Senior Thesis in Mathematics

Sigmoid Modeling with `sicegar`: Key Issues and Improvements

Author:
Mira Terdiman

Advisor:
Dr. Jo Hardin

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

May 7, 2025

Acknowledgements

I first want to thank Professor Hardin for her support on this thesis project, and for her mentorship and guidance throughout my time at Pomona. I am also thankful for my advisor, Dr. Radunskaya, without whom I would have never decided to be a math major, for her thoughtful and consistent mentorship over the past four years. Finally, I want extend my gratitude to Federica Domecq Lacroze (PO '26) for all the work she did on the project before I started, as well as to my other professors, coaches, family, and friends for their positive influence on my time here at Pomona. I am grateful and lucky to have such a strong support system.

Abstract

Escheria coli (E. coli), a bacterium often found in the intenstinal tract of warm-blooded vertebrates, adapts to environmental stressors (e.g., osmotic or temperature changes) by altering the patterns of its gene expression. In response to stress, the abundance of RNA associated with particular genes will change in the cell [2], and this process can be measured using RNA-sequencing data. In order to model RNA-sequencing data, Adams et. al 2023 used `sicegar`, a package in `R` which fits sigmoidal curves to time course data using the Levenberg-Marquardt Algorithm. In this thesis, we identify key issues in `sicegar`'s estimation techniques to provide suggestions on best practices to `sicegar` users. We explore three potential roadblocks: (1) Corner cases in sigmoidal and double sigmoidal models, (2) Biases in the Levenberg-Marquardt Algorithm, and (3) Setting hyperparameters for model estimation in `sicegar`. In order to assess how accurately `sicegar` estimates model parameters, we use a simulation procedure to generate noisy data from known parameters, which allows us to identify potential estimation biases. We conclude that setting appropriate hyperparameters, both the upper bounds and starting values of the nonlinear least squares estimation, are highly influential on `sicegar`'s ability to estimate the underlying sigmoid model of RNA-seq data. We are able to get `sicegar` to work extremely well when we set our hyperparameters directly from the data. Therefore, we believe there is more work to be done to explore techniques for setting hyperparameters, such as estimating them directly from RNA-seq data, to ensure the package works reliably.

Contents

List of Figures

Chapter 1

Introduction

Bacteria often respond to changing environments or stressors by altering the pattern of their gene expression, which functions as a survival mechanism to relieve stress and adapt to unfavorable conditions [8]. Many of these stress-responses are triggered by particular conditions, such as exposure to increased temperature, salt, or starvation.

Escherichia coli (*E. coli*) is a bacterial species that includes both pathogenic and nonpathogenic strains, and is often found in the intestinal tract of warm-blooded vertebrates [3]. Like other bacteria, *E. coli* alters its gene transcription patterns in response to external stimuli and stressors. The regulation of gene expression in *E. coli* occurs at the transcriptional level and is controlled by RNA polymerase. Sigma factors, which are subunits of all bacterial RNA polymerases, are responsible for directing the polymerase to specific regions on DNA, called promoters, to initiate transcription [9]. RNA polymerase then transcribes sections of the cell's DNA into RNA, which is then translated into a protein.

The sigma factor RpoS regulates the expression for over 1000 different genes in *E. coli*. RpoS activates in response to stress and accumulates in the cell after the initial exposure. The timing of RpoS activation varies depending on the stressor to which the cell is exposed (e.g., stationary phase, high osmolarity, or cold shock). Since the availability of RpoS varies in time, the expression for the genes that it regulates similarly varies. Additionally, the transcriptional response to RpoS accumulation varies based on gene-type. While some genes achieve high levels of transcription in response to low levels of RpoS, others only achieve high levels of transcription in response to high levels of RpoS [1].

Adams et al. (2023) build upon this understanding of RpoS to investigate the expression of over 1,000 genes in *E. coli* and their response to three distinct stressors: high osmolarity, low temperature, and transition to stationary phase. They hypothesized that the transcriptional response of *E. coli* cells will differ across gene sensitivity classes and stressor type [2]. To understand how gene expression differed across gene and stressor type, gene expression was measured using RNA-Sequencing data.

To compare the transcriptional timing of different genes, Adams et al. (2023) were concerned with extracting the onset time of transcription. When a gene is activated via RpoS for transcription, the RNA polymerase synthesizes RNA molecules, so their abundance increases. Onset time marks the moment where the half-maximal abundance of RNA is reached, and serves as a readout of the transcriptional response regulated by RpoS. Onset time can help us measure how quickly a gene responds after transcription is initiated, which allows us to compare transcriptional timing of different genes across different stress conditions.

Adams et. al (2023) used `sicegar` to estimate onset time. `sicegar` is an R package that fits a sigmoidal model to time-intensity data [5]. Sigmoid functions model time versus intensity, where we assume intensity increases over time until a max saturation is reached. In a single sigmoid model, this max saturation is asymptotically approached as time goes to infinity, whereas in a double sigmoid model, intensity decreases down to a lower asymptotic value. Sigmoid functions are useful to us in approximating the biological phenomenon of transcription, where the RNA sequence for a specific gene accumulates in the cell.

Measuring and reporting onset time was crucial in supporting the Adams' (2023) hypotheses that the transcriptional timing of RpoS-regulated genes would differ across stressor types. Adams found that the temporal dynamics of RpoS levels (when and how rapidly it accumulates) drive stress-specific patterns of gene expression, affecting the onset time of differentially expressed genes. The median onset time of RpoS-regulated genes varied across the three stressors. Onset time functioned as a measure of transcriptional timing, and was dependent upon `sicegar` estimates.

It would be fairly simple to accept `sicegar` results without further investigation. However, given the biological importance of onset time, a deeper exploration of the package is necessary to determine the accuracy of its parameter estimation techniques. In order to uncover biases, we are presented with a problem when using real data: how can we know whether or not a pa-

rameter estimation is accurate, since the reason we fit our model is to unveil these parameters in the first place?

Simulation, where we generate noisy data from a known, underlying model, can help to measure bias and variability. After feeding simulated data into `sicegar`, we have the ability to compare the fitted parameters to ones that we know to be true. This past summer, Professor Jo Hardin and Federica Domecq Lacroze investigated the bias in `sicegar`'s estimation of onset time using this technique. After running numerous simulations, Prof. Hardin and Federica found that the optimization routine consistently underestimated onset time, especially as the data became noisier.

In this thesis, we build upon the foundation they set to ask: why? Can we uncover the reason that the estimation of onset time is biased? We take three main approaches to the problem:

1. **Further Exploration of the Sigmoid and Double Sigmoid Functions:** By gaining a deeper understanding of the mathematical behavior of sigmoidal functions, we explore what parameters conditions impact onset time and our ability to measure it.

2. **Levenberg-Marquardt Algorithm:** `sicegar` uses the nonlinear-least squares Levenberg-Marquardt algorithm to estimate all model parameters. We investigated this optimization routine to determine whether it presents any potential biases.

3. `sicegar`: Combining the previous two approaches, we examined `sicegar` functions via simulations to better understand how the package returns the parameter estimations. We also determined that some user-control of the optimization settings improved estimation accuracy.

Ultimately, our goal was to provide recommendations to scientists using this package to model time-course data with biological implications, by properly quantifying and describing the biases it has, as well as offering solutions to combat them.

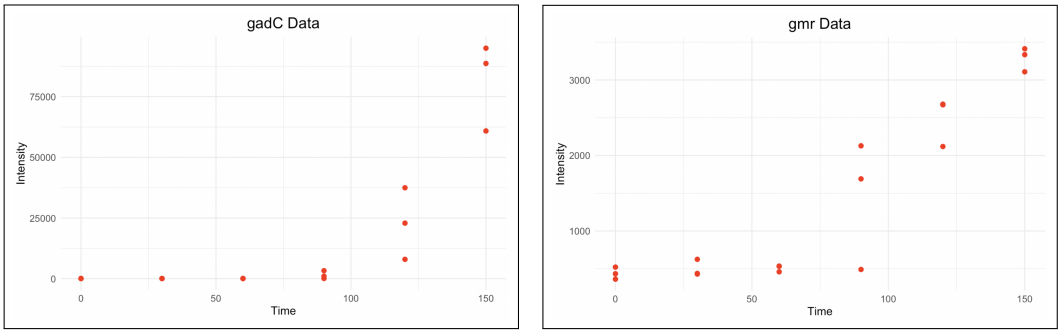
Chapter 2

Sigmoid Models

Sigmoid functions are used to model time versus intensity, where we assume intensity increases over time until a max saturation is reached. They are employed in various fields, including biology. In our case, the sigmoid function is a useful tool for modeling the level of RNA expression in *E. coli* cells after the onset of stress. In this chapter, we characterize two main types of sigmoid functions, single and double, and describe their relevance to our project.

2.1 Primer on RNA-Sequencing Data

To understand why sigmoid models are useful to our project, we first need to understand the structure of RNA-Sequencing (RNA-seq) data. RNA-seq is a technique that examines the quantity of RNA in a biological sample using next-generation sequencing, and can indicate which genes in a cell are turned on or off at a given time [10]. The data provide a snapshot in time of the abundance of RNA associated with a particular gene in the cell. Therefore, we can use RNA-seq data to measure RNA abundance over time. Figure 2.1 illustrates the form of RNA-seq data for two genes in *E. coli* studied in Adams et. al (2023).



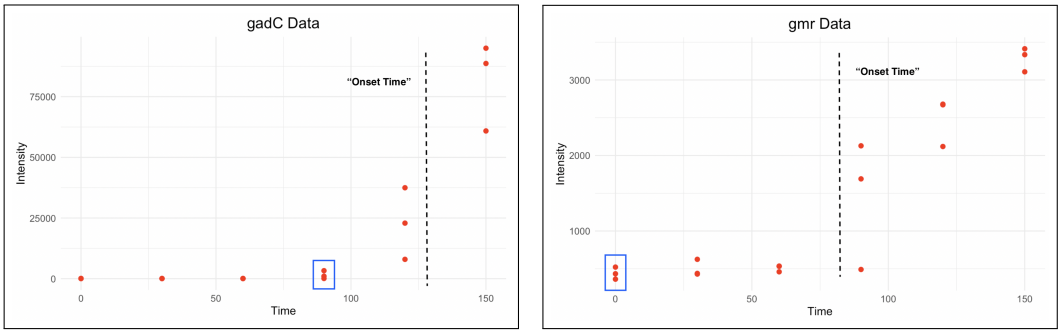
(a) gadC Gene (b) gmr Gene

Figure 2.1: RNA-seq data for gadC and gmr genes in *E. coli* cells after undergoing cell starvation. Multiple replicates are taken at each time point. Time, in minutes, is plotted along the x-axis, and the intensity/abundance of RNA in the cell is plotted on the y-axis.

2.1.1 Onset Time

When working with time-course RNA-seq data, specifically from Adams et. al (2023), there are two important considerations. The first is that each time point will have multiple intensity values, because replicates were taken in the experiment.

Secondly, we are concerned with the measurement of **onset time**, which is defined as the time when the half-maximal abundance of RNA is reached in the cell [2]. Mathematically, that is the argument of the function at the midpoint between the minimum and maximum intensity levels. Figure 2.2 shows us the same RNA-seq data as Figure 2.1, along with the approximate measure of onset time.



