model's successful iterations.

Is used to create a unique name for several models with the same parent model.

Parameters **model_id** (str) – The id for the parent model.

Returns **len_to_iter** – The total number of children the parent model already has.

Return type int

get_temp_key()

Returns **key** – A dictionary of the format: {"limit_type": "index in key"}

Return type dict

Example

key = {"zero": 0, "inf": 1}

init_geodesic()
Creates a new geodesic object inside MongoDB.

Returns **geo_id** – The string representation of the new geodesic.

Return type str

load_data_id(*data_id_str*)

Parameters **data_id_str** (str) – The id for the data object to find.

Returns **data** – The dictionary containing all the data found in the database.

Return type dict

load_model_id(*model_id_str*)
Retrieves a model from the database with the given id.

Parameters **model_id_str** (str) – The model id to query.

Returns **model_dict** – The model and data (if exists) connected to the given id.

Return type dict

load_templates(*key=None*, *clss=None*)
Loads all limit templates and filters them by *key* and *clss*.

Parameters

- **key** (list or tuple of int) – A list or tuple of ints defining how many of each limit type occurs within a template.
- **clss** (str) – A string used to define the model class that is going to use the templates.

Returns **list** – A list of dictionary templates after filtering by key and class.

Return type list

Example

Model classes are user defined, just be consistent.

push_geodesic(*geo_id*, *data*)
Append new geodesic data to the end of the current geodesic data.

Parameters

- **geo_id** (str) – The ID for the geodesic to add the data to.
- **data** (dict) – The dictionary of new data to be appended to the current geodesic.

push_model_from_iter(*model_id*, *iter_id*)
Add the successful iteration id to the child model.

Parameters

- **model_id** (str) – The id for the child model.
- **iter_id** (str) – The id for the successful iteration.

push_model_to_iter(*model_id*, *iter_id*)
Add the successful iteration id to the parent model.

Parameters

- **model_id** (str) – The id for the parent model.
- **iter_id** (str) – The id for the successful iteration.

query_geodesic(*geo_id*)

Parameters **geo_id** (str) – The ID for the geodesic to query.

Returns **geo_data** – The data queried from the current geodesic, excluding its ID.

Return type dict

save_iteration(*iteration*)
Add the successful iteration to the database.

Parameters **iteration** (dict) – All data corresponding to the iteration.

Returns **iter_id** – The id generated by MongoDB. Used to udpate parent/child models.

Return type str

save_model(*mbam_model*, *data_id=None*)

Parameters

- **mbam_model** (mbammodel) – The model object to be saved.
- **data_id** (str) – The id of the saved data in the database. Used to connect the model to its data.

save_model_data(*data_json*)

Parameters **data_json** (dict) – The HDF5 data to be saved inside the database.

Returns **data_id** – The id generated by MongoDB while saving the data.

Return type str

save_temp(*temp*)
Saves a single template in the database.

Parameters **temp** (dict) – A template dictionary.

Example

template = {'key': [0, 1], 'template': '1/inf_1', 'label': 'epsilon', 'class': 'michaelis-menten' }

save_temps(*temp_list*)
Saves a list of templates in the database.

Parameters **temp** (list) – A list of template dictionaries.

Example

temp_list = [{'key': [0, 1], 'template': '1/inf_1', 'label': 'epsilon', 'class': 'michaelis-menten' }]

temp_key_filter(*curr_key*, *temp_key*)
Returns true if *curr_key[i]* < *temp_key[i]* for all i.

Parameters

- **curr_key** (list or tuple of int) – A list or tuple of ints defining how many of each limit type occurs within the current limit.
- **temp_key** (list or tuple of int) – A list or tuple of ints defining how many of each limit type occurs within a queried template.

Returns **valid** – True if the queried template is functional with the current limit.

Return type bool

update_temp_key(*key*)
Overwrites the current template key with the given *key*.

Parameters **key** (dict) – A dictionary of the format: {"limit_type": "index in key"}

Example

key = {"zero": 0, "inf": 1}

4.7 mbam.ui module

A module for communication between the user interface websocket handler and MBAM functionality. This class takes the place of MBAM Engine when using the UI. It is not made to be used outside of use with the UI.

class mbam.ui.**MbamUI**
Bases: object

begin_iteration()
Checks if a model and its data have been saved, if so, an mbam iteration will be created.

eval_epsilon(*eq*)
Evaluates the limit as epsilon goes to zero in the given equation.

Parameters **eq** (str) – The equation to be evaluated.

Returns The dictionary containing the equation evaluated as epsilon goes to zero.

Return type dict

iterate()
Saves and updates the N-1 model to the N model.

Returns The dictionary containing the N-1 model to be udpated as the N model in the UI.

Return type dict

iterate_model()
Updates the model and id to the N-1 model and N-1 model id, then creates a new mbam iteration object.

kill_geodesic()
Kill the subprocesses used to run the geodesic.

load_ftheta(*fthetas*)
Takes the fthetas from the UI, transforms them to ftildes, creates the substitutes for those ftildes, and applies those substitutions to the model.

Parameters **fthetas** (list) – A list of ftheta dictionaries.

Returns True if the reparameterization with ftilde is successful.

Return type bool

load_model_id(*id_str*)
Use a model id to load the model and its data. Send necessary information forward to the user interface.

Parameters **id_str** (str) – A string representing a model id.

Returns A dictionary containing a model and its data, if data is associated with the given model.

Return type dict

manual_iterate(*model_dict*)
Used when the user updates the model manually.

Recieves a new model, checks if the model is valid, then completes the iteration from N to N-1.

Parameters **model_dict** (dict) – The dictionary representing the N-1 model.

Returns The new model dictionary to have the UI update with it as the N model.

Return type dict

query_geo()
Get the geodesic data and get it ready to be sent to the UI.

Returns The geodesic data to be sent to the UI.

Return type dict

read_model_data(*data_str*)
Takes in hdf5 byte string from the websocket, saves it to a temp.h5 file to be loaded in the Julia scripts.

Parameters **data_str** (byte_str) – A raw string of bytes representing all hdf5 data contained in a file.

save_model(*model_dict*)
Saves a model dictionary to the database and creates a model object.

Parameters **model_dict** (dict) – A dictionary containing all necessary information for a model.

Returns A dictionary containing model info to be sent to the UI.

Return type dict

save_model_dict(*model_dict*)
Turns a model dictionary into an MbamModel object.

Parameters **model_dict** (dict) – A dictionary containing all necessary information for a model.

scripts_ready()
Checks if an iteration has been made. If it has been made, then the julia scripts will be ready to be sent.

Returns True if the scripts are ready to be sent.

Return type bool

send_data_done()

Returns Response to be send to the UI containing the data read in the hdf5 file.

