Topological Data Analysis for Parameter Estimation of Noisy Biexponential Decay

Tyler Hecht

August 9, 2022

1 Introduction

A problem in Magnetic Resonance Imaging (MRI) is the estimation of parameters for a biexponential model with Gaussian noise corrupting the signal. Our biexponential model is defined as follows:

$$S(TE; d_1, d_2, T_{21}, T_{22}) = d_1 e^{-TE/T_{21}} + d_2 e^{-TE/T_{22}} + \epsilon$$

where

$$d_1(TI) = c_1(1 - 2e^{-TI/T_{11}})$$
$$d_2(TI) = c_2(1 - 2e^{-TI/T_{12}})$$

and

- TE is conventional time
- $T11$, $T12$, $T21$, $T22$ are parameters to be estimated
- c_1, c_2 are dimensionless coefficients
- TI is a parameter we can adjust in the MRI
- ϵ is Gaussian noise

Notice that for $TI = T_{11}\ln(2)$ or $T_{12}\ln(2)$, the model becomes monoexponential. We call these values of TI **null points**, and we seek to estimate them using **topological data analysis (TDA)**, as explained below. If we can estimate these null points, we will be able to estimate T_{11} and T_{12}. Also note the concept of **signal-to-noise ratio (SNR)**, which will play an important role later in this report. We define the signal as $c_1 + c_2$ and noise as the standard deviation σ of the Gaussian noise ϵ.

Default values of $c_1 = 0.5$, $c_2 = 0.5$, $T_{11} = 600$, $T_{12} = 1200$, $T_{21} = 45$, $T_{22} = 100$ are used for the entirety of this report, but further research may involve changing these values. However, note that this means that the null points occur at approximately $TI =$ **416**, **832**. These values are referred to throughout the report, so keep them in mind.

This research is the continuation of Ryan's work, and his code for the Metropolis-Hastings algorithm is used for data generation. The code was originally part of a larger Jupyter notebook, but I extracted the relevant parts, made it a function of parameters such as TI and SNR, and made a few modifications including adding a burn-in period and allowing for the initial guess to be flipped. An explanation of this algorithm will be discussed in the following section.

The Metropolis Algorithm code and code for the other parts of this project can be found on my GitHub (github.com/Tyler-Hecht/GEMS-TDA). All code is written in Python, and you are welcome to use and modify it for further research.

2 Methods

2.1 Metropolis Algorithm

Put simply, the **Metropolis Algorithm** takes the true values of $c_1, c_2, T_{11}, T_{12}, T_{21}, T_{22}$ and generates estimates of d_1, d_2, T_{21}, T_{22}. It requires specifying the values of TI, SNR, and the number of **iterations**, or steps, m. The algorithm will give m estimates of the parameters (lists of length 4), with each list representing a point in 4D space. Indeed, it is important to get in "4D space mindset" to better understand the methods. These points form a **point cloud** that is essentially a probability density function (PDF) of the model.

Although the inner mechanics of the algorithm are not the focus of this research, it is still valuable to understand it. An initial guess is put in, and for every step, the algorithm takes a random walk in 4D space to another point. The point is randomly accepted or rejected, with the probability of acceptance being higher if the point is closer to the true distribution, as specified by the parameters. If accepted, the new point is added to the estimates; if rejected, the old point is added again. The result is a PDF roughly representing the the PDF of the true distribution. There is a burn-in period consisting of an extra $m/10$ iterations that is used to allow the random walk to move toward a region of higher density in case it starts in a region of low density.

For our methods, half the iterations are done with an initial guess of the true parameters and half are done with an initial guess of the true parameters but with d_1 swapped with d_2 and T_{21} swapped with T_{22}. The result is two "clusters" of point clouds. This phenomenon is a key part of this research and will be elaborated on shortly.

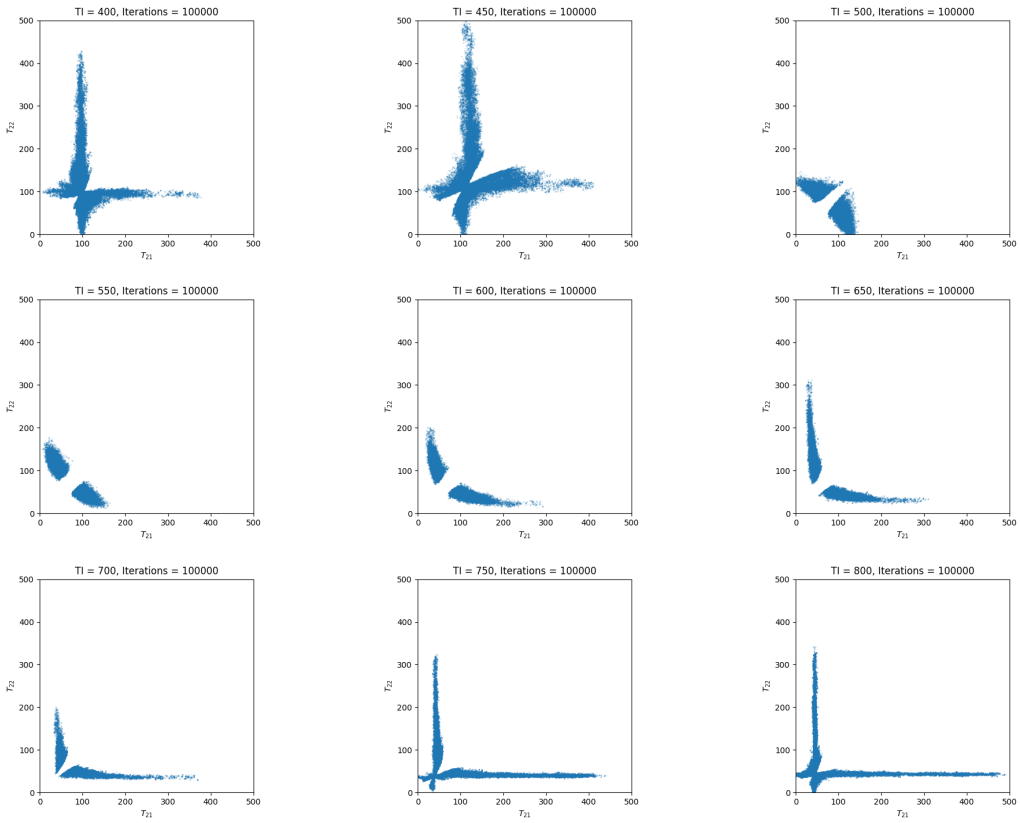

Figure 1: T_{21} vs T_{22} for TIs ranging from 400 to 800 for SNR = 1,000 and m = 100,000. These are replications of Ryan's results. The regions of higher density represent values of the parameters that fit the data well.