(a) gadC Gene (b) gmr Gene

Figure 2.2: RNA-seq data for gadC and gmr genes in *E. coli* with onset time labeled. Note that the onset time (dashed line) is later for gadC than gmr. Additionally, as you can see clearly highlighted in the boxes, each time point has multiple replicates.

2.2 Sigmoid Model

To model RNA-seq data, we can employ sigmoid models. A single sigmoid function models time versus intensity, where intensity increases until reaching an asymptotic level, where it remains as time approaches infinity. A common single-sigmoid function is the logistic function,

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}. \tag{2.1}$$

The logistic function is a continuous-deterministic model of population growth which was first published by Pierre Verhulst in 1838 [13]. Notably, it supplements the exponential growth function by adding a carrying capacity L which represents the maximum population size given available resources.

We can apply this model to the rise of RNA expression for a particular gene in *E. coli* cells post-stress exposure. Equation 2.2 characterizes the sigmoid function we reference in this thesis:

$$f_{ss}(x) = h_0 + \frac{h_1 - h_0}{1 + e^{-a(x-t_1)}}. \tag{2.2}$$

This function, which from now on we will refer to as the single-sigmoid function, has four key parameters:

1. $h_0 = lim_{x \to -\infty} f(x)$
2. $h_1 =$ maximum of the sigmoid curve, where $h_1 > h_0$
3. $a =$ proportional to the slope of the sigmoid curve at $x = t_1$
4. $t_1 =$ inflection point of the sigmoid curve (shown below).
5. $h_0 < h_1$

A typical single-sigmoid function is shown in Figure 2.3 with labeled parameters.

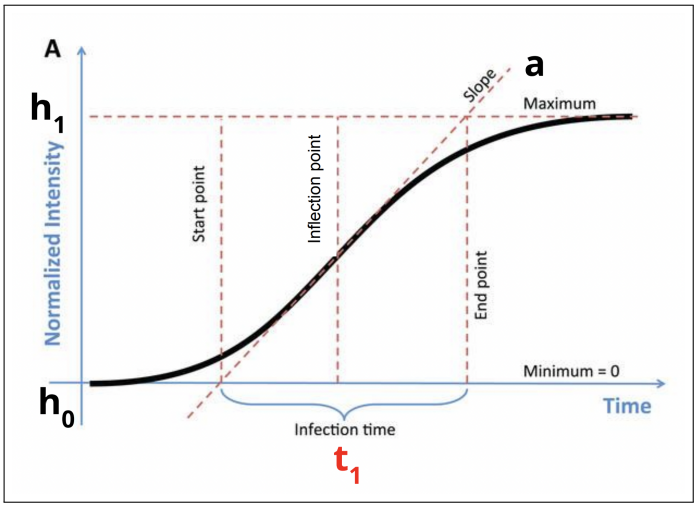


Figure 2.3: Single Sigmoid with parameters labeled [5]

For the single sigmoid function, t_1 is both the inflection point and midpoint of the sigmoid curve (aka onset time).

Claim 1. *t_1 is the inflection point and onset time of the single sigmoid curve.*

Proof. We will first take the second derivative of our single sigmoid function.

$$f''_{ss}(x) = [(h_1 - h_0)(-a^2)(e^{-a(x-t_1)})][-2(1+e^{-a(x-t_1)})^{-3}(e^{-a(x-t_1)}) + (1+e^{-a(x-t_1)})^{-2}]$$

To find the inflection point, we set this second derivative equal to 0. However, we can exclude the first half of the equation, since none of those terms can possibly equal 0.

$$-2(1+e^{-a(x-t_1)})^{-3}(e^{-a(x-t_1)})+(1+e^{-a(x-t_1)})^{-2}=0$$

$$-2(1+e^{-a(x-t_1)})^{-3}(e^{-a(x-t_1)})=-(1+e^{-a(x-t_1)})^{-2}$$

$$2(e^{-a(x-t_1)})=\frac{(1+e^{-a(x-t_1)})^{-2}}{(1+e^{-a(x-t_1)})^{-3}}$$

$$2(e^{-a(x-t_1)})=1+e^{-a(x-t_1)}$$

$$e^{-a(x-t_1)}=1$$

We then take the natural log of both sides:

$$-a(x-t_1)=0$$

$$at_1=ax$$

$$t_1=x$$

Therefore $x=t_1$ is our inflection point. We now show that the function $f_{ss}(x)$ evaluated at t_1 is equivalent to the midpoint between h_0 and h_1:

$$f_{ss}(t_1)=h_0+\frac{h_1-h_0}{1+e^{-a(t_1-t_1)}}=h_0+\frac{h_1-h_0}{2}=\frac{h_0+h_1}{2}.$$

□

Therefore, we can use the parameter t_1 of the single sigmoid model to represent onset time when working with RNA-seq data.

2.3 Double Sigmoid Model

A double sigmoid curve also models time versus intensity, however the behavior of the function is slightly different. The model first increases to a maximum intensity, but then decays to a final asymptotic level. It has one global maximum.

2.3.1 Continuous Characterization of Double Sigmoid

We can start by writing a continuous double sigmoid function by taking the product of two sigmoid functions with slopes a_1 and a_2:

$$f_{ds}(x) = \frac{1}{h_1}\left(h_0 + \frac{h_1 - h_0}{1 + e^{-a_1(x-t_1)}}\right)\left(h_2 + \frac{h_1 - h_2}{1 + e^{a_2(x-t_2)}}\right). \tag{2.3}$$

We scale the function by a factor of h_1 to ensure the maximum amplitude does not exceed h_1.

In this model we have six key parameters:

1. $h_0 = lim_{x\to-\infty} f(x)$
2. h_1 = maximum of the sigmoid curve
3. $h_2 = lim_{x\to\infty} f(x)$
4. a_1 = proportional to the slope of the sigmoid curve at $x = t_1$
5. a_2 = proportional to the slope of the sigmoid curve at $x = t_2$
6. t_1 = approximate inflection point of the first sigmoid curve
7. t_2 = approximate inflection point of the second sigmoid curve

In order to maintain the correct double sigmoidal shape as shown in Figure 2.4, the parameters must follow the restrictions:

1. $h_0 < h_2 < h_1$
2. $a_1, a_2 > 0$
3. $t_1 < t_2$

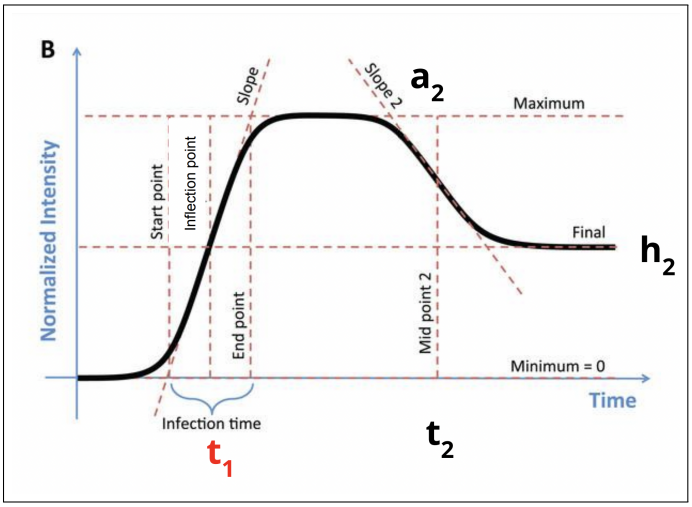


Figure 2.4: Double sigmoid model with additional parameters [5]

Since we are still concerned with estimating the onset time of the double-sigmoidal function, it is important to note that t_1 is only the approximate inflection point of the first sigmoid curve, rather than the true inflection point. In certain cases, t_1 may be very far from the correct value of onset time. Figure 2.5 highlights one such case:

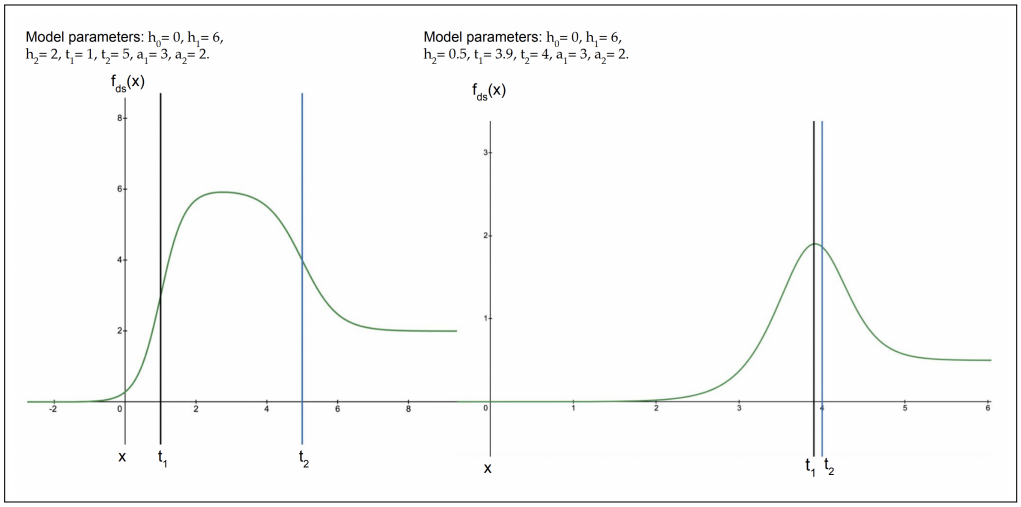


Figure 2.5: Comparing the values of t_1 across two double-sigmoid curves each defined by the parameters given on the plot. On the left, t_1 approximates the inflection point well on the curve, whereas on the right, t_1 is clearly not the inflection point.

For our project, we want to understand when t_1 is approximately the inflection point of the sigmoid curve, because that would allow us to use t_1 as a proxy for onset time.

We can estimate when t_1 is approximately the inflection point of the first sigmoid curve by analyzing the behavior of the second derivative of the double sigmoid:

$$f''_{ds}(x) = \frac{1}{h_1}(g''_1(x)g_2(x) + 2g'_1(x)g'_2(x) + g_1(x)g''_2(x)), \tag{2.4}$$

where

$$g_1(x) = \left(h_0 + \frac{h_1 - h_0}{1 + e^{-a_1(x-t_1)}}\right),$$

$$g_2(x) = \left(h_2 + \frac{h_1 - h_2}{1 + e^{a_2(x-t_2)}}\right),$$

and

$$g'_1(x) = \frac{(h_1 - h_0)a_1 e^{-a_1(x-t_1)}}{(1 + e^{-a_1(x-t_1)})^2},$$

$$g_2'(x) = \frac{(h_1 - h_2)a_2 e^{-a_2(x-t_2)}}{(1 + e^{-a_2(x-t_2)})^2}.$$

The second term of Equation 2.4, $2g_1'(x)g_2'(x)$, couples the derivatives of the two single sigmoid functions. In the single-sigmoid case, $f_{ss}''(t_1) = 0$ because $g_1''(t_1) = 0$. However, the second term moves our inflection point away from t_1 in cases when $2g_1'(x)g_2'(x)$ is very large. Therefore, for t_1 to be the inflection point of the double sigmoid, $2g_1'(x)g_2'(x)$ needs to be close to 0. This happens in two key cases:

1. The value of $g_2'(x)$ is small at $x = t_1$ (i.e, t_1 and t_2 are far apart).

   The behavior of
   $$g_2'(x) = \frac{(h_1 - h_2)a_2 e^{-a_2(x-t_2)}}{(1 + e^{-a_2(x-t_2)})^2}$$
   tells us how $g_2(x)$ changes at any given value of x. The maximum of $g_2'(x)$ occurs at $x = t_2$, and decays rapidly as x moves away from t_2. At $x = t_1$,
   $$g_2'(t_1) = \frac{(h_1 - h_2)a_2 e^{-a_2(t_1-t_2)}}{(1 + e^{-a_2(t_1-t_2)})^2}.$$

   If t_1 and t_2 move further apart, the value of $e^{-a_2(t_1-t_2)} \to 0$, so $g_2'(x) \to 0$. In other words, the value of t_1 will be in the tail of the sigmoid curve for $g_2(x)$, where it has a slope ≈ 0. Therefore, $g_2(x)$ has very little influence over the double sigmoid curve at $x = t_1$, so t_1 remains close to the inflection point of the first curve.