Return type dict

send_model_load_done()

Returns Response to be send to the UI containing a model dict.

Return type dict

send_save_done()

Returns Response to be send to the UI containing a model dict and a model in latex formatting.

Return type dict

send_scripts()

Returns Response to be send to the UI containing the julia model and geodesic scripts.

Return type dict

simplify_eq(*eq*)
Takes an equation from the UI, simiplifies it, and sends the simplified version back to the UI.

Parameters **eq** (str) – The equation to be simplified.

Returns A dictionary containing the equation as both a string and latex string.

Return type dict

start_geodesic()
Start the subprocesses necessary for the geodesic to run.

sub_ftildes(*eq*)
Substitutes f_inv in for the old parameters in the given equation.

Parameters **eq** (str) – The equation with parameters to be substituted.

Returns Contains the newly substituted equation as a string and latex string.

Return type dict

update_geo_options(*options*)
Update the julia geodesic options and send the new generated scripts to the UI.

Parameters **options** (dict) – Julia geodesic options.

Returns A dictionary containing the new script to be sent to the UI.

Return type dict

update_geo_script(*script*)
Update the julia geodesic script sent from the UI.

Parameters **options** (str) – Julia geodesic script.

update_julia_options(*options*)
Update the julia model options and send the new generated scripts to the UI.

Parameters **options** (dict) – Julia model options.

Returns A dictionary containing the new script to be sent to the UI.

Return type dict

update_julia_script(*script*)
Update the julia model script sent from the UI.

Parameters `options` (`str`) – Julia model script.

4.8 Module contents

Loads in all the module necessary to run mbam automatically.

CHAPTER

FIVE

MAIN MODULE

The websocket handler for connecting with the User Interface. Runs until killed or crashes.

main.**time**(*websocket*, *path*)

CHAPTER

SIX

JULIATOMONGO MODULE

This module initializes a *Collector* object. This object then runs until it is manually killed, or until the geodesic that it is connected to completes or crashes. It forwards all the data from the geodesic on to MongoDB.

This module is started at the same the Julia Geodesic script is started. Once a stable package for Julia is released connecting to MongoDB, this class should be removed and the geodesic should report directly to MongoDB. However, it seems that is not currently an option.

class juliatomongo.**Collector**(*geo_id*)

Bases: object

Uses ZMQ sockets to connect to the Julia Geodesic currently running, forwards the data collected to MongoDB.

collect()

Collects the data from the Julia Geodesic running.

Will run until the process is killed manually, or until the Geodesic reports that the script has either completed or crashed.

start_sockets()

Intializes the ZMQ socket on port 5556.

Appendix C

Algorithm Results

This chapter contains four models approximated using the algorithms designed in this project. Two of the models were reduced by running the geodesic with respect to the output of the model all being zero. The other two models have been approximated in previous work. These prior approximations guided the algorithm through the use of the recorded limits.

C.1 Simple Models with Zero Data

The following are two simple models which geodesics were calculated fitting the model to all zeros. Fitting the data to zeros allows for the engine to run the geodesic with minimal trouble and is possible to result in a model with zero parameters.

C.1.1 Michaelis-Menten

This model deals with four variables and represents four chemicals interacting with one another as seen in Fig. C.1. The model was derived using Michaelis-Menten kinetics.

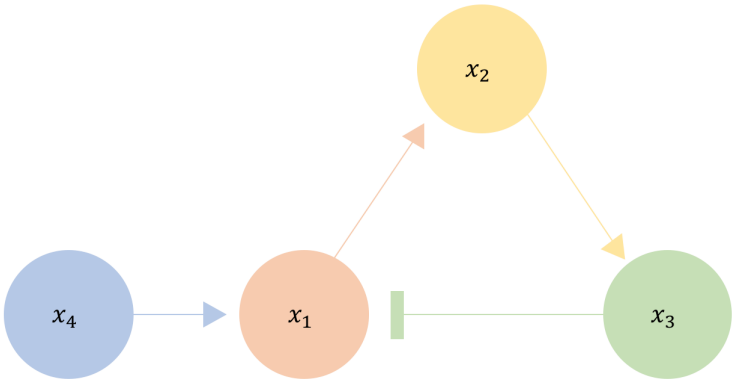


Figure C.1: The behavior modeled by Eq. C.1.1 using Michaelis-Menten kinetics.

$$N = 8$$

$$\begin{aligned}
0 =& \frac{k_3 x_{1in} x_3}{K_3 + x_{1in}} + \frac{k_4 x_{1in} x_4}{K_4 + x_{1in}} - \dot{x}_1 \\
0 =& \frac{k_1 x_1 x_{2in}}{K_1 + x_{2in}} - \dot{x}_2 \\
0 =& - \frac{k_2 x_2 x_3}{K_2 + x_3} - \dot{x}_3 \\
0 =& - \dot{x}_4
\end{aligned}$$

$N = 7$, Limit Evaluated: $K_4 \to \infty, \quad k_4 \to \infty.$

New Parameter: K_{44}.

$$\begin{aligned}
0 =& \frac{1}{K_3 K_{44} + K_{44} x_{1in}} (-K_3 K_{44} \dot{x}_1 + K_3 x_{1in} x_4 \\
& + K_{44} k_3 x_{1in} x_3 - K_{44} x_{1in} \dot{x}_1 + x_{1in}^2 x_4) \\
0 =& - \frac{K_1 \dot{x}_2}{K_1 + x_{2in}} + \frac{k_1 x_1 x_{2in}}{K_1 + x_{2in}} - \frac{x_{2in} \dot{x}_2}{K_1 + x_{2in}} \\
0 =& - \frac{K_2 \dot{x}_3}{K_2 + x_3} - \frac{k_2 x_2 x_3}{K_2 + x_3} - \frac{x_3 \dot{x}_3}{K_2 + x_3} \\
0 =& - \dot{x}_4
\end{aligned}$$

$N = 6$, Limit Evaluated: $K_3 \to \infty, \quad k_3 \to \infty.$

New Parameter: K_{33}.

$$
\begin{aligned}
0 = &- \dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} + \frac{x_{1in}x_3}{K_{33}} \\
0 = &- \frac{K_1\dot{x}_2}{K_1 + x_{2in}} + \frac{k_1x_1x_{2in}}{K_1 + x_{2in}} - \frac{x_{2in}\dot{x}_2}{K_1 + x_{2in}} \\
0 = &- \frac{K_2\dot{x}_3}{K_2 + x_3} - \frac{k_2x_2x_3}{K_2 + x_3} - \frac{x_3\dot{x}_3}{K_2 + x_3} \\
0 = &- \dot{x}_4
\end{aligned}
$$

$N = 5$, Limit Evaluated: $K_2 \to \infty, \quad k_2 \to \infty$.