Unfortunately, we can't visualize the 4D point clouds, but a 2D point cloud of T_{21} and T_{22} shows the key features (Figure 1). Observe the changes in the point cloud across values of TI from 400 to 800

for m = 100,000 and SNR = 1,000 (we will continue to use SNR = 1,000 for future runs of Metropolis until specified otherwise; SNR = 100 was initially tried, but the result was just a seemingly unorganized scattering of points). The point cloud starts off as one "blob" of points but as TI is increased from 450 to 500, it separates into two. These blobs, or clusters, are known as **connected components** in topology. The connected components separate as TI increases but then start coming back together to form one connected component again between 700 and 750. Interestingly, this action happens between the null points (416 and 832).

2.2 Topological Data Analysis

How can we determine quantitatively how many connected components there are and the distance between the components? We use **persistent homology**, a technique from TDA, to answer these questions. There is much that persistent homology can do, including detecting holes and voids in the point clouds. This research however, uses it for a very simple purpose: measuring the distance between the connected components. As such, only the details of persistent homology relevant to these methods will be discussed. The Plotting file in my GitHub repository plots persistence barcodes and diagrams, which are useful for understanding persistent homology results.

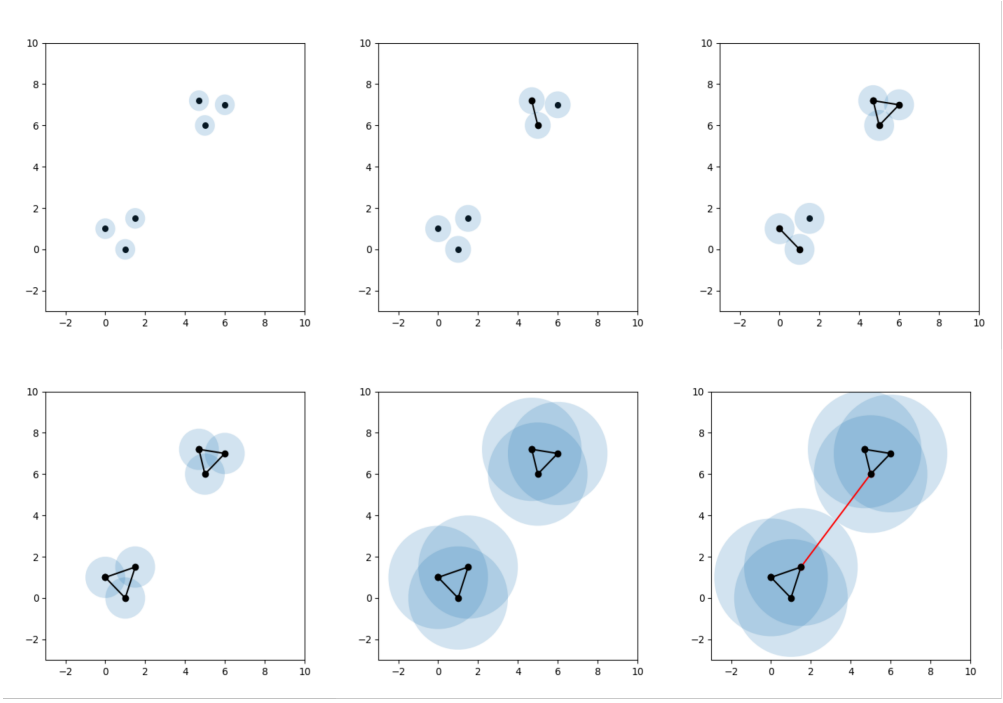

Figure 2: Plots of a 2D point cloud of 6 points with circles of increasing radius. The red line on the bottom right is our measure of the distance between the components.

To illustrate persistent homology, consider a 2D point cloud of just 6 points (Figure 2). There are clearly two clusters of point, but we wish to measure the distance between them. We draw a circle of radius r around each point. As we increase r, some circles intersect. When they do, we connect the points and say that they form one connected component. Each point is its own connected component consisting of one point, so the top left figure has 6 connected components while the top center figure has 5. Keep increasing r and more circles will connect. The top right figure has 3 connected components while the bottom left figure has just 2. We now must increase r quite a bit for there to be another connection, which indicates that the clusters are quite separated. The bottom right figure has the minimum radius such that there is only 1 connected component. Indeed, this value of r such that the number of connected components goes from 2 to 1 is our takeaway from persistent homology. The red line, which has length $2r$, is our measure of the distance between the components. We refer to this distance as the **critical**

radius, and it is very important for our research. We expect a high critical radius when the components are well-separated, but when there is only one connected component, the radius will be quite small and only represent the stochasticity of the Metropolis algorithm.

Technically, the distance between the clusters is not really the radius but the diameter of the circles. However, the calculation of it is done with the Python package **Ripser**, which gives the value $2r$ rather than r. This is purely semantics, and the results of this research are not impacted. Nonetheless, further research may wish to redefine the critical radius or use a more appropriate name.

For our research, we are in 4D space, so hyperspheres are drawn around the points. The drawing of circles around the points is purely illustrative. The main idea is the distance between the points, which is why the intersection of circles is relevant. The actual computation done by Ripser involves a distance matrix (for m points, it is an $m \times m$ matrix consisting of the distances between the two points for each row and column). This leads to a high memory requirement for large point clouds. My computer can only handle around 30,000 points ($m = 30{,}000$), so it is preferable to use methods that allow for a lower m.

2.3 Data Preprocessing

We are almost ready use this method on the Metropolis data. However, we will first preprocess the data in three ways in order to best apply TDA: normalization, binning, and thresholding.

Although the plots in Figure 1 have the axes at around the same scale, the same cannot be said when d_1 and d_2 are incorporated. The estimates of T_{21} and T_{22} are often in the 100s while the estimates of d_1 and d_2 are usually between 0 and 1 in absolute value. If we apply persistent homology with the parameters at such different scales, we won't capture the impact of d_1 and d_2 since their values are relatively so small, and the hyperspheres will connect points of different d_1 and d_2 values much more quickly than ones of different T_{21} and T_{22} values (if this is hard to picture, image two dimensions where the x-axis values range from 0 to 1 and the y-axis values range from 0 to 100; the point cloud will appear to be a vertical line). To solve this problem, each parameter is **normalized**, so that its minimum value is 0 and its maximum value is 1. The point cloud, then, is inside a hypercube of side length 1, and the maximum possible value of the critical radius is 2 (the distance between the two furthest corners).

Normalizing the parameters allows us to perform the next step, **binning**, quite easily. We divide the hypercube into smaller hypercubes, or bins, of side length 0.01 (there will be 10^8 bins). For each point, we determine which bin it falls in. It may seem difficult to handle so many bins, but the maximum number of relevant bins is m, and realistically, there are much fewer bins to consider. For each bin with at least one point, we count how many points are in it then move on to the third step.

It is possible that a stray point occurs between the connected components. If so, the effect is severe: the critical radius will be smaller than it should because the stray point is as valid as a large cluster of points in persistent homology, and so the radius necessary to connect all the points will not represent the distance between the large clusters. To prevent a situation like this, we want to ignore points that are not in a region of significant density. We achieve this by **thresholding** the bins. We pick a threshold value h and ignore the bins that have a number of points less than h. For much of this report, $h = 1$, and so a bin is kept as long as it has a point in it; however, for larger m, higher values of h may be necessary.