   We could formalize this relationship as:
   $$e^{-a_2(t_1-t_2)} = 0 \Rightarrow -a_2(t_1 - t_2) \ll 0 \Rightarrow t_2 - t_1 \gg \frac{1}{a_2}.$$

   This leads into our second consideration.

2. The value of a_2 is small enough that the slope of the second sigmoid curve does not impact that of the first sigmoid curve across all three terms of Equation 2.4.

Additionally, even if t_1 and t_2 are sufficiently apart, there are specific corner cases in which Equation 2.3 becomes non-sigmoidal. For example, for

certain values of a_1 and a_2, our function will take on a local max and local min, which makes it non-sigmoidal (shown in Figure 2.6).

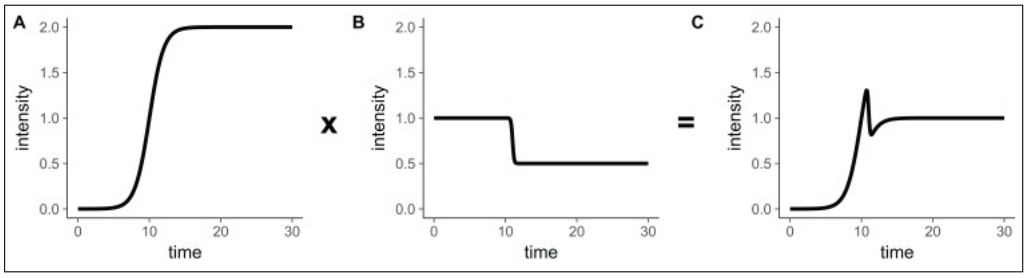


Figure 2.6: Example of two single sigmoidal functions whose product takes on a local minimum in addition to a local maximum. Plot (A) shows the first sigmoid function, $g_1(x)$, while Plot (B) shows the second sigmoid function, $g_2(x)$. Plot (C) shows their product, which is not double-sigmoidal. Parameters: $h_0 = 0, h_1 = 2, h_2 = 0.5, a_1 = 1, a_2 = 10, t_1 = 10, t_2 = 11$ [5]

There are a number of cases in which our function becomes non-double sigmoidal. For example, while holding all other parameters constant, we can observe how the number of critical points of $f_{ds}(x)$ changes as we increase the steepness of a_2. Note, we would expect a double sigmoid function to have one critical point, because there is one maximum and no minimum values. Figure 2.7 shows the number of critical points of Equation 2.5, which is a simplified version of Equation 2.3,

$$f(x) = \frac{1}{1 + e^{-a_1(x-t_1)}} + \frac{1}{1 + e^{a_2(x-t_1)}} + \frac{1}{(1 + e^{-a_1(x-t_1)})(1 + e^{a_2(x-t_2)})}, \quad (2.5)$$

when plotting 90 a_2 values between $(1, 10)$ along with a set of parameters:

1. $a_1 = 3$
2. $t_1 = 10$
3. $t_2 = 11$

The critical points of the function in Equation 2.5 will be the same as Equation 2.3, since we only scale by constants h_0, h_1, h_2.

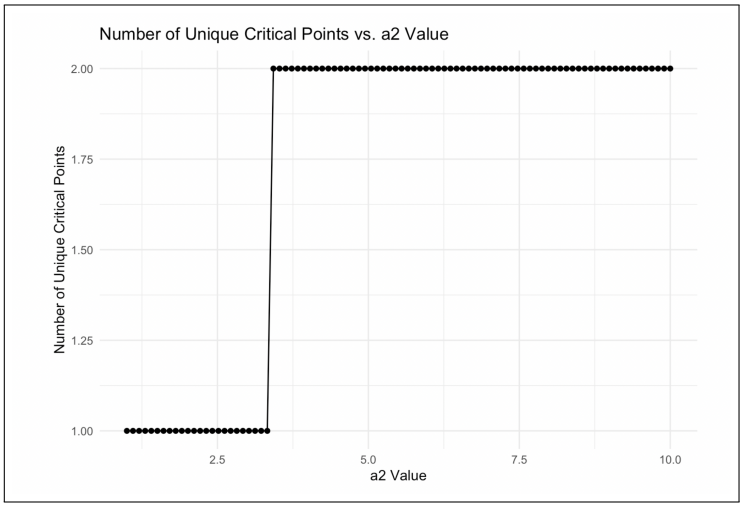


Figure 2.7: We numerically solved for the critical points of Equation 2.5 for a set of parameters $a_1 = 3, t_1 = 10, t_2 = 11$ and iterating through 90 values of $a_2 \in [1, 10]$, shown on the x-axis. We then plotted the number of critical points on the y-axis. The number of critical points that Equation 2.5 takes on increases from 1 to 2 as the value of a_2 increases.

Here, we hold a_1, t_1, and t_2 constant at 3, 10, and 11 respectively. You can see that at a certain value of a_2 (around 3.3), our function takes on 2 critical points rather than 1.

2.3.2 Piecewise

In order to address edge cases like the one in Figure 2.7, Caglar et. al (2018) proposes a piecewise double sigmoid function [5], where for some value t^*:

$$I(x) = \begin{cases} c_1 f_{base}(t) & \text{if } t \leq t^* \\ c_2 f_{base}(t) + h_2 & \text{if } t > t^* \end{cases} \tag{2.6}$$

where

$$c_1 = \frac{h_1}{f_{max}} \tag{2.7}$$

$$c_2 = \frac{h_1 - h_2}{f_{max}}. \tag{2.8}$$

The piecewise double sigmoid function begins with a base function, which is the product of two sigmoid functions:

$$f_{base}(x) = \frac{1}{1 + e^{-a_1(x-t_1)}} \cdot \frac{1}{1 + e^{a_2(x-t_2)}} \tag{2.9}$$

where $argmax_x f_{base}(x) = x^*$.

Equation 2.9 follows the same double sigmoid shape as Equation 2.3, but always decays back to zero. However, we use Equation 2.9 as the base of our piecewise function because it never attains a local minimum, like that seen in Figure 2.6 (C).

Claim 2 (Base Double Sigmoid)**.** *$f_{base}(x)$ has one local maximum.*

Proof. We follow the proof provided in Caglar et. al 2018 [5]:

We define the argument of the maximum of $f_{base}(x)$ as x^*, therefore

$$max(f_{base}(x)) = f_{base}(x^*)$$

We know that $t_2 > t_1$, so we can rewrite $t_2 = t_1 + L$, where L is some positive real number. Thus, rewrite Equation 2.9 as:

$$f_{base}(x) = \frac{1}{1 + e^{-a_1(x-t_1)}} \cdot \frac{1}{1 + e^{a_2(x-L-t_1)}}$$

By the nature of derivatives, we know that at $x = x^*$, the derivative of $f_{base}(x)$ will be 0,

$$\frac{d}{dx} f_{base}(x)\bigg|_{x=x^*} = 0$$

We define variables $u = x - t_1$ and $u^* = x^* - t_1$, and define the function $g(u) = f_{base}(u + t_1)$. Therefore, finding the roots of $g(u)$ is equivalent to finding the roots of $f_{base}(x)$, and we define $g(u)$ as:

$$g(u) = \frac{1}{1 + e^{-a_1(u)}} \cdot \frac{1}{1 + e^{a_2(u-L)}}$$

Which implies that:

$$\frac{d}{du} g(u)\bigg|_{u=u^*} = 0$$

Now, we take the derivative of $g(u)$ at u^* and set equal to 0 to solve for the critical point:

$$\frac{d}{du}g(u^*) = \frac{e^{a_1u^*+a_2(L-u^*)}[a_1(e^{a_2(L-u^*)}+1) - a_2(e^{a_1u^*}+1)]}{(e^{a_1u^*}+1)^2(e^{a_2(L-u^*)}+1)^2} = 0$$

Since we will set this derivative = 0, we are only concerned with the numerator, which we can rewrite as:

$$e^{a_2(u^*-L)-a_1u^*}(a_1 + a_1e^{-a_2(u^*-L)} - a_2 - a_2e^{a_1u^*}) = 0$$

Since $e^{a_2(u^*-L)-a_1u^*}$ can never equal 0, we want to focus on the term in the parantheses to determine our critical points. We define this section as:

$$h(u) = a_1 + a_1e^{-a_2(u-L)} - a_2 - a_2e^{a_1u}$$

To find the number of roots of $h(u)$, we first examine its limiting properties:

$$\lim_{u\to-\infty} h(u) = \infty$$
$$\lim_{u\to\infty} h(u) = -\infty$$

By the Mean Value Theorem, $h(u)$ must have at least one root.

We can also observe that $h(u)$ is strictly decreasing, where

$$h'(u) = -a_1a_2e^{-a_2(u-L)} - a_1a_2e^{a_1u} < 0$$

for any u, since a_1, a_2 are both always positive. Therefore, by Rolle's Theorem, $h(u)$ can at most one root.

Now we can show the root of $h(u)$ is a local maximum of $g(u)$ by taking the double derivative of $g(u)$:

$$\frac{d^2}{du^2}g(u) = \frac{e^{a_1u-2a_2(L-u)}}{(e^{a_1u}+1)^3 + (e^{-a_2(L-u)}+1)^3} \cdot [(a_1(e^{a_2(L-u)}+1) - a_2(e^{a_1u}+1))^2$$
$$-a_1^2e^{a_1u}(e^{a_2(L-u)}+1)^2 - a_2^2(e^{a_1u}+1)^2e^{a_2(L-u)}]$$

The term $(a_1(e^{a_2(L-u)}+1) - a_2(e^{a_1u}+1))^2$ is equivalent to $h(u)^2$, and thus equals 0 at $u = u^*$. Additionally:

$$-a_1^2 e^{a_1 u}(e^{a_2(L-u)}+1)^2 - a_2^2(e^{a_1 u}+1)^2 e^{a_2(L-u)} < 0,$$

and our first term is strictly positive. Therefore, $\frac{d^2}{du^2}g(u) < 0$ at $u = u^*$, which means that our critical point of $u = u^*$ is a local maximum.