New Parameter: K_{22}.

$$
\begin{aligned}
0 = &- \dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} + \frac{x_{1in}x_3}{K_{33}} \\
0 = &- \frac{K_1\dot{x}_2}{K_1 + x_{2in}} + \frac{k_1x_1x_{2in}}{K_1 + x_{2in}} - \frac{x_{2in}\dot{x}_2}{K_1 + x_{2in}} \\
0 = &- \dot{x}_3 - \frac{x_2x_3}{K_{22}} \\
0 = &- \dot{x}_4
\end{aligned}
$$

$N = 4$, Limit Evaluated: $K_1 \to \infty, \quad k_1 \to \infty$.

New Parameter: K_{11}.

$$
\begin{aligned}
0 = &- \dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} + \frac{x_{1in}x_3}{K_{33}} \\
0 = &- \dot{x}_2 + \frac{x_1x_{2in}}{K_{11}} \\
0 = &- \dot{x}_3 - \frac{x_2x_3}{K_{22}} \\
0 = &- \dot{x}_4
\end{aligned}
$$

$N = 3$, Limit Evaluated: $K_{33} \to \infty$.

$$
\begin{aligned}
0 &= -\dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} \\
0 &= -\dot{x}_2 + \frac{x_1 x_{2in}}{K_{11}} \\
0 &= -\dot{x}_3 - \frac{x_2 x_3}{K_{22}} \\
0 &= -\dot{x}_4
\end{aligned}
$$

$N = 2$, Limit Evaluated: $K_{22} \to \infty$.

$$
\begin{aligned}
0 &= -\dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} \\
0 &= -\dot{x}_2 + \frac{x_1 x_{2in}}{K_{11}} \\
0 &= -\dot{x}_3 \\
0 &= -\dot{x}_4
\end{aligned}
$$

$N = 1$, Limit Evaluated: $K_{11} \to \infty$.

$$
\begin{aligned}
0 &= -\dot{x}_1 + \frac{x_{1in}x_4}{K_{44}} \\
0 &= -\dot{x}_2 \\
0 &= -\dot{x}_3 \\
0 &= -\dot{x}_4
\end{aligned}
$$

$N = 0$, Limit Evaluated: $K_{44} \to \infty$.

$$
\begin{aligned}
0 &= -\dot{x}_1 \\
0 &= -\dot{x}_2 \\
0 &= -\dot{x}_3 \\
0 &= -\dot{x}_4
\end{aligned}
$$

C.1.2 Mass Action

This example uses an enzyme substrate reaction found in Eq. C.1.2. To model this reaction, mass action dynamics were used leading to Eq. C.1.2.

$$x_1 + x_2 \Longleftrightarrow x_2 \Rightarrow x_1 + x_4 \tag{C.0}$$

$N = 3$

$$\begin{aligned}
0 =& k_c x_3 - k_f x_1 x_2 + k_r x_3 - \dot{x}_1 \\
0 =& - k_f x_1 x_2 + k_r x_3 - \dot{x}_2 \\
0 =& - k_c x_3 + k_f x_1 x_2 - k_r x_3 - \dot{x}_3 \\
0 =& k_c x_3 - \dot{x}_4
\end{aligned}$$

$N = 2$, Limit Evaluated: $k_f \to \infty, \quad k_r \to \infty.$

New Parameter: K_{fr}

New Variables: $x_5 = -x_1 + x_2, \quad x_6 = -x_1 - x_3.$

$$\begin{aligned}
0 =& - x_1^2 - x_1 x_5 - \frac{x_1}{K_{fr}} - \frac{x_6}{K_{fr}} \\
0 =& - k_c x_1 - k_c x_6 + \dot{x}_5 \\
0 =& \dot{x}_6 \\
0 =& - k_c x_1 - k_c x_6 - \dot{x}_4
\end{aligned}$$

$N = 1$, Limit Evaluated: $K_{fr} \to \infty.$

$$\begin{aligned}
0 =& - x_1^2 - x_1 x_5 \\
0 =& - k_c x_1 - k_c x_6 + \dot{x}_5 \\
0 =& \dot{x}_6 \\
0 =& - k_c x_1 - k_c x_6 - \dot{x}_4
\end{aligned}$$

$N = 0$, Limit Evaluated: $k_c \to 0$.

$$
\begin{aligned}
0 =& -x_1^2 - x_1 x_5 = x_2 \\
0 =& \dot{x}_5 \\
0 =& \dot{x}_6 \\
0 =& -\dot{x}_4
\end{aligned}
$$

C.2 Larger Models

This section contains results from running the engine on two larger models. These models are different versions of the Wnt cellular pathway. Each model has been reduced previously. The limits used in the previous reduction were applied using the engine, thus no geodesics were calculated. More details about the models are given in their respective sections.

C.2.1 Wnt (Lee)

This Wnt model is based off the findings of Lee, et al. [14]. It was originally reduced several years ago. Using the results from the past geodesic runs, this model was approximated automatically using the algorithms in this project. While the entire model was updated for each iteration, only the residual functions are shown.

N = 22

$$0 = -WX_{01}k_{01} + X_{02}k_{02} - \dot{X}_{01}$$

$$0 = WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02}$$

$$0 = -X_{03}X_{11}k_{08f} - X_{03}k_{05} + X_{04}k_{04} + X_{08}k_{08b} + X_{09}k_{10} - \dot{X}_{03}$$

$$0 = -X_{02}X_{04}k_{03} + X_{03}k_{05} - X_{04}k_{04} - X_{04}k_{06b} + X_{05}X_{06}k_{06f} - \dot{X}_{04}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - \dot{X}_{05}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - X_{06}k_{07b} + X_{07}X_{12}k_{06f} - \dot{X}_{06}$$

$$0 = X_{06}k_{07b} - X_{07}X_{11}k_{17f} - X_{07}X_{12}k_{06f} + X_{15}k_{17b} - \dot{X}_{07}$$

$$0 = X_{03}X_{11}k_{08f} - X_{08}k_{09} - X_{08}k_{08b} - \dot{X}_{08}$$

$$0 = X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09}$$

$$0 = X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10}$$

$$0 = -X_{03}X_{11}k_{08f} - X_{07}X_{11}k_{17f} + X_{08}k_{08b} - X_{11}X_{13}k_{16f}$$
$$- X_{11}k_{13} + X_{14}k_{16b} + X_{15}k_{17b} - \dot{X}_{11} + k_{12}$$

$$0 = X_{06}k_{07b} - X_{07}X_{12}k_{06f} - X_{12}k_{15} - \dot{X}_{12} + k_{14}$$

$$0 = -X_{11}X_{13}k_{16f} + X_{14}k_{16b} - \dot{X}_{13}$$

$$0 = X_{11}X_{13}k_{16f} - X_{14}k_{16b} - \dot{X}_{14}$$

$$0 = X_{07}X_{11}k_{17f} - X_{15}k_{17b} - \dot{X}_{15}$$

$N = 21$, Limit Evaluated: $k08b \to 0$.