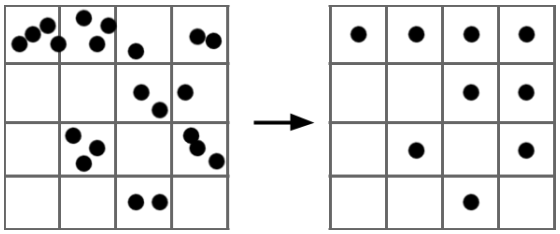

Figure 3: A 2D example of binning and thresholding with a bin size of 0.25 (assuming normalization has been applied) and $h = 1$.

The bins that are kept have a single point placed at their center (see Figure 3 for an example in 2D). These preprocessing steps reduce the size of our data and allows for Ripser to run faster while still preserving the general topological properties of the data. Ideally, the relative values of the critical radii across TI values will not be affected much by the choice of threshold and bin size, but a bin size that is too large may have problems. We use 0.01 as the bin size exclusively throughout this report, and it seems to work well.

2.4 The Full Process

To understand how the point clouds change across values of TI, we will examine how the critical radius changes across TI. However, for the same m and TI, the critical radius can vary each time we run the Metropolis algorithm. To account for this, we run the Metropolis algorithm n times and take the mean of the critical radius for the n runs. Figure 4 shows a histogram of the critical radii for $n = 1{,}000$.

We refer to the mean as the **average critical radius** and n as the **sample size**. We also calculate the standard deviation σ of the n observations for when we later plot the error bars with length $2\sigma/\sqrt{n}$.

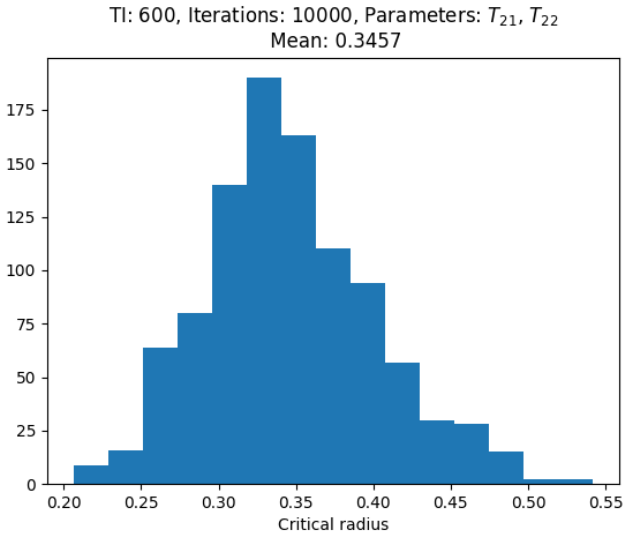

Figure 4: A histogram of the critical radii for $n = 1{,}000$ Metropolis runs with $m = 10{,}000$ and $TI = 600$. Only the T_{21} and T_{22} parameters are used for TDA.

Now that we have discussed all the steps, here is the full process:

1. Generate Metropolis data with m iterations for a value of TI
2. Preprocess the data (normalize, bin, and threshold)
3. Calculate the critical radius
4. Repeat steps 1 through 3 n times and calculate the average critical radius
5. Repeat steps 1 through 4 for different values of TI
6. Track the average critical radius across the TIs

As shown in Figure 1, we expect the critical radius to achieve its highest values between the null points, as that is when there are two connected components. Will anything happen at the null points in particular? What about for TI less than 416 and 832?

One more thing — for early applications of these methods, binning and thresholding were not always used. Sometimes a different step, removing duplicates (caused by a rejection in a Metropolis step), was used instead, as removing duplicates decreases the data size while keeping the exact topological properties. The difference between these methods is likely negligible, but some figures will mention it.

3 Results

3.1 Initial Plots

I initially performed the process for the 2D case (only T_{21} and T_{22}) with $m = 10{,}000, n = 100$, and TI values ranging from 400 to 860. As shown in Figure 5, the critical radius increases significantly a bit after $TI = 416$ and comes back down before 832. The lowest values of the critical radius are below 0.05, indicating that there is only one connected component. These results are interesting, but there's still no way to identify the null points if they weren't marked in red. While it seems that the separation between one connected component and two doesn't occur at the null points, we still have two more parameters to consider.

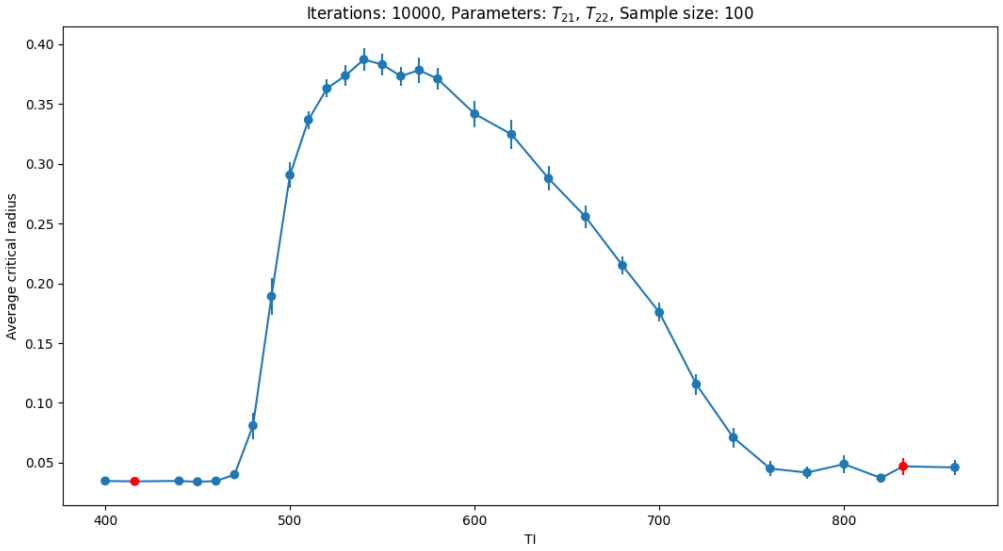

Figure 5: Average critical radius across a range of TI values. Null points (416, 832) are marked in red. The error bars have length $2\sigma/\sqrt{n}$. Only T_{21} and T_{22} are used. Binning is performed with the threshold value as $h = 1$.

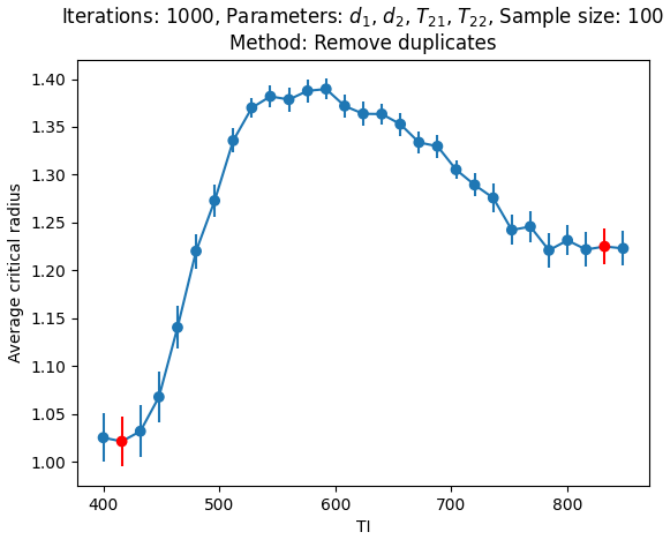

Figure 6: Average critical radius across a range of TI values. All parameters are used.