□

To characterize the double sigmoid function, we can split $f_{base}(x)$ at $x = x^*$ and scale each base function separately to represent sigmoidal growth and sigmoidal decay. This ensures we never attain a local minimum. Our piecewise is defined as:

$$I(x) = \begin{cases} c_1 f_{base}(x) & \text{if } x \leq x^* \\ c_2 f_{base}(x) + h_2 & \text{if } x > x^* \end{cases} \tag{2.10}$$

where

$$c_1 = \frac{h_1}{f_{max}} \tag{2.11}$$

$$c_2 = \frac{h_1 - h_2}{f_{max}}. \tag{2.12}$$

Equation 2.10 will have the same behavior as Equation 2.9, since the presence of a critical point will not be impacted by simply scaling the function by constants. The derivatives of $c_1 f_{base}(t)$ and $c_2 f_{base}(t) + h_2$ will behave identically to $f_{base}(t)$, since those constants will become irrelevant when we set the derivatives = 0. For the remainder of this thesis, when describing a double-sigmoid, we will use Equation 2.10.

Chapter 3

sicegar

3.1 Overview

`sicegar`, an R package that fits sigmoidal models to time-intensity data, was created in 2018 to analyze single-cell viral growth curves [5]. After the user inputs their own data, the package categorizes it as sigmoidal. double sigmoidal, or ambiguous, and returns a set of parameters of best fit if appropriate.

The package has a number of key functions, which include the following:

3.1.1 Key Functions

1. `fitAndCategorize`

   `fitAndCategorize` is the simplest way to use `sicegar`, since it categorizes data as sigmoidal, double sigmoidal, or ambiguous, and then identifies the best-fitting parameters within that model-type. Many of the other key sicegar functions are nested in `fitAndCategorize` in order to optimize, report, and plot estimated parameters to the input data. The key inputs are:

   `dataInput, threshold-t0-max-int, startList-sm, startList-dsm, threshold-intensity-range`

   The start values and intensity thresholds can be adjusted by the user, and should represent scaled (normalized) parameters, since our data is normalized before the fitting process begins.

This leads to the first key nesting function in `fitAndCategorize`:

2. `normalizeData`

   In sicegar, data are normalized to a [0,1] interval on both axes before the fitting process begins. Normalizing the data allows the program to set universal hyperparameters (start values, upper, and lower bounds) for the non-linear least squares optimization algorithm, so that the user does not have to set initial hyperparameters to match the original scale of the data.

   `normalizeData` extracts time and intensity columns from our data input and performs a min-max normalization on both the time and intensity axes:

$$X_{norm} = \frac{X - X_{min}}{X_{range}} \tag{3.1}$$

$$Y_{norm} = \frac{Y - Y_{min}}{Y_{range}} \tag{3.2}$$

   The function returns a new, normalized dataset, as well as the parameters used to scale the data, including X_{min}, Y_{min}, etc. Storing the scaling parameters allows the package to transform the optimized parameters back into the original space for user interpretability.

3. `sigmoidalFitFunction`

   `sigmoidalFitFunction` takes our normalized data and `startList-sm`. It initializes a starting vector (either the default or user generated) and then runs an optimization of `sigmoidalFitFormula` using:

```
try(minpack.lm::nlsLM(intensity ~
sicegar::sigmoidalFitFormula(time, maximum, slopeParam, midPoint),
dataFrameInput,
start = counterDependentStartList,
control = list(maxiter = n_iterations, minFactor = min_Factor),
lower = lowerBounds, upper = upperBounds, trace = F),
silent = TRUE)
```

The optimization technique uses the Levenberg-Marquardt Algorithm (LMA), implemented in the `minpack.lm` package via the `nlsLM` function [12]. The LMA is a non-linear least squares method of approximation, which is described in Section 3.2.

4. `sigmoidalRenormalizeParameters`

   `sigmoidalRenormalizeParameters` uses the saved data scaling metrics to convert our estimated parameters back into original space. The function primarily does this for parameters P by following the formula:

$$P_{norm} = \frac{P - P_{min}}{P_{range}},$$

   therefore

$$P = P_{norm} \cdot P_{range} + P_{min}.$$

   For parameters h_0, h_1, the normalized values are determined by the y-axis (intensity), so we would have:

$$h_0 = h_{0_{norm}} \cdot Y_{range} + Y_{min},$$
$$h_1 = h_{1_{norm}} \cdot Y_{range} + Y_{min}$$

   Similarly, for the parameter t_1, the normalized values are determined by the x-axis (time), so we have:

$$t_1 = t_{1_{norm}} \cdot X_{range} + X_{min}.$$

   However, transforming the slope parameter, a, back into original space is slightly more complicated. The function assumes our slope is linearly transformed from original space into normalized space. We know that a linear slope a follows the formula:

$$a \propto \frac{Y_2 - Y_1}{X_2 - X_1}.$$

   Therefore, Equation (3.3) represents our normalized slope a_{norm}.

$$a_{norm} = \frac{\frac{Y_2 - Y_{min}}{Y_{range}} - \frac{Y_1 - Y_{min}}{Y_{range}}}{\frac{X_2 - X_{min}}{X_{range}} - \frac{X_1 - X_{min}}{X_{range}}}. \tag{3.3}$$

Simplifying this equation we get

$$a_{norm} = \frac{(Y_2 - Y_1) \cdot X_{range}}{(X_2 - X_1) \cdot Y_{range}} = a \cdot \frac{X_{range}}{Y_{range}}. \tag{3.4}$$

Therefore, to transform our slope back into original space, we can compute:

$$a = a_{norm} \cdot \frac{Y_{range}}{X_{range}}. \tag{3.5}$$

However, `sigmoidalRenormalizeParameters()` rescaling of the slope parameter follows the formula

$$a = a_{norm} \cdot \frac{1}{X_{range}}. \tag{3.6}$$

This discrepancy stems from the fact that the maximum slope of our sigmoid curve $S_{max} \neq a$, our slope parameter for the sigmoid model. Defining our sigmoid curve as,

$$f(x) = h_0 + \frac{h_1 - h_0}{1 + e^{-a(x - t_1)}} \tag{3.7}$$

we can determine the maximum slope of our function, which occurs at the midpoint $x = t_1$, by computing $f'(t_1)$:

$$f'(t_1) = \frac{a(h_1 - h_0)}{4}. \tag{3.8}$$

Therefore, $S_{max} = \frac{a(h_1 - h_0)}{4}$, which implies that

$$a = \frac{S_{max} \cdot 4}{h_1 - h_0}. \tag{3.9}$$

At the same time, we know that $S_{max} = S_{max_{norm}} \cdot \frac{Y_{range}}{X_{range}}$ by Equation 3.5.

Therefore, we have:

$$a = \frac{S_{max} \cdot 4}{h_1 - h_0} = \frac{S_{max} \cdot 4}{Y_{range} \cdot (h_{1_{norm}} - h_{0_{norm}})}, \tag{3.10}$$

which we can rewrite as

$$a = \frac{4S_{max_{norm}} \cdot \frac{Y_{range}}{X_{range}}}{Y_{range} \cdot (h_{1_{norm}} - h_{0_{norm}})} = \frac{4S_{max_{norm}}}{X_{range} \cdot (\frac{(h_1 - h_0)}{Y_{range}})}. \tag{3.11}$$

Since $\frac{(h_1 - h_0)}{Y_{range}}$ is approximately equal to 1, we can approximate Equation 3.11 with:

$$\frac{4S_{max_{norm}}}{X_{range}} \tag{3.12}$$

Equation 3.12 looks similar to `sicegar's` descaling calculation, off by a factor of four. This factor is something for us to keep in mind as we estimate a using `fitAndCategorize`.

3.2 Optimization: Levenberg-Marquardt Algorithm

The Levenberg-Marquardt Algorithm (LMA) is a method of least-squares approximation which combines the Gauss-Newton algorithm and the gradient descent method [7]. In particular, the LMA is used to solve nonlinear least squares problems which require an iterative optimization process to find a solution. Nonlinear least squares algorithms minimize the sum of the squares of the errors between the estimated model function and our data by repeatedly updating the values of model coefficients until we arrive at a minimum. The LMA is employed by `sicegar` through the `minpack.lm` package to numerically approximate the parameters of best fit for a sigmoidal or double-sigmoidal model. In order to understand the LMA, we first need to define the optimization methods of gradient descent and Gauss-Newton.

3.2.1 Gradient Descent

The gradient descent algorithm updates model coefficients in the direction opposite to the gradient of the objective function. In other words, the "steepest" direction downhill from the gradient.

Given an ordinary-least-squares optimization problem with a set of m points (x_i, y_i) and a parameter vector β, we want to minimize the objective function:

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2$$

where $f(x_i, \beta)$ is the output of our function as predicted by our current model, for example, the sigmoid or double sigmoid function. Since we are working with a nonlinear least squares problem, we can apply the gradient descent algorithm to our objective function to iteratively solve for the parameters β of best fit. We follow the derivation of Gavin (2023) [4]:

First, we compute the gradient of the objective function with respect to β:

$$\nabla_{S(\beta)} = \sum_{i=1}^{m} -2(y_i - f(x_i, \beta)) \frac{\partial(f(x_i, \beta))}{\partial \beta} = -2 \sum_{i=1}^{m} (y_i - f(x_i, \beta)) \frac{\partial(f(x_i, \beta))}{\partial \beta} \tag{3.13}$$

To make things simpler, we can rewrite this gradient in vector form. Our residual vector

$$y - f(x, \beta)$$

has m rows. We define the Jacobian:

$$J = \frac{\partial(f(x, \beta))}{\partial \beta}$$

to be the partial derivative evaluated over all points m and all p parameters in β. Therefore, the Jacobian is an (m x p) dimensional matrix. To write the gradient with respect to β as a (p x 1) vector, we multiple by the transpose of the Jacobian:

$$\nabla_{S(\beta)} = -2J^T[y - f(x, \beta)].$$

Therefore, the coefficient update h_{gd} that moves the coefficients in the direction of steepest descent is:

$$h_{gd} = -\alpha \nabla S(\beta) = -\alpha(-2J^T[y - f(x, \beta)]) = \alpha' J^T[y - f(x, \beta)],$$

where α' is a positive scalar that determines the size of the step in the direction opposite the gradient. This coefficient update will move us in the steepest direction towards the minimum of our objective function [4]. The gradient descent method is robust, and converges well for problems with simple objective functions, but often it can take a long time to converge to the optimal solution.

3.2.2 Gauss-Newton

The Gauss-Newton Method is another minimization algorithm which assumes our objective function is approximately quadratic in coefficients near the optimal solution.

We start with the same objective function:

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2$$

At each iteration, we update our parameter vector to a new estimate $(\beta + \delta)$, where

$$f(x_i, \beta + \delta) \approx f(x_i, \beta) + J_i \delta$$

and J from Section 3.2.1 is equivalent to J_i in vector form. We substitute the approximation into our original objective function:

$$S(\beta + \delta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta) - J_i \delta]^2 \tag{3.14}$$

To minimize the new objective function in Equation 3.14, we take the derivative with respect to δ rather than β:

$$\frac{\partial}{\partial \delta} \sum_{i=1}^{m} [y_i - f(x_i, \beta) - J_i \delta]^2 = 2 \sum_{i=1}^{m} J_i [y_i - f(x_i, \beta) - J_i \delta]$$

To find the minimum of our function, we set this derivative equal to zero and solve for δ, the updated vector of our coefficients:

$$0 = 2\sum_{i=1}^{m} J_i[y_i - f(x_i, \beta) - J_i\delta]$$

$$0 = \sum_{i=1}^{m} J_i[y_i - f(x_i, \beta)] - \sum_{i=1}^{m} J_i^2\delta$$

$$J^T J\delta = J^T(y - f(x, \beta))$$

$$\delta = (J^T J)^{-1} J^T(y - f(x, \beta))$$

Therefore, the coefficient updates to the vector δ that minimize our objective function follow $(J^T J)^{-1} J^T(y - f(x, \beta))$ [4]. The Gauss-Newton method, while less stable than gradient descent, typically converges much faster, especially as the objective function approaches the optimal solution.