$$
\begin{aligned}
0 =& - WX_{01}k_{01} + X_{02}k_{02} - \dot{X}_{01} \\
0 =& WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02} \\
0 =& - X_{03}X_{11}k_{08f} - X_{03}k_{05} + X_{04}k_{04} + X_{09}k_{10} - \dot{X}_{03} \\
0 =& - X_{02}X_{04}k_{03} + X_{03}k_{05} - X_{04}k_{04} - X_{04}k_{06b} + X_{05}X_{06}k_{06f} - \dot{X}_{04} \\
0 =& X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - \dot{X}_{05} \\
0 =& X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - X_{06}k_{07b} + X_{07}X_{12}k_{06f} - \dot{X}_{06} \\
0 =& X_{06}k_{07b} - X_{07}X_{11}k_{17f} - X_{07}X_{12}k_{06f} + X_{15}k_{17b} - \dot{X}_{07} \\
0 =& X_{03}X_{11}k_{08f} - X_{08}k_{09} - \dot{X}_{08} \\
0 =& X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09} \\
0 =& X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10} \\
0 =& - X_{03}X_{11}k_{08f} - X_{07}X_{11}k_{17f} - X_{11}X_{13}k_{16f} \\
& - X_{11}k_{13} + X_{14}k_{16b} + X_{15}k_{17b} - \dot{X}_{11} + k_{12} \\
0 =& X_{06}k_{07b} - X_{07}X_{12}k_{06f} - X_{12}k_{15} - \dot{X}_{12} + k_{14} \\
0 =& - X_{11}X_{13}k_{16f} + X_{14}k_{16b} - \dot{X}_{13} \\
0 =& X_{11}X_{13}k_{16f} - X_{14}k_{16b} - \dot{X}_{14} \\
0 =& X_{07}X_{11}k_{17f} - X_{15}k_{17b} - \dot{X}_{15}
\end{aligned}
$$

$N = 20$, Limit Evaluated: $k_{16f} \to \infty, \quad k_{16b} \to \infty.$

New Parameter: K_{16}.

New Variables: $x_1 = -X_{11} + X_{13}, \quad x_2 = -X_{11} - X_{14}.$

$$0 = -WX_{01}k_{01} - \dot{X}_{01} + X_{02}k_{02}$$

$$0 = WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02}$$

$$0 = -X_{03}X_{11}k_{08f} - X_{03}k_{05} - \dot{X}_{03} + X_{04}k_{04} + X_{09}k_{10}$$

$$0 = -X_{02}X_{04}k_{03} + X_{03}k_{05} - X_{04}k_{04} - X_{04}k_{06b} - \dot{X}_{04} + X_{05}X_{06}k_{06f}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - \dot{X}_{05}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - X_{06}k_{07b} - \dot{X}_{06} + X_{07}X_{12}k_{06f}$$

$$0 = X_{06}k_{07b} - X_{07}X_{11}k_{17f} - X_{07}X_{12}k_{06f} - \dot{X}_{07} + X_{15}k_{17b}$$

$$0 = X_{03}X_{11}k_{08f} - X_{08}k_{09} - \dot{X}_{08}$$

$$0 = X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09}$$

$$0 = X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10}$$

$$0 = -X_{11}^2 - X_{11}x_1 - \frac{X_{11}}{K_{16}} - \frac{x_2}{K_{16}}$$

$$0 = X_{06}k_{07b} - X_{07}X_{12}k_{06f} - X_{12}k_{15} - \dot{X}_{12} + k_{14}$$

$$0 = -X_{03}X_{11}k_{08f} - X_{07}X_{11}k_{17f} - X_{11}k_{13} + X_{15}k_{17b} + k_{12} + \dot{x}_1$$

$$0 = -X_{03}X_{11}k_{08f} - X_{07}X_{11}k_{17f} - X_{11}k_{13} + X_{15}k_{17b} + k_{12} + \dot{x}_2$$

$$0 = X_{07}X_{11}k_{17f} - X_{15}k_{17b} - \dot{X}_{15}$$

$N = 19$, Limit Evaluated: $k17f \to \infty, \quad k17b \to \infty.$

New Parameter: K_{17}.

New Variables: $x_3 = -X_{07} - x_1, \quad x_4 = -X_{07} - x_2, \quad x_5 = -X_{07} - X_{15}.$

$$0 = - WX_{01}k_{01} - \dot{X}_{01} + X_{02}k_{02}$$

$$0 = WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02}$$

$$0 = - X_{03}X_{11}k_{08f} - X_{03}k_{05} - \dot{X}_{03} + X_{04}k_{04} + X_{09}k_{10}$$

$$0 = - X_{02}X_{04}k_{03} + X_{03}k_{05} - X_{04}k_{04} - X_{04}k_{06b} - \dot{X}_{04} + X_{05}X_{06}k_{06f}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - \dot{X}_{05}$$

$$0 = X_{02}X_{04}k_{03} + X_{04}k_{06b} - X_{05}X_{06}k_{06f} - X_{06}k_{07b} - \dot{X}_{06} + X_{07}X_{12}k_{06f}$$

$$0 = - X_{07}X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}}$$

$$0 = X_{03}X_{11}k_{08f} - X_{08}k_{09} - \dot{X}_{08}$$

$$0 = X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09}$$

$$0 = X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10}$$

$$0 = X_{07}X_{11} - X_{11}^2 + X_{11}x_3 + \frac{X_{07}}{K_{16}} - \frac{X_{11}}{K_{16}} + \frac{x_4}{K_{16}}$$

$$0 = X_{06}k_{07b} - X_{07}X_{12}k_{06f} - X_{12}k_{15} - \dot{X}_{12} + k_{14}$$

$$0 = X_{03}X_{11}k_{08f} + X_{06}k_{07b} - X_{07}X_{12}k_{06f} + X_{11}k_{13} - k_{12} + \dot{x}_3$$

$$0 = X_{03}X_{11}k_{08f} + X_{06}k_{07b} - X_{07}X_{12}k_{06f} + X_{11}k_{13} - k_{12} + \dot{x}_4$$

$$0 = X_{06}k_{07b} - X_{07}X_{12}k_{06f} + \dot{x}_5$$

$N = 18$, Limit Evaluated: $k_{06f} \to \infty, \quad k_{06b} \to \infty, \quad k_{03} \to \infty.$