Figure 6 shows the results when all parameters are used ($m = 1{,}000, n = 100$. The results are better: the critical radius seems to increase immediately after 416 and slowly decreases before 832. If you were given the plot without the null points marked, you might be able to make a reasonable guess for their values. Also note that the separation between TI values used is 16. We'll define this separation as ΔTI and keep it at 16 for future plots until specified otherwise.

There are however, two problems that you may have noticed. The first is that the range of values for TI makes it unclear what is happening before 416 and after 832. Knowing this might provide some useful insight regarding how to estimate the null points given the average critical radii. This problem can be solved easily, and the range will be extended in future figures.

The second is that the average critical radius is much higher than in the 2D case. Even the lowest values are greater than 1, and the 832 side is around 1.25. As mentioned before, the maximum possible value of the critical radius is 2 in a hypercube of side length 1, so values of the critical radius this high mean that there are almost certainly two connected components (which was later confirmed).

3.2 Number of Connected Components

Let's figure out what's happening here. Looking at just T_{21} and T_{22}, the point cloud seems to be one connected component at, for example, $TI = 400$. Figure 5 corroborates this. However, since the average critical radius at 400 in Figure 6 is above 1, that can't possibly be true. The point clouds must be disconnected in 4D but not in the 2D projection. Figure 7 confirms this. The left figure is roughly what is seen when $TI = 400$ in Figure 1 (from a different angle). The point cloud appears fully connected. The right figure, which includes d_1, reveals that the point cloud is actually two connected components! Since this phenomenon occurs when $TI = 400$, the point cloud should be two connected components for all TI (again, confirmed with a persistence barcode).

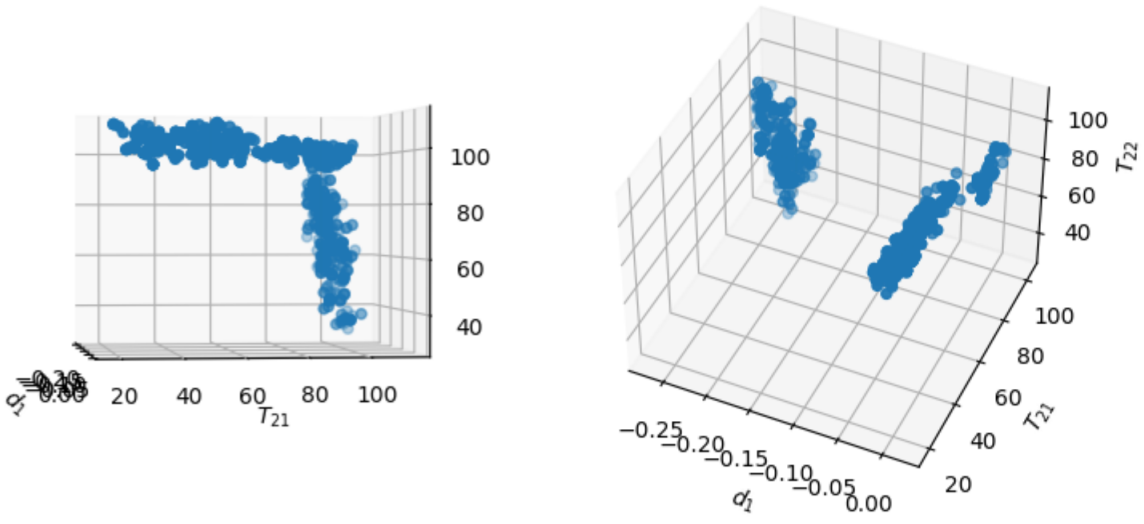

Figure 7: Point clouds for $TI = 400$ and $m = 1{,}000$. Left: A (mostly) 2D projection of the point cloud onto the T_{21} and T_{22} axes. Right: A 3D projection of the point cloud that includes the d_1 axis.

It seems that we have solved this mystery — and for the purposes of this report, we have. Nevertheless, I would like to mention that Figure 7 uses $m = 1{,}000$. The small number of iterations means that the point clouds are small since the Metropolis algorithm can't "walk around" as much. Figure 8 plots the point clouds for $m = 100{,}000$. The T_{21} and d_1 axes are used since if there are two connected components in 4D, there should also be two connected components on those axes.

Notice that the point clouds actually connect at and around the null points. Some thresholding should