3.2.3 Combining Methods: Levenberg-Marquardt Algorithm

The Levenberg-Marquardt Algorithm alternates between the gradient descent and Gauss-Newton coefficient updates depending on how close we are to our solution. Importantly, the algorithm adds a damping parameter λ to the Gauss-Newton minimization function:

$$(J^T J + \lambda I)\delta = J^T(y - f(x, \beta))$$

Small values of λ result in a Gauss-Newton update, while large values of λ result in a gradient descent update. When $\lambda \to \infty$, $(J^T J + \lambda I)^{-1} \to \frac{1}{\lambda} I$, which tends towards 0. Therefore, our optimization is dominated by a scaled identity matrix in the gradient direction. In the LMA, the damping coefficient λ is initialized to be large so that the first few iterative steps are in the steepest direction, mimicking gradient descent. If any iteration happens in a "worse" direction than before, such that $S(\beta+\delta) > S(\beta)$, λ is increased. However, if the solution improves, then λ is lowered and LMA approaches the Gauss-Newton method (which finds our local minimum more more quickly). This allows us to use a faster and more flexible method (Gauss-Newton)

when we are close to our optimal solution, and a slower but stabler method (gradient descent) when we are far from our solution. In Marquardt's update, λ is scaled by the diagonal of the Hessian matrix $J^T J$ for each coefficient [6].

3.3 `sicegar` Parameter Estimation

The primary use of `sicegar` is to fit sigmoidal models to time-intensity data. In our project, we want to use `sicegar` to estimate the onset time of RNA-abundance in *E. coli* cells after being exposed to a stressful stimulus. As long as our data are in this time-intensity form, we can feed it directly into `fitAndCategorize`, where the data are normalized, the LMA is run on our data, and estimated parameters are rescaled and returned in the model output.

3.3.1 Setting Hyperparameters

Hyperparameters are user-set parameters whose values dictate the learning process of an algorithm. `sicegar` allows the user to set multiple hyperparameters for the Levenberg-Marquardt Algorithm before the fitting process begins. The two key hyperparameters we will outline here are:

1. **Bounds** Each parameter is given an upper and lower bound, which mark the maximum and minimum values that parameter can take in any given iteration of our model. In the case of the sigmoid model, we set bounds for h_0, h_1, a, t_1.

2. **Starting Values** Starting values are the initial guess we give the LMA for what we think our parameters should be. The LMA begins the fitting process with the initial guesses for the vector $\beta = (h_0, h_1, a, t_1)$ and updates the coefficient vector from there.

3.3.2 Simulating

Our main project goal is to assess how well `sicegar` estimates sigmoid and double-sigmoid model parameters: specifically, how close are the `sicegar` estimates to the true, underlying parameters of our data.

In order to understand `sicegar`'s performance, we need to run the optimization routine on **simulated** data, which is noisy data generated from

known parameter values [11]. This allows us to run `sicegar` functions, such as `fitAndCategorize`, on data where we have the ability to directly compare the estimated parameters to the true parameter values.

Below, we outline an example of how the simulation process works: this method was crucial in running analyses for this thesis project. All simulations were run in R.

Simulation Example

We start by defining a base sigmoidal function following the form given in Equation 3.15:

$$f_{sim}(x) = \frac{750}{1 + e^{-0.05(x-160)}} \tag{3.15}$$

Thus, the true parameters for our model are:

1. $h_0 = 0$
2. $h_1 = 750$
3. $a = 0.05$
4. $t_1 = 160$

The above parameters were chosen arbitrarily, but meet all parameter restrictions for a sigmoid model. Namely, $h_0 < h_1$, $t_1 > 0$, and $a > 0$.

Next, we calculate the values of $f_{sim}(x)$ at 16 time points within the range $x \in [0, 300]$. The range of x was also arbitrarily chosen, but closely resembles real-world RNA-seq data. We calculate $f_{sim}(x)$ five times per x value, which mimics taking multiple replicates at each time point. This process results in a dataset resembling Table 3.1:

Sample	x	$f(x)$
1	0	0.25
2	0	0.25
⋮	⋮	⋮
80	300	302.74

Table 3.1: Example dataset before adding noise.

To simulate noisy data, we generate a new column of our dataset, where each value is drawn from a normal distribution centered at $f(x)$, with a variance of 100:

$$y \sim N(f(x), 100)$$

For instance, consider the first row of the dataset. We generate a new y value by randomly sampling from the distribution $y \sim N(0.25, 100)$. The first row of our new dataset is shown in Table 3.2:

Sample	x	$f(x)$	y
1	0	0.25	199.7

Table 3.2: Example of a noisy data point.

This process is repeated for all 80 rows, yielding a noisy dataset that simulates real-world experimental variation. Figure 3.1 illustrates the process of simulating sigmoidal data and using the data to compare `sicegar` estimates to the known generative parameters:

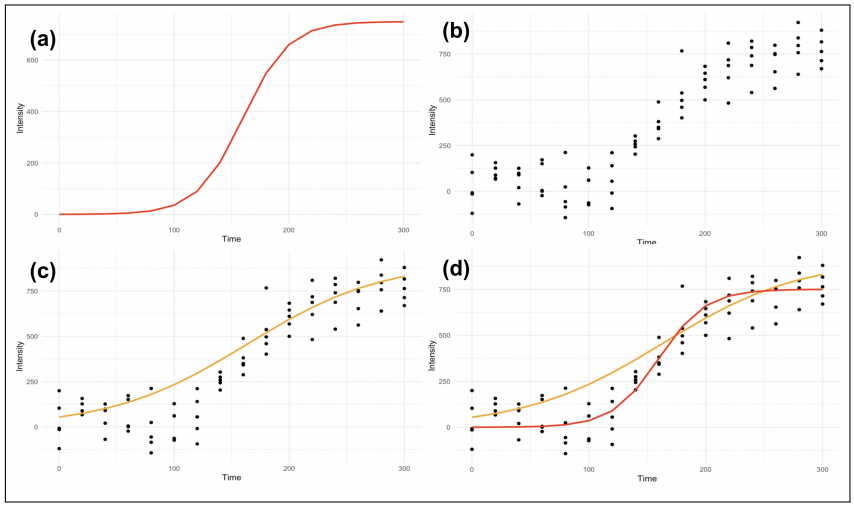


Figure 3.1: Using a simulated dataset to compare `sicegar` estimated parameters (orange line) to the underlying model parameters (red line). Panel (a) shows an initial function, where the parameters are set as described in Equation 3.15. Panel (b) shows the noisy data generated from the function in (a). Panel (c) shows `sicegar`'s estimated sigmoid model. Panel (d) shows a direct comparison between `sicegar`'s model and the true, underlying model of the data.

By introducing controlled randomness into the data while knowing the underlying generative model, simulation provides a valuable way to assess the accuracy of `sicegar`. Using simulated data, we were able to conduct a preliminary analysis of `sicegar` estimates to determine what biases the package introduced into parameter estimation.

First Round of Simulations

In order to obtain a baseline understanding of how well `sicegar` performs when estimating sigmoid models, we ran a series of simulations to determine whether there were any trends in the estimated parameters.

In Prof. Hardin and Fede's earlier work, they determined that the onset time of the single sigmoid function, t_1, was repeatedly **underestimated**. Figure 3.2 illustrates this trend:

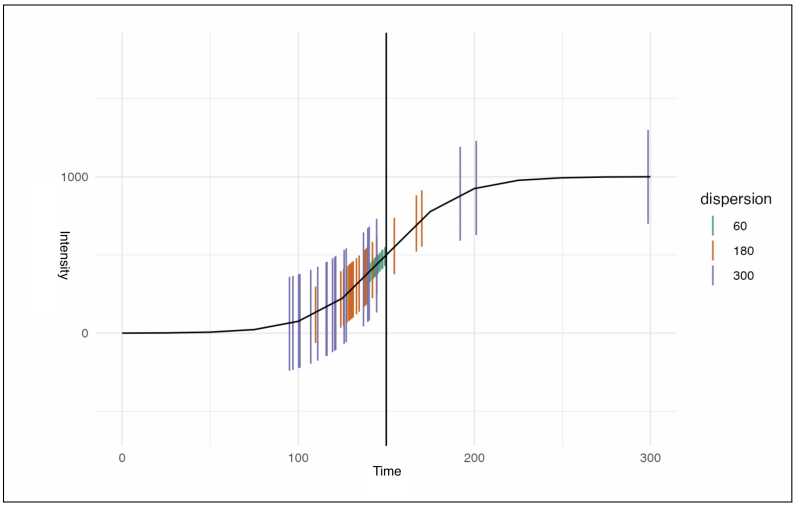


Figure 3.2: (Hardin & Domecq Lacroze, 2024) Data were simulated at three dispersion levels (60, 180, 200) from an underlying model with parameters $h_0 = 0, h_1 = 750, a = 0.05, t_1 = 160$. Data was then run through `sicegar` and the estimated onset time ($\hat{t_1}$) was extracted and plotted against the true value of onset time. Onset time was typically underestimated across all dispersion levels.

Our next step was to determine why these estimation errors/bias occur, and whether or not we can manually improve `sicegar` estimates.

3.3.3 The h_0 Problem

Although `sigmoidalFitFunction` returns parameter estimations for h_1, t_1, and a, it does not include an estimation for h_0. This is because the `sicegar` optimization routine assumes h_0 to be zero, and thus forces h_0 to be zero in all estimations. There are a number of drawbacks to this assumption. Namely, forcing $h_0 = 0$ prevents us from accurately modeling data in `sicegar` where the true value of $h_0 \neq 0$. Forcing $h_0 = 0$ can also influence the estimation of our other key parameters. Figure 3.3 shows the results of comparing estimations of parameters a, h_1, t_1 when forcing $h_0 = 0$ in the model approximation, versus letting h_0 also be estimated as a parameter.

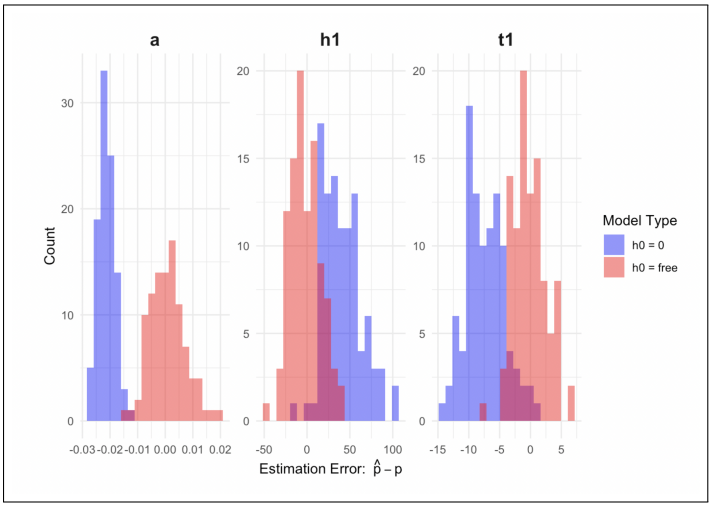


Figure 3.3: We compared bias in parameter estimation using the LMA when forcing $h_0 = 0$ versus allowing the algorithm to fit h_0. We simulated 100 datasets from the underlying sigmoid model $f(x) = 100 + \frac{750-100}{1+e^{-0.05(x-160)}}$ and estimated their parameters using the Levenberg-Marquardt algorithm in two ways: (1) assuming $h_0 = 0$, and (2) estimating h_0 as its own parameter. The estimation error for each parameter $(\hat{p} - p)$ across all 100 models is plotted along the x-axis for both approximation methods.