New Parameters: $K_{63}, \quad K_{06}.$

New Variables: $x_6 = -X_{04} - X_{05}, \quad x_7 = -X_{04} - X_{06}.$

$$
\begin{aligned}
0 =& - WX_{01}k_{01} - \dot{X}_{01} + X_{02}k_{02} \\
0 =& WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02} \\
0 =& - X_{03}X_{11}k_{08f} - X_{03}k_{05} - \dot{X}_{03} + X_{04}k_{04} + X_{09}k_{10} \\
0 =& - \frac{X_{02}X_{04}}{K_{63}} + X_{04}^2 + X_{04}x_6 + X_{04}x_7 - \frac{X_{04}}{K_{06}} + x_6x_7 \\
0 =& X_{03}k_{05} - X_{04}k_{04} + \dot{x}_6 \\
0 =& X_{03}k_{05} - X_{04}k_{04} + X_{04}k_{07b} + X_{07}X_{12}k_{07f} + k_{07b}x_7 + \dot{x}_7 \\
0 =& - X_{07}X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}} \\
0 =& X_{03}X_{11}k_{08f} - X_{08}k_{09} - \dot{X}_{08} \\
0 =& X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09} \\
0 =& X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10} \\
0 =& X_{07}X_{11} + \frac{X_{07}}{K_{16}} - X_{11}^2 + X_{11}x_3 - \frac{X_{11}}{K_{16}} + \frac{x_4}{K_{16}} \\
0 =& - X_{04}k_{07b} - X_{07}X_{12}k_{07f} - X_{12}k_{15} - \dot{X}_{12} - k_{07b}x_7 + k_{14} \\
0 =& X_{03}X_{11}k_{08f} - X_{04}k_{07b} - X_{07}X_{12}k_{07f} + X_{11}k_{13} - k_{07b}x_7 - k_{12} + \dot{x}_3 \\
0 =& X_{03}X_{11}k_{08f} - X_{04}k_{07b} - X_{07}X_{12}k_{07f} + X_{11}k_{13} - k_{07b}x_7 - k_{12} + \dot{x}_4 \\
0 =& - X_{04}k_{07b} - X_{07}X_{12}k_{07f} - k_{07b}x_7 + \dot{x}_5
\end{aligned}
$$

$N = 17$, Limit Evaluated: $k_{15} \to \infty, \quad k_{14} \to \infty$.

New Parameter: K_{1514}.

$$0 = -WX_{01}k_{01} - \dot{X}_{01} + X_{02}k_{02}$$

$$0 = WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02}$$

$$0 = -X_{03}X_{11}k_{08f} - X_{03}k_{05} - \dot{X}_{03} + X_{04}k_{04} + X_{09}k_{10}$$

$$0 = -\frac{X_{02}X_{04}}{K_{63}} + X_{04}^2 + X_{04}x_6 + X_{04}x_7 - \frac{X_{04}}{K_{06}} + x_6x_7$$

$$0 = X_{03}k_{05} - X_{04}k_{04} + \dot{x}_6$$

$$0 = X_{03}k_{05} - X_{04}k_{04} + X_{04}k_{07b} + X_{07}X_{12}k_{07f} + k_{07b}x_7 + \dot{x}_7$$

$$0 = -X_{07}X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}}$$

$$0 = X_{03}X_{11}k_{08f} - X_{08}k_{09} - \dot{X}_{08}$$

$$0 = X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09}$$

$$0 = X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10}$$

$$0 = X_{07}X_{11} + \frac{X_{07}}{K_{16}} - X_{11}^2 + X_{11}x_3 - \frac{X_{11}}{K_{16}} + \frac{x_4}{K_{16}}$$

$$0 = -X_{12} + \frac{1}{K_{1514}}$$

$$0 = X_{03}X_{11}k_{08f} - X_{04}k_{07b} - X_{07}X_{12}k_{07f} + X_{11}k_{13} - k_{07b}x_7 - k_{12} + \dot{x}_3$$

$$0 = X_{03}X_{11}k_{08f} - X_{04}k_{07b} - X_{07}X_{12}k_{07f} + X_{11}k_{13} - k_{07b}x_7 - k_{12} + \dot{x}_4$$

$$0 = -X_{04}k_{07b} - X_{07}X_{12}k_{07f} - k_{07b}x_7 + \dot{x}_5$$

$N = 16$, Limit Evaluated: $k_{08f} \to \infty, \quad k_{05} \to \infty.$

New Parameter: K_{8f5}.

New Variables: $x_8 = x_6 - x_7, \quad x_9 = -X_{08} - x_3, \quad x_{10} = -X_{08} - x_4, \quad \tilde{X}_{03} = \frac{X_{03}}{\varepsilon}.$

$$
\begin{aligned}
0 =& - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02} \\
0 =& W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02} \\
0 =& - \tilde{X}_{03} X_{11} - \frac{\tilde{X}_{03}}{K_{8f5}} + X_{04} k_{04} + X_{09} k_{10} \\
0 =& - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_7 + X_{04} x_8 - \frac{X_{04}}{K_{06}} + x_7^2 + x_7 x_8 \\
0 =& \frac{\tilde{X}_{03}}{K_{8f5}} - X_{04} k_{04} + \dot{x}_7 + \dot{x}_8 \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_7 + \dot{x}_8 \\
0 =& - X_{07} X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}} \\
0 =& \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08} \\
0 =& X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09} \\
0 =& X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10} \\
0 =& X_{07} X_{11} + \frac{X_{07}}{K_{16}} - X_{08} X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 - X_{11} x_9 - \frac{X_{11}}{K_{16}} - \frac{x_{10}}{K_{16}} \\
0 =& - X_{12} + \frac{1}{K_{1514}} \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_7 + k_{12} + \dot{x}_9 \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_7 + k_{12} + \dot{x}_{10} \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_7 + \dot{x}_5
\end{aligned}
$$

$N = 15$, Limit Evaluated: $K_{8f5} \to 0, \quad k_{04} \to \infty$.

New Parameter: K_{8f54}.