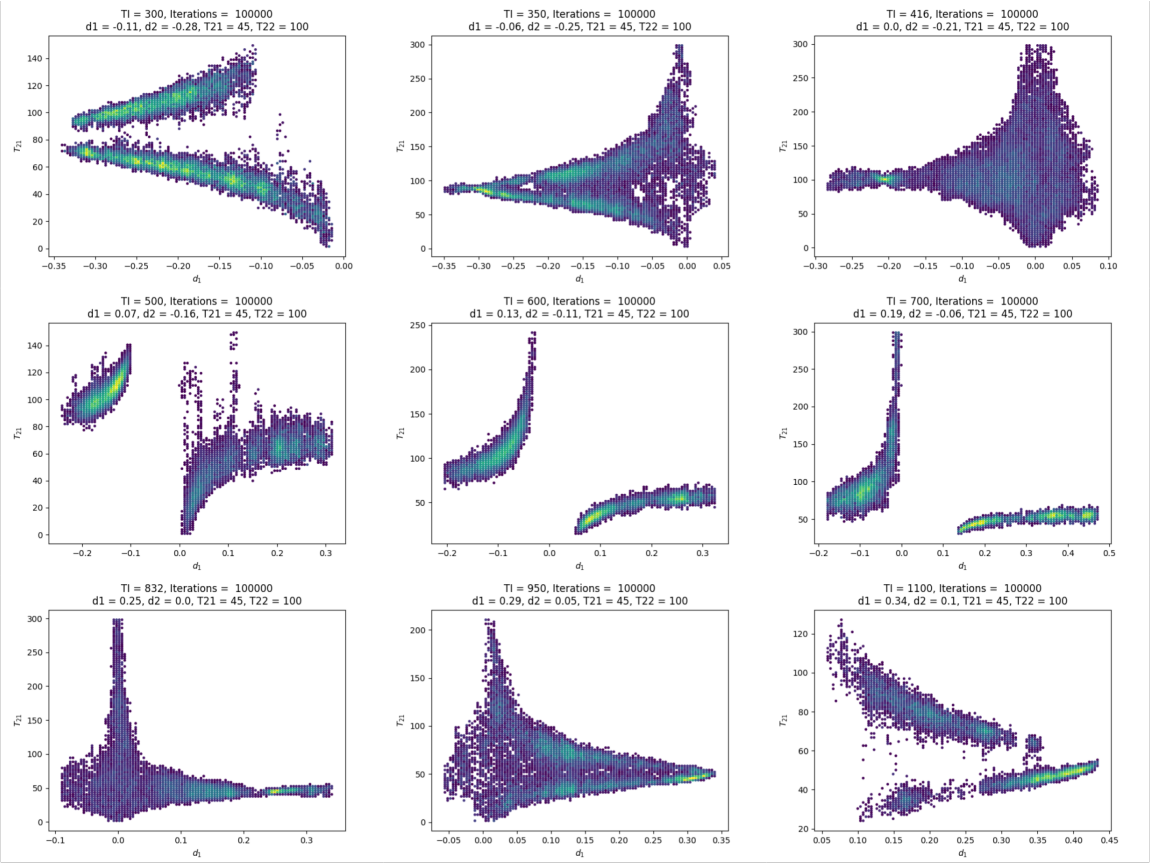

Figure 8: Point clouds on the T_{21} and $d1$ axes for $m = 100{,}000$ across a range of TI values. Denser regions are more yellow while less dense regions are purple.

make the plots with separation less noisy. The density distribution does seem to change between, for example, 350 and 416, but it's unclear how to quantify that. The 350 point cloud seems more separated, but thresholding probably won't do the trick.

These results weren't focused on much in this research since 100,000 iterations is computationally expensive; $m = 1{,}000$ was mostly used instead. Future research may wish to consider this aspect more. In any case, always having two connected components isn't necessarily a bad thing and can actually be beneficial since one connected component for many TI values may cause a relatively flat line in the average critical radius plot (like in Figure 5). We will simply take note of the critical radii being high and proceed with our methods.

3.3 Extending the Range

It's about time that we address the first problem mentioned in Section 3.1: the range of TI values being too small to show what is before and after the first and second null points, respectively. Figure 9 shows some of the plots that result from extending the TI values range to between 256 and 992 for a variety of m, n, and h.

The results are both promising and concerning. The critical radius increases eventually when TI is less than 416 or greater than 832, causing minima to occur near the null points. If we take the minima as our estimates for the null points, however, we'll be a bit off. For most plots, this bias is unacceptable large (in magnitude). Furthermore, the error bars are quite large around many null points. I haven't mentioned this until now since it is implied by the error bar formula ($2\sigma/\sqrt{n}$), but a larger sample size causes smaller error bars. This obviously comes at the cost of a longer runtime, so $n = 100$ was used for all plots except $m = 1{,}000$, for which the number of iterations was small enough that $n = 10{,}000$ could

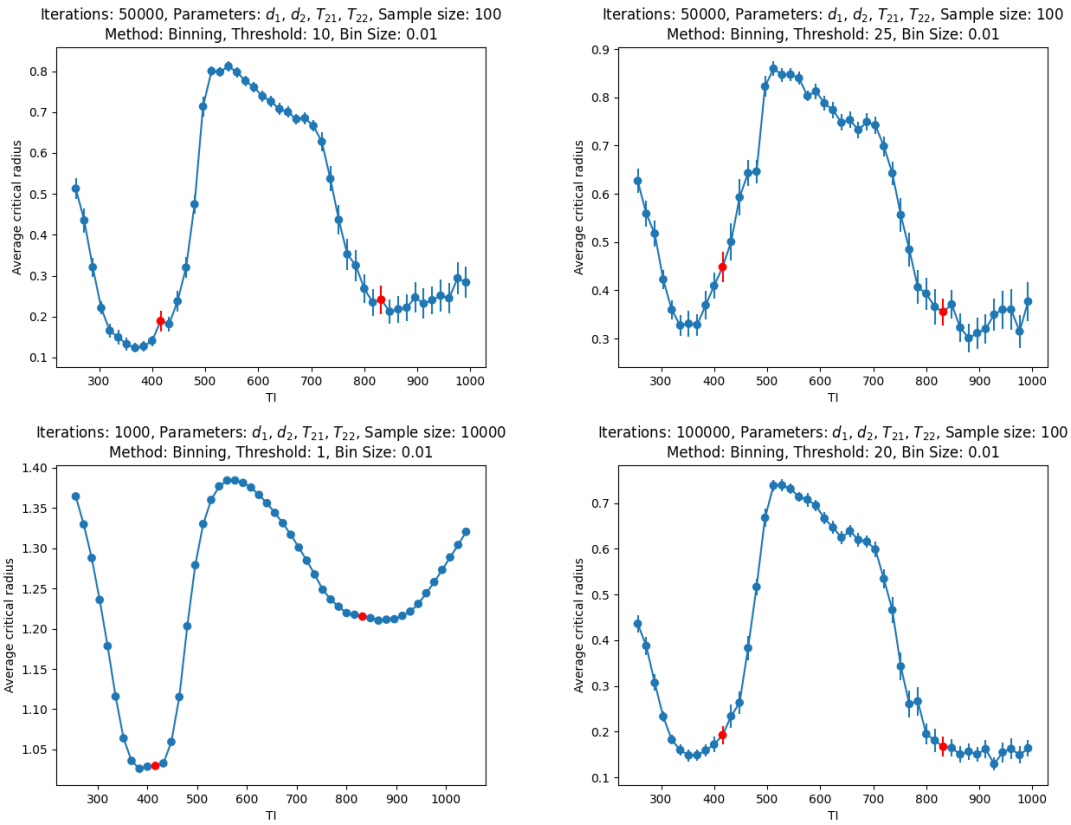

Figure 9: Plots of the average critical radius across a larger range of TI values. Top left: $m = 50{,}000$, $n = 100$, $h = 10$. Top right: $= 50{,}000$, $n = 100$, $h = 25$. Bottom left: $m = 1{,}000$, $n = 10{,}000$, $h = 1$. Bottom right: $m = 100{,}000$, $n = 100$, $h = 20$.

be run in a reasonable amount of time.

I'll briefly comment on the runtime issue now since it is a constant limitation throughout this report. It should be clear that the runtime increases linearly with respect to the sample size n and the number of TI values used. The runtime w.r.t to the number of iteration m slightly more tricky. Ripser's runtime increases faster than linearly with respect to the number of data points given, but applying binning beforehand mostly solves this problem. I haven't looked much into the time complexity in this case, but the Metropolis algorithm seems to take up the vast majority of the runtime, so I've been considering the process as approximately linear w.r.t m.

Now, we still need to figure out what's going on with these plots. The good news is that the same general shape is occurring for different values of m, n, and h. The bad news is that this shape is not what we want to be seeing. The error bars do make it difficult to tell what happening around some null points, but all are somewhat off from the minima. Can the critical radius only take us so close?

3.4 Increasing SNR

I mentioned in Section 2.1 that SNR = 100 was initially tried for generating the 2D point clouds, but the result was just a seemingly unorganized scattering of points and SNR = 1,000 was when they actually took shape. Remembering this, I had the idea to increase SNR in the hopes that it would decrease the bias. For the entirety of this report, we've been using SNR = 1,000, but now let's increase it to SNR = 10,000.