The estimation errors for the model with a fixed h_0 are not centered at 0 as compared to the model with an estimated h_0. On average, the biases in parameter estimation were greater for those models whose h_0 was fixed at 0. In conclusion, forcing $h_0 \to 0$ when estimating parameters using the LMA may cause unintentional biases in the estimation results of parameters h_1, a, t_1. Therefore, moving forward, we modified `sicegar` to include h_0 in the model estimation process to eliminate any of the potential biases discussed above. To do this, we altered the underlying sigmoidal function formula (stored as `sigmoidalFitFormula` in `sicegar`) to follow the sigmoid function described in Equation 2.2.

Chapter 4

Simulation Results

4.1 Getting `sicegar` to work

In order to better understand the biases in `sicegar`'s estimation, our first step was to determine whether or not we could get `sicegar` to estimate t_1 within a reasonable range. We were particularly interested in whether the LMA introduced bias into our parameter estimations. No nonlinear-least squares algorithm will be able to perfectly predict our parameters 100 percent of the time. However, we can run sigmoidal data directly through the algorithm itself to see whether or not it produces any underlying biases and to measure the variability of the estimates.

To get a better sense of how well the LMA performs, we stripped away the majority of `sicegar`'s nested functions and simply ran our time-intensity data through `nlsLM`, without normalizing our data beforehand. With the addition of h_0 into our sigmoidal model, this process returned the following estimated parameters: `h0, maximum, midPoint, slopeParam`. We also set our own hyperparameters (bounds & starting values, see Section 3.3.1) rather than relying on the ones provided by `sicegar`. The purpose of these simulations was to determine whether the LMA itself had any biases and whether or not the package could work properly with some user-control.

We started with a very specific example:

Estimating Parameters of a Single Sigmoid

Our goal was to estimate the parameters of Equation 2.2, where data were generated from a single-sigmoid model with

$$h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 160.$$

We started by creating four groups of simulated datasets. In each grouping, we held three parameters constant, and then iterated through sixty reasonable values for the fourth parameter. For example, in Group 1, we simulated sixty datasets where

$$h_1 = 750, \quad a = 0.05, \quad t_1 = 160,$$

and

$$h_0 \in [0, 300].$$

This process allowed us to determine whether certain parameter combinations broke the optimization routine (i.e., do really large h_0 values make it more difficult to estimate t_1?), as well as to assess the overall accuracy of `sicegar` estimates. Figure 4.1 shows the results of these simulations.

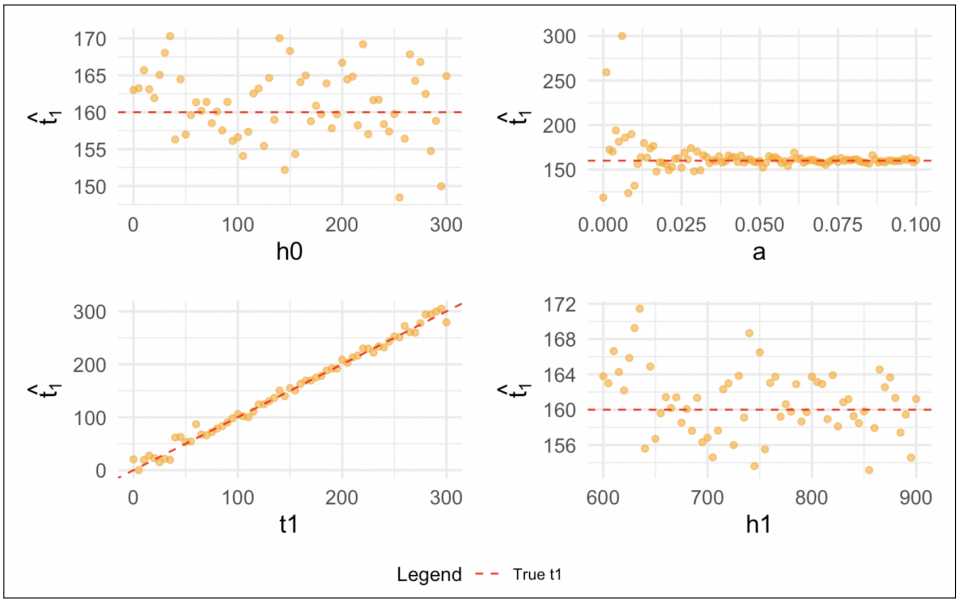


Figure 4.1: These plots show how estimates of t_1 from a single-sigmoid model vary when one parameter is systematically changed. For each plot, we generated 60 datasets while holding three parameters constant at $h_0 = 0$, $h_1 = 750$, $a = 0.05$, and $t_1 = 160$, and varying the fourth parameter across the x-axis. The orange points represent the estimated values of t_1 from each individual dataset, and the red dashed line marks the true value of t_1.

Overall, `sicegar` was able to estimate t_1 within a reasonable range. Table 4.1 shows the proportion of datasets whose value of t_1 was correctly estimated across the four simulation groups, where we define a correct estimation as being with 10-points of t_1 in either direction. In each case, `sicegar` correctly estimated the value of t_1 well over 75 percent of the time.

Iterated Parameter	t_1 **Accuracy**
h_0	0.93
h_1	0.80
a	0.80
t_1	0.98

Table 4.1: Preliminary accuracy results for t_1 across different iterated parameters.

However, when running these simulations, we paid careful attention to set the hyperparameters as "perfectly" as possible. First, we set our starting values at

$$h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 160,$$

so that the starting guess for each parameter would be equivalent to the true value of that parameter in at least three of the four cases. Secondly, we set our bounds as

$$h_0 = (-100, 1000), \quad h_1 = (0, 1000), \quad a = (0, 1), \quad t_1 = (0, 500).$$

These bound settings ensured that the true value of each parameter was between its upper and lower bound, but also made the bounds restrictive enough so that the algorithm would be able to converge. Therefore, our next step was to determine whether we could get `sicegar` to work in the general case, where we did not have an underlying knowledge of the true structure of our data.

4.2 Setting Hyperparameters on Normalized Data

As mentioned in Section 3.1.1, `sicegar` normalizes time-course data on both the x and y axes such that all values of $x, y \in [0, 1]$. This process allows the package to set "universal" bounds and starting values so that the user does not need to determine appropriate hyperparameters before estimating. However, normalizing our data makes parameter estimations particularly sensitive to hyperparameter settings. We were interested in how upper bound and start values would influence the LMA's ability to estimate the correct sigmoidal parameters. We will outline the influence of hyperparameters on estimation accuracy in the next two sections.

Before adjusting the default hyperparameters in `sicegar`, we explored the optimization behavior of our single sigmoid function. Specifically, we plotted the sum of squares errors (SSE) across various parameter combinations. SSE is a statistical metric used to evaluate how accurately a posited model with parameter vector β fits some data. Specifically, SSE measures the total sum of squared differences between n observed and predicted data points:

$$SSE = \sum_{i=1}^{n} (y_i - f(x_i, \beta))^2 \tag{4.1}$$

To better understand the optimization behavior of the single sigmoid, we plotted SSE for four simulated datasets (simulated from the same parameters $h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 160$, with 5 replicates for 20 time points between 0 and 300) across 8470 combinations of the parameter vector β using the following parameter ranges:

1. $h_0 \in [0, 1]$
2. $h_1 \in [600, 900]$
3. $a \in [0.01, 0.1]$
4. $t_1 \in [100, 200]$

These ranges were centered around the true, generative parameter value for each parameter. For each combination of h_0, h_1, a, t_1, we calculated $\sum_{i=1}^{80} (y_i - f(x_i, (h_0, h_1, a, t_1)))^2$. Figure 4.2 shows the results of these simulations for one of the four datasets:

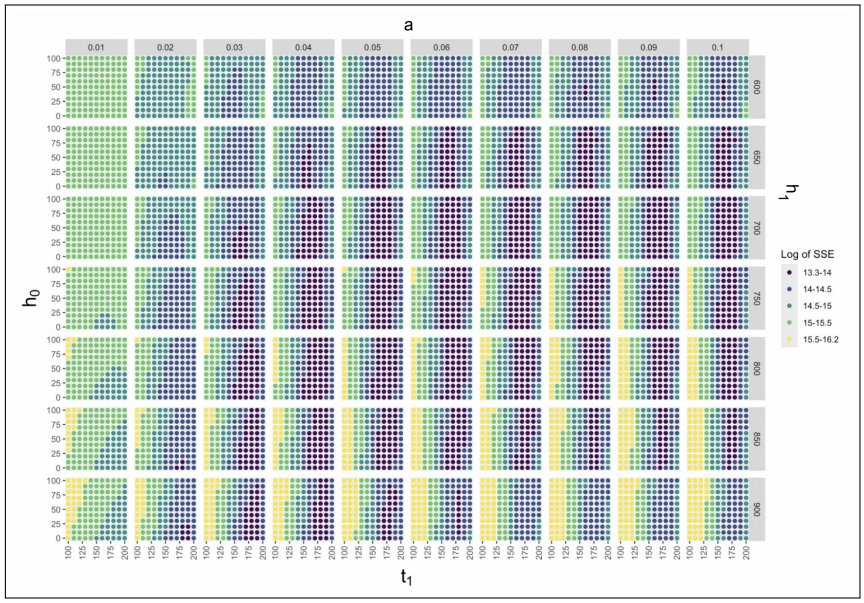


Figure 4.2: Each point represents a unique parameter vector $\beta = (h_0, h_1, a, t_1)$, with values for t_1 along the x-axis, values for h_0 along the y-axis, values for a along the upper grid, and values for h_1 along the grid on the right. The color of each point represents the SSE, or the total sum of squared differences between the true and estimated data points across all 80 data points. Darker colors correspond to a lower SSE, and lighter colors correspond to a lower SSE. SSE was plotted on a log scale.

Local optimization behavior is marked by dark regions (low SSE) of a heatmap surrounded by lighter regions, indicating a potential minimum in that neighborhood of parameters, but not across the entire grid. In general, this heatmap does not show any clear signs of local minima, since a majority of the dark regions are concentrated towards the edge of each sub grid of parameters. However, we can see for values $a = 0.03, 0.04$ and $h_1 = 700, 750$ that there are small clusters of darker regions surrounded by slightly lighter regions, which could be a potential concern when fitting models to our data. Additionally, for low values of a and h_1, our plot displays "flat" regions where there is a relatively constant SSE level. These regions could potentially cause the optimizer to stall or entirely miss the global minimum of our objective function.

For all four simulated datasets, we compared the parameter vector β associated with the lowest SSE to the true parameter values, and determined how

far apart the lowest SSE was to the "truth" SSE. Note that for all datasets, the true parameter values were $h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 160.$

$\beta = (h_0, h_1, a, t_1)$	**Log SSE**	**True Log SSE**
$(40, 750, 0.06, 170)$	13.33	13.42
$(0, 700, 0.06, 160)$	13.72	13.73
$(0, 750, 0.05, 160)$	13.44	13.44
$(20, 750, 0.06, 170)$	13.60	13.63

Table 4.2: Comparing lowest SSE to SSE associated with true parameter values across four simulated datasets.