New Variables: $x_{11} = x_7 + x_8$.

$$
\begin{aligned}
0 =& - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02} \\
0 =& W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02} \\
0 =& - \tilde{X}_{03} + X_{04} K_{8f54} \\
0 =& - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + X_{04} x_{11} + X_{04} x_7 - \frac{X_{04}}{K_{06}} + x_{11} x_7 \\
0 =& - \tilde{X}_{03} X_{11} + X_{09} k_{10} + \dot{x}_{11} \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_7 + \dot{x}_{11} - \dot{x}_7 \\
0 =& - X_{07} X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}} \\
0 =& \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08} \\
0 =& X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09} \\
0 =& X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10} \\
0 =& X_{07} X_{11} + \frac{X_{07}}{K_{16}} - X_{08} X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 - X_{11} x_9 - \frac{X_{11}}{K_{16}} - \frac{x_{10}}{K_{16}} \\
0 =& - X_{12} + \frac{1}{K_{1514}} \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_7 + k_{12} + \dot{x}_9 \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_7 + k_{12} + \dot{x}_{10} \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_7 + \dot{x}_5
\end{aligned}
$$

$N = 14$, Limit Evaluated: $K_{8f54} \to \infty, \quad K_{06} \to \infty.$

New Parameter: K_{8f546}.

New Variables: $x_{12} = x_{11} - x_7$.

$$
\begin{aligned}
0 =& - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02} \\
0 =& W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02} \\
0 =& X_{04} K_{8f546} \\
0 =& - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_{11} - X_{04} x_{12} + x_{11}^2 - x_{11} x_{12} \\
0 =& - \tilde{X}_{03} X_{11} + X_{09} k_{10} + \dot{x}_{11} \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_{11} + k_{07b} x_{12} + \dot{x}_{12} \\
0 =& - X_{07} X_{11} - \frac{X_{07}}{K_{17}} - \frac{x_5}{K_{17}} \\
0 =& \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08} \\
0 =& X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09} \\
0 =& X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10} \\
0 =& X_{07} X_{11} + \frac{X_{07}}{K_{16}} - X_{08} X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 - X_{11} x_9 - \frac{X_{11}}{K_{16}} - \frac{x_{10}}{K_{16}} \\
0 =& - X_{12} + \frac{1}{K_{1514}} \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_{11} - k_{07b} x_{12} + k_{12} + \dot{x}_9 \\
0 =& X_{04} k_{07b} + X_{07} X_{12} k_{07f} - X_{08} k_{09} - X_{11} k_{13} + k_{07b} x_{11} - k_{07b} x_{12} + k_{12} + \dot{x}_{10} \\
0 =& - X_{04} k_{07b} - X_{07} X_{12} k_{07f} - k_{07b} x_{11} + k_{07b} x_{12} + \dot{x}_5
\end{aligned}
$$

$N = 13$, Limit Evaluated: $k_{07f} \to \infty, \quad K_{17} \to \infty.$

New Parameter: K_{7f17}.

New Variables: $x_{13} = x_{12} + x_9 \quad x_{14} = x_{10} + x_{12} \quad x_{15} = x_{12} - x_5$.

$$0 = -WX_{01}k_{01} - \dot{X}_{01} + X_{02}k_{02}$$

$$0 = WX_{01}k_{01} - X_{02}k_{02} - \dot{X}_{02}$$

$$0 = X_{04}K_{8f546}$$

$$0 = -\frac{X_{02}X_{04}}{K_{63}} + X_{04}^2 + 2X_{04}x_{11} - X_{04}x_{12} + x_{11}^2 - x_{11}x_{12}$$

$$0 = -\tilde{X}_{03}X_{11} + X_{09}k_{10} + \dot{x}_{11}$$

$$0 = -X_{04}k_{07b} - \tilde{X}_{07}X_{12} - k_{07b}x_{11} + k_{07b}x_{12} + \dot{x}_{12}$$

$$0 = -\tilde{X}_{07}X_{11} - K_{7f17}x_{12} + K_{7f17}x_{15}$$

$$0 = \tilde{X}_{03}X_{11} - X_{08}k_{09} - \dot{X}_{08}$$

$$0 = X_{08}k_{09} - X_{09}k_{10} - \dot{X}_{09}$$

$$0 = X_{09}k_{10} - X_{10}k_{11} - \dot{X}_{10}$$

$$0 = -X_{08}X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 + X_{11}x_{12} - X_{11}x_{13} - \frac{X_{11}}{K_{16}} + \frac{x_{12}}{K_{16}} - \frac{x_{14}}{K_{16}}$$

$$0 = -X_{12} + \frac{1}{K_{1514}}$$

$$0 = -X_{08}k_{09} - X_{11}k_{13} + k_{12} + \dot{x}_{13}$$

$$0 = -X_{08}k_{09} - X_{11}k_{13} + k_{12} + \dot{x}_{14}$$

$$0 = \dot{x}_{15}$$

$N = 12$, Limit Evaluated: $K_{8f546} \to \infty, \quad k_{07b} \to \infty.$

New Parameter: $K_{8f5467b}$.

New Variable: $\tilde{X}_{07} = \frac{X_{07}}{\varepsilon}$.

$$0 = - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02}$$

$$0 = W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02}$$

$$0 = X_{04}$$

$$0 = - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_{11} - X_{04} x_{12} + x_{11}^2 - x_{11} x_{12}$$

$$0 = - \tilde{X}_{03} X_{11} + X_{09} k_{10} + \dot{x}_{11}$$

$$0 = - \frac{X_{04}}{K_{8f5467b}} - \frac{x_{11}}{K_{8f5467b}} + \frac{x_{12}}{K_{8f5467b}}$$

$$0 = - \tilde{X}_{07} X_{11} - K_{7f17} x_{12} + K_{7f17} x_{15}$$

$$0 = \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08}$$

$$0 = X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09}$$

$$0 = X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10}$$

$$0 = - X_{08} X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 + X_{11} x_{12} - X_{11} x_{13} - \frac{X_{11}}{K_{16}} + \frac{x_{12}}{K_{16}} - \frac{x_{14}}{K_{16}}$$

$$0 = - X_{12} + \frac{1}{K_{1514}}$$

$$0 = - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{13}$$

$$0 = - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{14}$$

$$0 = \dot{x}_{15}$$

$N = 11$, Limit Evaluated: $K_{7f17} \to \infty, \quad K_{1514} \to \infty.$

New Parameter: $K_{7f171514}$.

$$0 = - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02}$$

$$0 = W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02}$$

$$0 = X_{04}$$

$$0 = - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_{11} - X_{04} x_{12} + x_{11}^2 - x_{11} x_{12}$$

$$0 = - \tilde{X}_{03} X_{11} + X_{09} k_{10} + \dot{x}_{11}$$

$$0 = - \frac{X_{04}}{K_{8f5467b}} - \frac{x_{11}}{K_{8f5467b}} + \frac{x_{12}}{K_{8f5467b}}$$

$$0 = - K_{7f171514} x_{12} + K_{7f171514} x_{15}$$

$$0 = \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08}$$

$$0 = X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09}$$

$$0 = X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10}$$

$$0 = - X_{08} X_{11} - \frac{X_{08}}{K_{16}} - X_{11}^2 + X_{11} x_{12} - X_{11} x_{13} - \frac{X_{11}}{K_{16}} + \frac{x_{12}}{K_{16}} - \frac{x_{14}}{K_{16}}$$

$$0 = - X_{12}$$

$$0 = - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{13}$$

$$0 = - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{14}$$

$$0 = \dot{x}_{15}$$

$N = 10$, Limit Evaluated: $K_{16} \to 0$.