Figure 10 speaks for itself. The null points are exactly at the minima! At least, they appear to be at this scale. Since SNR = 1,000 was somewhat off we expect SNR = 10,000 to be a little bit off but much closer (enough so that it's unnoticeable at this scale). The key part of these findings is that we should be able to decrease the bias by increasing SNR. Moreover, the critical radius now seems to be a great

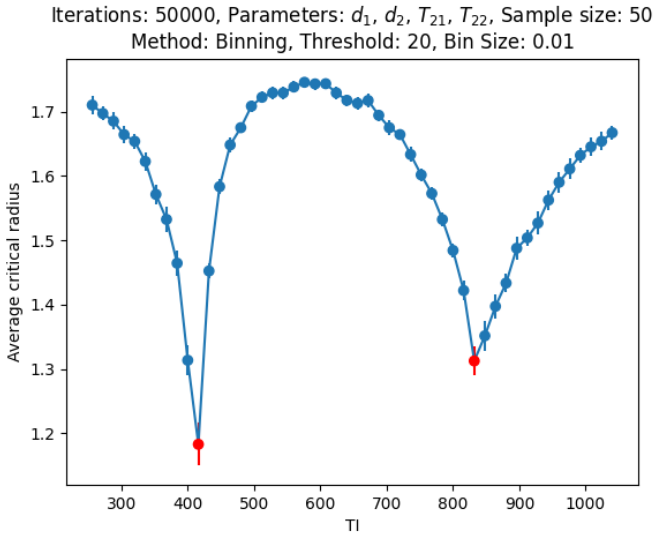

Figure 10: Plot of the average critical radius for SNR = 10,000 with $m = 50{,}000$, $n = 50$, $h = 20$.

method for estimating the null points, at least for low noise.

What about SNR between 1,000 and 10,000? Figure 11 shows the plots for SNR = 10,000, 5,000, 3,000, and 2,000. For $\Delta TI = 16$, SNR = 10,000 and 5,000 have the null points exactly at the minima. SNR = 3,000 does this for 416 and a very good job for 832. Even just SNR = 2,000 is a considerable improvement over SNR = 1,000.

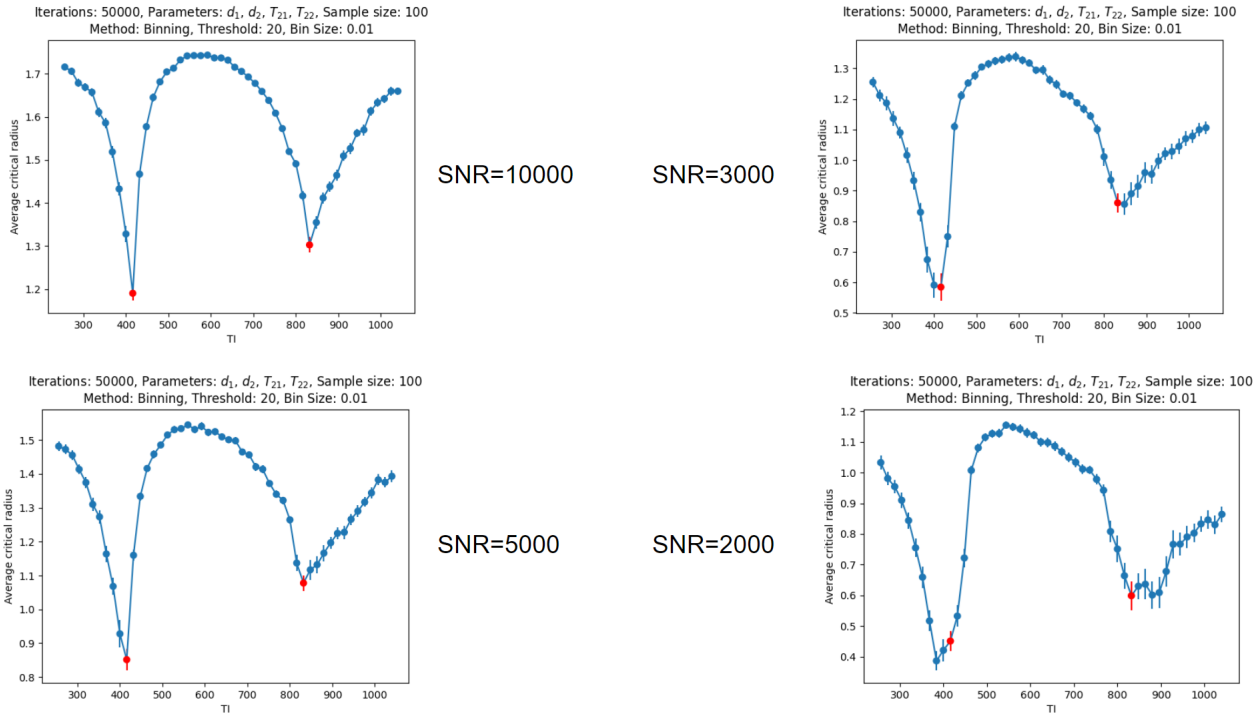

Figure 11: Plots of the average critical radius for $m = 50{,}000$, $n = 100$, $h = 20$. Top left: SNR = 10,000. Bottom left: SNR = 5,000. Top right: SNR = 3,000. Bottom right: SNR = 2,000.

Figure 12 shows the plot for SNR = 10,000 and with $m = 1{,}000$, allowing for us to increase n to 1,000 and decrease the error bars enough to make them hidden, creating a very smooth curve.

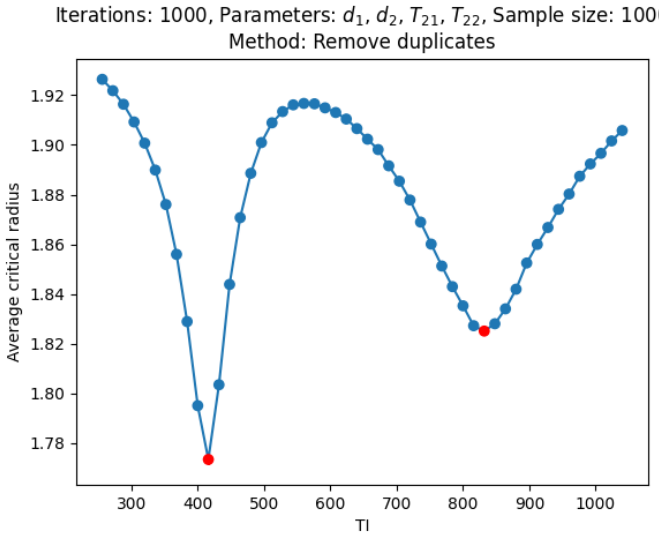

Figure 12: Plot of the average critical radius for SNR= 10,000 with $m = 1{,}000$ and $n = 1{,}000$. The remove duplicates method is used, but binning doesn't affect much for low m.