As shown in Table 4.2, the parameter vector associated with the lowest SSE was typically close to the true parameter vector for all four datasets. Additionally, the lowest log SSE value was never more than 0.1 points away from the SSE value associated with the true parameters. Therefore, based on this small simulation, we were not too concerned with getting stuck in local optima when minimizing our objective function across this parameter grid. Although it may be beneficial to use multiple starting values to estimate model parameters, the the Levenberg–Marquardt algorithm (LMA) relies on a single starting point for parameter estimation. Thus, we first sought to explore how different hyperparameters influence the outcomes of LMA estimation.

4.3 Manipulating Upper Bounds

In order to determine whether the upper bound hyperparameter had an influence on LMA-parameter estimation, we ran simulated data through various upper-bound values and compared the estimated parameters to the generative parameters of our model.

We started by simulating 20 datasets using the same generative parameters:

$$h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 160.$$

We then set a reasonable range for the upper bound of each parameter. The lower end of the range was the default starting value in `sicegar`, and the upper end of the range was the default upper bound in `sicegar`:

Parameter	**Range**
h_0	$[0, 2]$
h_1	$[1, 1.5]$
a	$[1, 180]$
t_1	$[0.33, 1.15]$

Table 4.3: Upper Bound Values for LMA Estimation

We generated 3750 unique upper bound combinations by iterating through each range, and ran our 20 simulated datasets through the LMA using each individual combination. We extracted $\nabla p = |p - \hat{p}|$ for all four parameters, and averaged that value across 20 datasets. Estimating the sigmoidal model for data under numerous upper bound combinations gave us a sense of whether a particular upper bound combination yielded a more or less accurate estimation result. We wanted to directly compare estimations across all four parameters, since our intuition was that parameter estimations were intertwined. Figure 4.3 displays the results of these simulations:

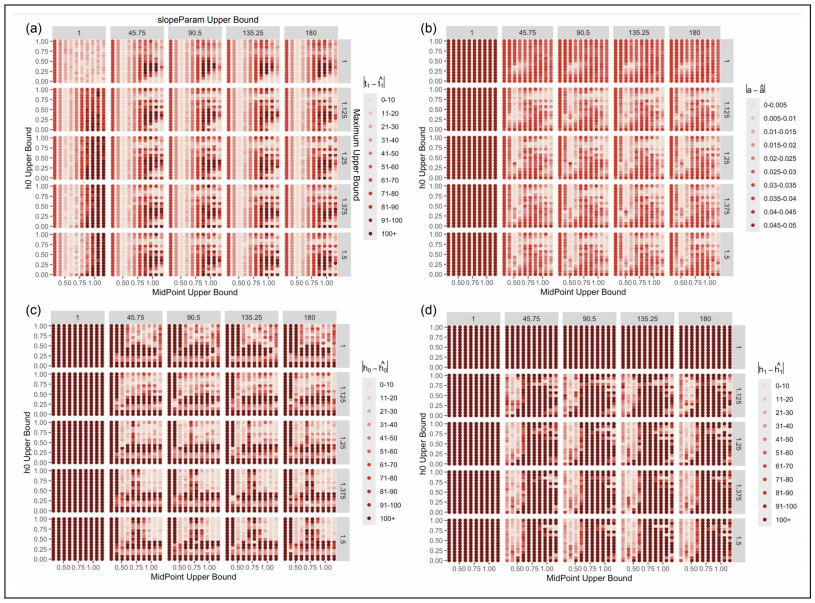


Figure 4.3: Each point represents a unique upper bound combination, with the x-axis iterating through t_1 upper bound values, the y-axis iterating through h_0 upper bound values, the grid across the top iterating through a upper bound values, and the grid along the right side iterating through h_1 upper bound values. The color of each point displays how close we were to correctly estimating one parameter: (a) t_1, (b) a, (c) h_0, and (d) h_1. Lighter colors indicate a more accurate fit, and darker colors indicate a less accurate fit.

Looking specifically at ∇t_1, in Figure 4.3 (Plot (a)) we observe that both the slopeParam (a) Upper Bound and Maximum (h_1) Upper Bound do not have a large impact on t_1 estimation past a certain threshold. Once the slopeParam (a) Upper Bound is larger than 1, each individual square in the grid looks fairly alike. Similarly, once the Maximum (h_1) Upper Bound jumps from 1 to 1.125, we see a similar structure to each individual square. However, the combination of MidPoint (t_1) Upper Bound and h_0 Upper Bound proved to be highly influential on ∇t_1. Specifically, we can see that when the MidPoint (t_1) Upper Bound is just below or just above 0.5, we have worse fit (darker colored points). However, as both the Midpoint (t_1) and h_0 upper bounds increase (top right corner of each individual square), we can see that our fit of t_1 improves, which indicates that larger upper bound values allow for more accurate fitting. We also need to set the slopeParam (a) Upper

Bound fairly liberally (much greater than 1) in order to get the improved fit when increasing the MidPoint (t_1) and h_0 Upper Bounds.

We can simplify Plot (a) to examine whether or not t_1 was overestimated, underestimated, or approximately correct at each combination. Figure 4.4 shows this simplified version of Plot (a) of Figure 4.3, where we defined overestimating t_1 as a value > 170, underestimating t_1 as a value < 150 and correctly estimating t_1 as a value $\in [150, 170]$ (keeping in mind that the true value of t_1 is 160).

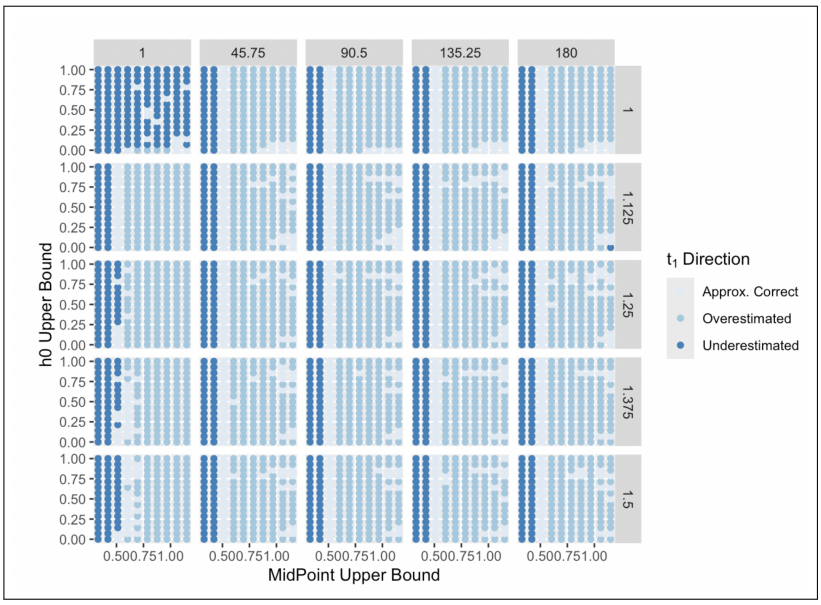


Figure 4.4: Each point represents a unique upper bound combination, and the color indicates whether t_1 was overestimated, underestimated, or correctly estimated.

As expected, t_1 is underestimated when the MidPoint (t_1) Upper Bound is lower than the true value of t_1 in normalized space. t_1 is estimated accurately when our MidPoint (t_1) Upper Bound is set to the approximate true value of t_1, and for some combinations of larger MidPoint (t_1) and h_0 Upper Bounds, and is overestimated in most cases when the MidPoint (t_1) Upper Bound is greater than the approximate true value of t_1.

As mentioned, we were interested in how intertwined the estimations across different parameters were: for example, if t_1 was correctly estimated, did that mean that the estimations for a, h_0, h_1 were also fairly accurate?

Most notably, the fit of all parameters deterioriates when the upper bound for a is set to be too low (i.e., when the upper bound of a is set to 1). Although it is less apparent, the fit across all parameters also deteriorates when the Maximum (h_1) Upper Bound is equal to 1. This deterioration is most noticeable when estimating h_1 and a, but the proportion of closely estimated t_1 and h_0 values also decreases when the Maximum (h_1) Upper Bound is too small.

Interestingly, when MidPoint (t_1) Upper Bound is approximately equal to 0.5, we get an accurate fit of t_1, shown by the pale line in each square at $x = 0.5$. We hypothesized that this increased accuracy may be because the true value of $t_1 = 160$ is approximately 0.5 in normalized space (since $\frac{160}{300} = 0.53$), and therefore capping the upper bound at 0.5 forces t_1 to be correct.

Given our finding that t_1 was most accurately estimated when we set our upper bound to be the approximate true value of t_1, we tested two additional t_1 values and ran the same simulation procedure to see if this finding would hold.

4.3.1 Setting t_1 to 220

We first generated our datasets setting $t_1 = 220$, so we had the following generative parameters:

$$h_0 = 0, \quad h_1 = 750, \quad a = 0.05, \quad t_1 = 220.$$

Note that the normalized value of t_1 is 0.73. We then repeated the same procedure to extract the average of $\nabla p = |p - \hat{p}|$ across 20 datasets, and plot this value for all 3750 parameter combinations. Figure 4.6 shows the results of this simulation.

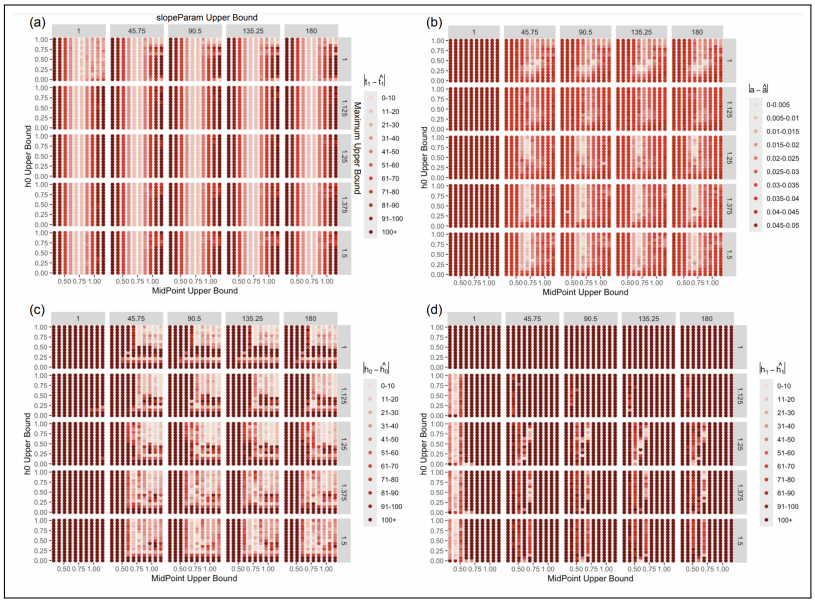


Figure 4.5: Each point represents a unique upper bound combination. All parameter values are the same except for t_1, which now equals 220, which is 0.73 in the normalized space.