$$
\begin{aligned}
0 =& - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02} \\
0 =& W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02} \\
0 =& X_{04} \\
0 =& - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_{11} - X_{04} x_{12} + x_{11}^2 - x_{11} x_{12} \\
0 =& - \tilde{X}_{03} X_{11} + X_{09} k_{10} + \dot{x}_{11} \\
0 =& - \frac{X_{04}}{K_{8f5467b}} - \frac{x_{11}}{K_{8f5467b}} + \frac{x_{12}}{K_{8f5467b}} \\
0 =& - K_{7f171514} x_{12} + K_{7f171514} x_{15} \\
0 =& \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08} \\
0 =& X_{08} k_{09} - X_{09} k_{10} - \dot{X}_{09} \\
0 =& X_{09} k_{10} - X_{10} k_{11} - \dot{X}_{10} \\
0 =& - X_{08} - X_{11} + x_{12} - x_{14} \\
0 =& - X_{12} \\
0 =& - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{13} \\
0 =& - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{14} \\
0 =& \dot{x}_{15}
\end{aligned}
$$

$N = 9$, Limit Evaluated: $k_{10} \to 0$.

$$
\begin{aligned}
0 =& - W X_{01} k_{01} - \dot{X}_{01} + X_{02} k_{02} \\
0 =& W X_{01} k_{01} - X_{02} k_{02} - \dot{X}_{02} \\
0 =& X_{04} \\
0 =& - \frac{X_{02} X_{04}}{K_{63}} + X_{04}^2 + 2 X_{04} x_{11} - X_{04} x_{12} + x_{11}^2 - x_{11} x_{12} \\
0 =& - \tilde{X}_{03} X_{11} + \dot{x}_{11} \\
0 =& - \frac{X_{04}}{K_{8f5467b}} - \frac{x_{11}}{K_{8f5467b}} + \frac{x_{12}}{K_{8f5467b}} \\
0 =& - K_{7f171514} x_{12} + K_{7f171514} x_{15} \\
0 =& \tilde{X}_{03} X_{11} - X_{08} k_{09} - \dot{X}_{08} \\
0 =& X_{08} k_{09} - \dot{X}_{09} \\
0 =& - X_{10} k_{11} - \dot{X}_{10} \\
0 =& - X_{08} - X_{11} + x_{12} - x_{14} \\
0 =& - X_{12} \\
0 =& - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{13} \\
0 =& - X_{08} k_{09} - X_{11} k_{13} + k_{12} + \dot{x}_{14} \\
0 =& \dot{x}_{15}
\end{aligned}
$$

To approximate this model further, parameters were added which are not included in the original residual functions. Therefore, the engine could no longer continue automatically.

C.2.2 Wnt (Jensen)

This Wnt model is based off the findings of Jensen, et al. [11]. Using the results from the past geodesic runs, this model was approximated automatically using the algorithms in this project. While the entire model was updated for each iteration, only the residual functions are shown.

Note: in this model, the initial values for the variables were treated as adjustable parameters. For those with limits that contain parameters with $_{init}$ appended to the name, those initial conditions were adjusted. Therefore, no changes in the residual functions may be seen.

$N = 20$.

$$
\begin{aligned}
0 =& BG_AC_{fc} - C_{bc}C - C\alpha - \dot{C} \\
0 =& AGC_{fGA} - BG_AC_{fc} + C\alpha - G_AC_{bGA} - \dot{G_A} \\
0 =& - BG_AC_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_AC_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} + A_LC_{bAL} - \dot{A} + A_mC_{tlA} + G_AC_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2C_{tsA} \\
0 =& ALC_{fAL} - A_LC_{bAL} - A_Lv - \dot{A_L} \\
0 =& - ALC_{fAL} + A_LC_{bAL} + A_Lv - \dot{L}
\end{aligned}
$$

$N = 19$, Limit Evaluated: $C_{bc} \to 0$.

$$
\begin{aligned}
0 =& BG_AC_{fc} - C\alpha - \dot{C} \\
0 =& AGC_{fGA} - BG_AC_{fc} + C\alpha - G_AC_{bGA} - \dot{G_A} \\
0 =& - BG_AC_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_AC_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} + A_LC_{bAL} - \dot{A} + A_mC_{tlA} + G_AC_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2C_{tsA} \\
0 =& ALC_{fAL} - A_LC_{bAL} - A_Lv - \dot{A_L} \\
0 =& - ALC_{fAL} + A_LC_{bAL} + A_Lv - \dot{L}
\end{aligned}
$$

$N = 18$, Limit Evaluated: $C_{init} \to 0$.

$$
\begin{aligned}
0 =& BG_AC_{fc} - C\alpha - \dot{C} \\
0 =& AGC_{fGA} - BG_AC_{fc} + C\alpha - G_AC_{bGA} - \dot{G_A} \\
0 =& - BG_AC_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_AC_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} + A_LC_{bAL} - \dot{A} + A_mC_{tlA} + G_AC_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2C_{tsA} \\
0 =& ALC_{fAL} - A_LC_{bAL} - A_Lv - \dot{A_L} \\
0 =& - ALC_{fAL} + A_LC_{bAL} + A_Lv - \dot{L}
\end{aligned}
$$

$N = 17$, Limit Evaluated: $\alpha \to \infty$.

$$
\begin{aligned}
0 =& BG_A C_{fc} - \tilde{C} \\
0 =& AGC_{fGA} - G_A C_{bGA} - \dot{G_A} \\
0 =& - BG_A C_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_A C_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} + A_L C_{bAL} - \dot{A} + A_m C_{tlA} + G_A C_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2 C_{tsA} \\
0 =& ALC_{fAL} - A_L C_{bAL} - A_L v - \dot{A_L} \\
0 =& - ALC_{fAL} + A_L C_{bAL} + A_L v - \dot{L}
\end{aligned}
$$

$N = 16$, Limit Evaluated: $L_{init} \to 0$.

$$
\begin{aligned}
0 =& BG_A C_{fc} - \tilde{C} \\
0 =& AGC_{fGA} - G_A C_{bGA} - \dot{G_A} \\
0 =& - BG_A C_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_A C_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} + A_L C_{bAL} - \dot{A} + A_m C_{tlA} + G_A C_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2 C_{tsA} \\
0 =& ALC_{fAL} - A_L C_{bAL} - A_L v - \dot{A_L} \\
0 =& - ALC_{fAL} + A_L C_{bAL} + A_L v - \dot{L}
\end{aligned}
$$