3.5 Estimating the Null Points

When you can bear to take your eyes off the beauty of Figure 12, consider how we can quantitatively estimate the null points. The minimum is clearly what we want, but we must zoom in (i.e. consider TI values just around the null point) if we want to be more accurate with our estimate. After all, the only reason the first minimum is exactly at 416 is that I set the TI values to 400, 416, 432, etc. knowing that 416 was the null point; without knowing the null point, however, 418 or 420 might also be a reasonable guess. We'll use $\Delta TI = 2$ to be more precise.

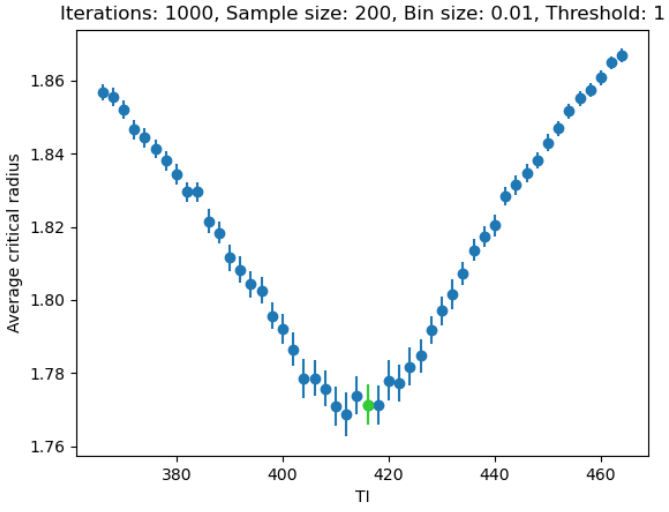

Figure 13: Plot of the average critical radius for TI between 366 and 464, with $\Delta TI = 2$. SNR = 10,000, $m = 1{,}000$, $n = 200$, $h = 1$. The null point is marked in green.

The problem with zooming in is that the error bars become relatively larger and the minimum is unclear. Figure 13 shows that even with SNR = 10,000 and $n = 200$, the point with the lowest average critical radius isn't so clearly the true minimum if n were larger. The zoomed in plot does look quite nice to fit a polynomial function, such as a quadratic or quartic. We can then calculate the minimum of

the fitted function in the range of TIs and use that as our estimate of the first null point. We'll define TI_{min} as this minimum, and $\delta = 416 - TI_{min}$ will be how off our estimate is. We expect δ to decrease as SNR increases, ideally approaching 0.

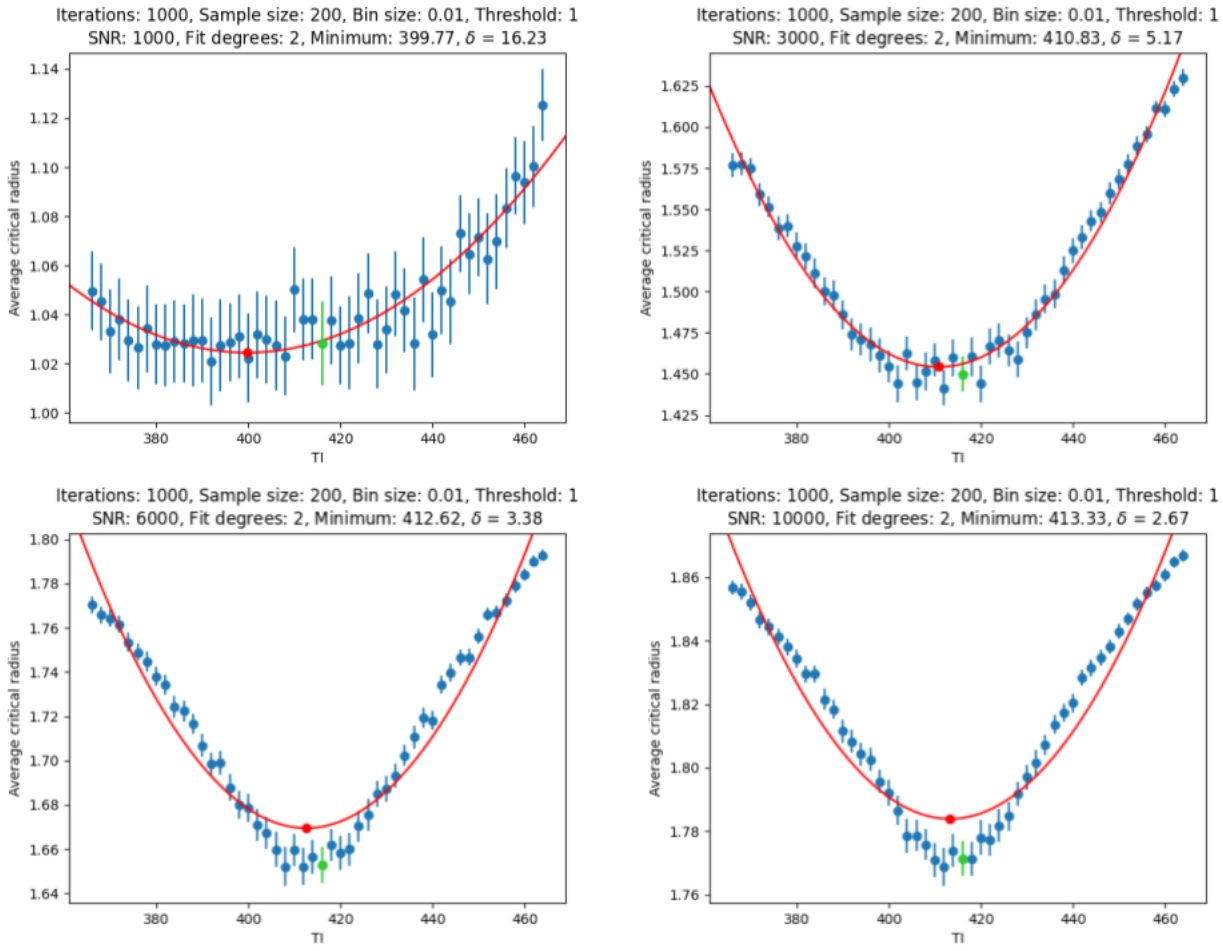

Figure 14: Plots of the average critical radius for TI between 366 and 464 with a quadratic fit applied. The fit is in red and the null point (416) is in green. Top left: SNR = 1,000. Top right: SNR = 3,000. Bottom left: SNR = 6,000. Bottom right: SNR = 10,000.

Figure 14 shows the quadratic fit for four different SNRs. As expected, δ decreases as SNR increases and possibly approaches 0. The most obviously problem is that the quadratic fit isn't particularly good for higher SNR values. Figure 15 shows a quartic fit for a larger range of SNRs, but the even it isn't perfect at the highest SNRs. In fact, SNR = 50,000 has a negative δ.

A plot of δ against SNR may provide some insight. Figure 16 shows this for a quadratic and quartic fit, as well as a log-log plot for both (log refers to the natural log, or ln, here). A straight line in a log-log plot indicates that the relationship is a power function, but the curve plummets around SNR = 30,000 ($\log(\text{SNR}) \approx 10.3$). That's around when δ starts to become negative.