In Plot (a), we see that the pale line has shifted to approximately $x = 0.75$, which is close to the value of $\frac{220}{300} = 0.733$. The shift in t_1 estimation accuracy supports our hypothesis that setting the MidPoint (t_1) Upper Bound approximately equal to t_1 forces the estimation to be more accurate. However, we also noticed that for this t_1 value, we no longer get an improved fit in the upper right corner of each square (where we maximize both MidPoint and h_0 Upper Bounds). The fit of all other parameters (Plots (b) - (d)) appears fairly similar, with the exception of h_1, which is consistently estimated incorrectly by around 100 points.

4.3.2 Setting t_1 to 90

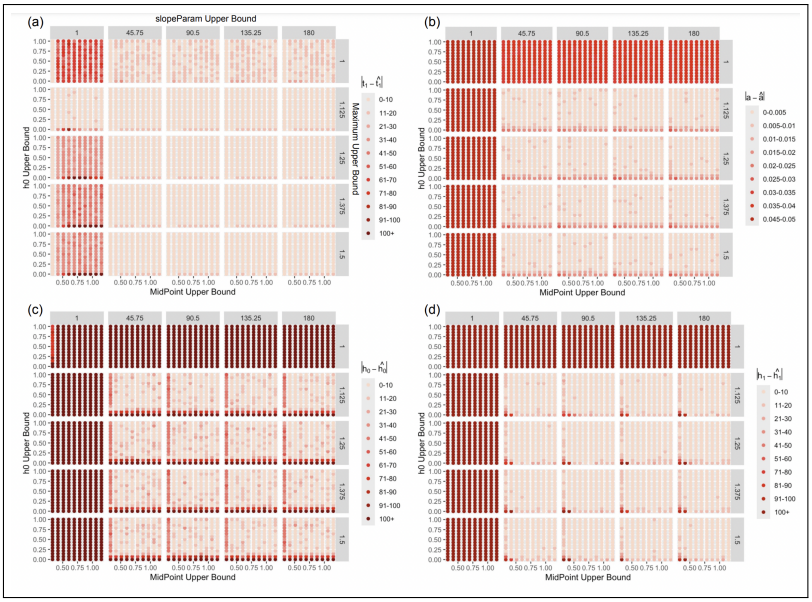


Figure 4.6: Each point represents a unique upper bound combination. All parameter values are the same except for t_1, which now equals 90.

While setting $t_1 = 220$ seemed to support our intuition, setting $t_1 = 90$ did not yield the expected results. Rather, the estimation of all parameters improved across the majority of upper bound combinations, with the exceptions of setting slopeParam (a) Upper Bound to 1 and the Maximum (h_1) Upper Bound to 1. Additionally, for smaller upper bounds of h_0, we can observe a worse fit. This improved estimation may have something to do with the structure of the sigmoid model at this particular parameter combination, which we did not investigate in this project.

4.4 Manipulating Starting Values

The second set of hyperparameters we were interested in was the set of starting values, which are the first guess the LMA makes when estimating the parameters of our sigmoid model. In `sicegar`, the default starting values are:

$$h_0 = 0, \quad h_1 = 1, \quad a = 1, \quad t_1 = 0.33.$$

Whereas h_1 and t_1 both seem like reasonable starting estimates for their respective parameters in normalized space, the starting value $a = 1$ stood out as being extremely low. Especially given the poor fit all of four parameters when setting the slopeParam (a) Upper Bound to 1, we hypothesized that a larger starting value for a might yield more accurate results.

Therefore, we ran the same initial simulations (as shown in Figure 4.3) but set a new starting value for a :

$$h_0 = 0, \quad h_1 = 1, \quad \mathbf{a = 15,} \quad t_1 = 0.33.$$

We set the starting value $a = 15$ since we know that our actual slope parameter $a = 0.05$ is approximately equal to 15 in normalized space. Figure 4.7 shows the results of this simulation.

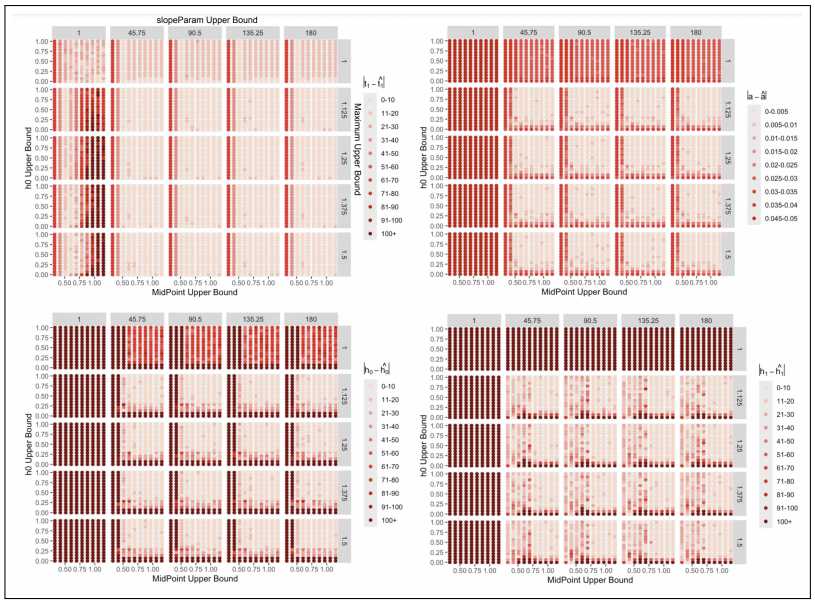


Figure 4.7: Each point represents a unique upper bound combination, now with the starting value for $a = 15$.

The estimation of all parameters improved drastically after altering the starting value of a from 1 to 15. When both the slopeParam (a) and Maximum (h_1) Upper Bounds were set larger than 1, the combination of a higher midPoint and h_0 upper bound consistently lent itself to accurate estimations of all four parameters. The drastic improvement in estimation accuracy indicates that starting values are highly influential on the LMA's ability to

accurately estimate onset time, and requires further exploration. Since we were able to set the starting value $a = 15$ from our omniscient perspective on the data, it may be fruitful to manually set LMA starting values based on approximations from the data. We will discuss potential avenues to address the omniscience quandry in more detail in the next chapter.

Chapter 5

Conclusion and Future Directions

RNA-sequencing (RNA-seq) data provides a snapshot of the RNA abundance associated with a particular gene in a biological sample. Adams et. al (2023) collected RNA-seq data to determine how *E. coli* cells alter their gene expression patterns in response to various environmental stressors and were particularly concerned with comparing the onset time of RNA abundance across stressors and gene types. In order to extract onset time, Adams et. al (2023) used `sicegar`, an R package that fits sigmoidal models to time course data [5]. In this thesis, we investigated how reliably `sicegar` was able to estimate the onset time of RNA abundance using data simulated from an underlying sigmoid model, where we knew the true parameter values beforehand. We identified potential roadblocks in the parameter identifiability of sigmoid models, as well as biases in the Levenberg-Marquardt Algorithm.

In conclusion, we found that both the upper bounds and starting values of the Levenberg-Marquardt Algorithm highly influence our ability to estimate the parameters of a single sigmoid model. When we set our hyperparameters directly from the data, such as setting the starting values to the true parameter values, and ensuring our bounds include the true parameter values, we can get `sicegar` to work with high accuracy across a variety of parameter combinations. However, real-life applications of `sicegar` require working with data for which we do not already know the underlying structure. Therefore, our goal was to find a way to expand our ability to accurately estimate onset time without having an omniscient perspective.

Therefore, per `sicegar`'s embedded structure, we worked with data nor-

malized to the $[0, 1]$ scale on both the x and y axes. We found that when working with normalized data, our estimations became extremely sensitive to hyperparameters. When we set the upper bounds of all four parameters liberally, we were able to accurately estimate t_1 a majority of the time. However, small adjustments in upper bound values led to large discrepancies in parameter estimations. For example, when h_1 is set to 1 rather than 1.125, our estimation of a, h_1, h_0 worsens, despite the fact that the true value of $h_1 < 1$ in normalized space. Although there is no clear rule for setting upper bound values that guarantee correct estimations 100% of the time, we found that using a combination of high MidPoint (t_1) and h_0 upper bounds generally lead to accurate estimates—provided that the slopeParam (a) and Maximum (h_1) upper bounds are also set above certain threshold values.

We also found that the estimation process worked particularly well when we set the starting value of our slope parameter a to be almost equal to the true value of a, which was around 15. Therefore, we believe there are a number of future directions this project can take to explore the influence of starting values more closely. First, we believe it would be fruitful to alter the LMA, or use another non-linear least squares estimation technique, to look over a large space of starting values before beginning the optimization routine. Additionally, we propose working with un-normalized data, but altering the default settings of the start values to be drawn directly from the data. For example, we could approximate the start values from a dataset as follows:

1. h_1 : mean of the three highest y values
2. h_0 : mean of the three lowest y values
3. t_1 : mean/median of x values, potentially where $f(x)$ starts increasing rapidly
4. a : slope from the local regression on data points right above and below t_1

Setting starting values from the data is one potential solution to finding more accurate starting values before the optimization procedure begins, and we recommend testing this procedure out as a next step to improving `sicegar` estimates. Once we have a better understanding of parameter estimation in the sigmoid model, we believe it would be fruitful to expand our analysis

to the double sigmoid model, where the difficulty in parameter estimation increases due to the distinction between t_1 and onset time. We hope that this thesis provides a helpful framework to continue investigating `sicegar` biases, with the goal of providing tangible suggestions to `sicegar` users.

Bibliography

[1] *Escherichia coli - an overview*, ScienceDirect Topics.

[2] Josephine Adams, Johnson Hoang, Emily Petroni, Ethan Ashby, Johanna Hardin, and Daniel M. Stoebel, *The timing of transcription of rpos-dependent genes varies across multiple stresses in escherichia coli k-12*, mSystems **8** (2023), no. 5, e00663–23.

[3] Ramos et al., *Escherichia coli as commensal and pathogenic bacteria among food-producing animals: Health implications of extended spectrum -lactamase (esbl) production*, Animals (Basel) (2020 Nov 29).

[4] Henri P. Gavin, *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems*, Duke University Department of Civil and Environmental Engineering (2019), 1–23.

[5] M. Umut Caglar, Ashley I. Teufel, and Claus O. White, *Sicegar: R package for sigmoidal and double-sigmoidal curve fitting*, PeerJ (2018).

[6] Donald W. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, Journal of the Society for Industrial and Applied Mathematics **11** (1963), no. 2, 431–441.

[7] Jorge J. Moré, *The levenberg-marquardt algorithm: Implementation and theory*, Numerical Analysis (Berlin, Heidelberg) (G. A. Watson, ed.), Springer Berlin Heidelberg, 1978, pp. 105–116.

[8] Patricia Foster, *Stress responses and genetic variation in bacteria - PubMed*, Mutation Research (2005), no. 569, 3–11.

[9] R. R. Burgess, *Sigma Factors*, Brenner's Encyclopedia of Genetics (2013), 432–434.

[10] RJ Mackenzie, *RNA-Seq: Basics, Applications and Protocol*, Technology Networks (2024).

[11] Tim P. Morris, Ian R. White, and Michael J. Crowther, *Using simulation studies to evaluate statistical methods*, Stat Med **38** (2019), no. 11, 2074–2102.

[12] Timur V. Elzhov, Katherine M. Mullen, Andrej-Nikolai Spiess, and Ben Bolker, *minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK*, CRAN (2023).

[13] Wolfram, *Logistic Equation*, Wolfram Mathworld.