$N = 15$, Limit Evaluated: $C_{bAL} \to 0$.

$$
\begin{aligned}
0 =& BG_A C_{fc} - \tilde{C} \\
0 =& AGC_{fGA} - G_A C_{bGA} - \dot{G}_A \\
0 =& - BG_A C_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_A C_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} - \dot{A} + A_m C_{tlA} + G_A C_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2 C_{tsA} \\
0 =& ALC_{fAL} - A_L v - \dot{A}_L \\
0 =& - ALC_{fAL} + A_L v - \dot{L}
\end{aligned}
$$

$N = 14$, Limit Evaluated: $B_{init} \to 0$.

$$
\begin{aligned}
0 =& BG_A C_{fc} - \tilde{C} \\
0 =& AGC_{fGA} - G_A C_{bGA} - \dot{G}_A \\
0 =& - BG_A C_{fc} - \dot{B} + S \\
0 =& - AGC_{fGA} + G_A C_{bGA} - \dot{G} \\
0 =& - AGC_{fGA} - ALC_{fAL} - \dot{A} + A_m C_{tlA} + G_A C_{bGA} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2 C_{tsA} \\
0 =& ALC_{fAL} - A_L v - \dot{A}_L \\
0 =& - ALC_{fAL} + A_L v - \dot{L}
\end{aligned}
$$

$N = 13$, Limit Evaluated: $C_{fGA} \to \infty, C_{bGA} \to \infty$.

New Parameter: C_{GA}.

New Variables: $x_1 = -G - GA, \quad x_2 = -A - GA$

$$
\begin{aligned}
0 =& BG_AC_{fc} - \tilde{C} \\
0 =& G_A^2 + G_Ax_1 + G_Ax_2 - \frac{G_A}{C_{GA}} + x_1x_2 \\
0 =& - BG_AC_{fc} - \dot{B} + S \\
0 =& x_{\dot{1}} \\
0 =& A_mC_{tlA} + G_ALC_{fAL} + LC_{fAL}x_2 + x_{\dot{2}} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2C_{tsA} \\
0 =& - A_Lv - \dot{A_L} - G_ALC_{fAL} - LC_{fAL}x_2 \\
0 =& A_Lv + G_ALC_{fAL} + LC_{fAL}x_2 - \dot{L}
\end{aligned}
$$

$N = 12$, Limit Evaluated: $G_{init} \to 0$.

$$
\begin{aligned}
0 =& BG_AC_{fc} - \tilde{C} \\
0 =& G_A^2 + G_Ax_1 + G_Ax_2 - \frac{G_A}{C_{GA}} + x_1x_2 \\
0 =& - BG_AC_{fc} - \dot{B} + S \\
0 =& x_{\dot{1}} \\
0 =& A_mC_{tlA} + G_ALC_{fAL} + LC_{fAL}x_2 + x_{\dot{2}} \\
0 =& - \frac{A_m}{\tau_{Am}} - \dot{A_m} + B^2C_{tsA} \\
0 =& - A_Lv - \dot{A_L} - G_ALC_{fAL} - LC_{fAL}x_2 \\
0 =& A_Lv + G_ALC_{fAL} + LC_{fAL}x_2 - \dot{L}
\end{aligned}
$$

Further approximation has been done on this model, however, the following limit requires more variable manipulation than is currently possible in the algorithm.

References

[1] Athanasios C Antoulas. *Approximation of Large-Scale Dynamical Systems*, volume 6. Siam, 2005.

[2] W Brian Arthur. Complexity and the economy. *Science*, 284(5411):107–109, 1999.

[3] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

[4] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7280–7287, 2002.

[5] Kevin S Brown, Colin C Hill, Guillermo A Calero, Christopher R Myers, Kelvin H Lee, James P Sethna, and Richard A Cerione. The statistical mechanics of complex signaling networks: nerve growth factor signaling. *Physical Biology*, 1(3):184, 2004.

[6] Fergal P Casey, Dan Baird, Qiyu Feng, Ryan N Gutenkunst, Joshua J Waterfall, Christopher R Myers, Kevin S Brown, Richard A Cerione, and James P Sethna. Optimal experimental design in an epidermal growth factor receptor signalling and down-regulation model. *IET Systems Biology*, 1(3):190–202, 2007.

[7] Andrea Caumo, Paolo Vicini, Jeffrey J Zachwieja, Angelo Avogaro, Kevin Yarasheski, Dennis M Bier, and Claudio Cobelli. Undermodeling affects minimal model indexes: insights from a two-compartment model. *American Journal of Physiology-Endocrinology And Metabolism*, 276(6):E1171–E1193, 1999.

[8] Bryan C Daniels, Yan-Jiun Chen, James P Sethna, Ryan N Gutenkunst, and Christopher R Myers. Sloppiness, robustness, and evolvability in systems biology. *Current Opinion in Biotechnology*, 19(4):389–395, 2008.

[9] Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950, 2007.

[10] JM Haile. Molecular dynamics simulation: Elementary methods. *Computers in Physics*, 7(6):625–625, 1993.

[11] Peter B Jensen, Lykke Pedersen, Sandeep Krishna, and Mogens H Jensen. A wnt oscillator model for somitogenesis. *Biophysical Journal*, 98(6):943–950, 2010.

[12] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[13] Hiroaki Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

[14] Ethan Lee, Adrian Salic, Roland Krüger, Reinhart Heinrich, and Marc W Kirschner. The roles of apc and axin derived from experimental and theoretical analysis of the wnt pathway. *PLoS Biol*, 1(1):e10, 2003.

[15] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[16] Robert D Ryne. Advanced computers and simulation. In *Particle Accelerator Conference*, pages 3229–3233. IEEE, 1993.

[17] Steven C Sherwood, Sandrine Bony, and Jean-Louis Dufresne. Spread in model climate sensitivity traced to atmospheric convective mixing. *Nature*, 505(7481):37–42, 2014.

[18] Mark K Transtrum and Peng Qiu. Model reduction by manifold boundaries. *Physical Review Letters*, 113(9):098701, 2014.

[19] Mark K Transtrum and Ping Qiu. Coarse-graining models of emergent behavior by manifold boundaries. *in submission*, 2012.

[20] Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Why are nonlinear fits to data so challenging? *Physical Review Letters*, 104(1060201), 2010.

[21] Mark K Transtrum, Benjamin B Machta, and James P Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E*, 83(3):036701, 2011.

[22] George M Whitesides and Rustem F Ismagilov. Complexity in chemistry. *Science*, 284 (5411):89–92, 1999.