Looking at figure 15, the minimum of the graph for SNR = 50,000 does seem to be at the null point, but quartic fit is a bit off. Perhaps we shouldn't take δ too seriously when it's less than $\Delta TI = 2$. Additionally, the slight asymmetry of the plot may be influencing the minimum.

We can solve both of these problems at once by zooming in even more and considering a TI range between 405 and 425 with $\Delta TI = 0.1$, which should provide more precise estimates. Unfortunately, this is as far as I've gotten in my research (but I'm generating the new data as I write this report).

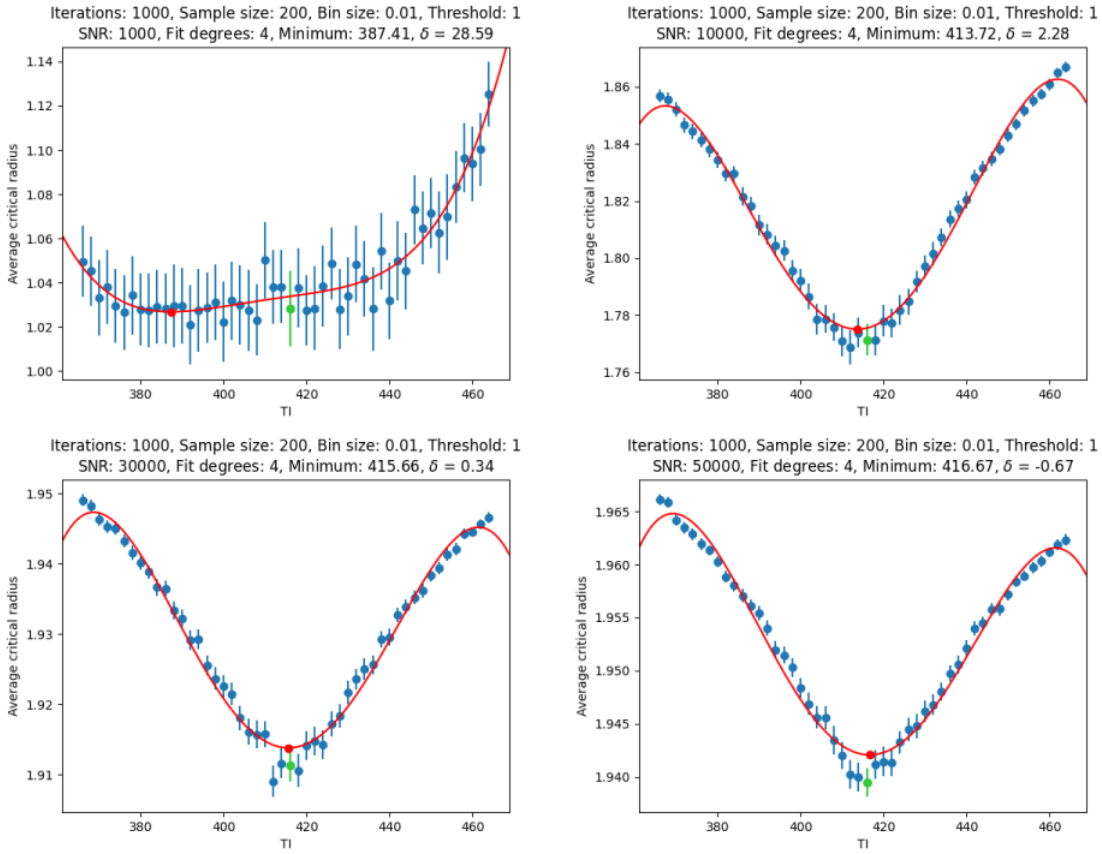

Figure 15: Plots of the average critical radius for TI between 366 and 464 with a quartic fit applied. Top left: SNR = 1,000. Top right: SNR = 10,000. Bottom left: SNR = 30,000. Bottom right: SNR = 50,000.

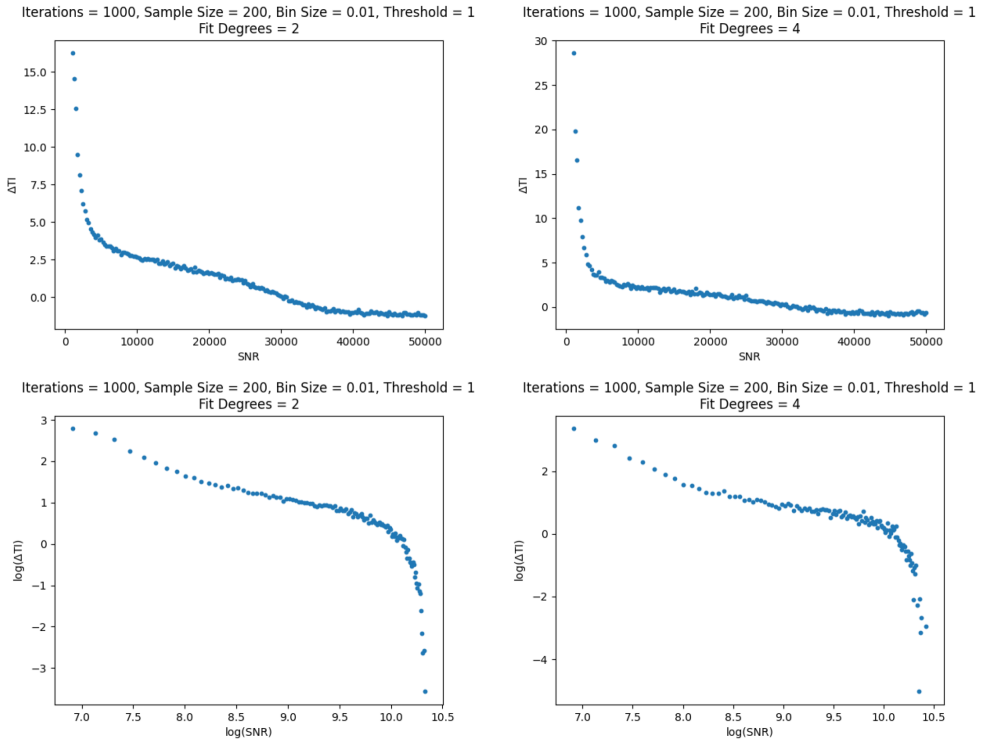

Figure 16: Plots of δ against SNR and their respective log-log plots. SNR what have nonpositive values of δ are omitted from the log-log plots. Top left: A quadratic fit is used. Top right: A quartic fit is used. Bottom left: Log-log plot when a quadratic fit is used. Bottom right: Log-log plot when a quartic fit is used.

4 Conclusion

This research has been, for the most part, very successful. The usage of the critical radius for estimation of the null points is a promising approach that performs well. For high enough SNR, the first null point can be estimated with reasonable accuracy.

Two things remain to be seen on that regard. Firstly, it is necessary to know how these methods perform for a narrower range of TI values. Ideally, δ will approach zero as SNR increases. Secondly, can we express δ as a function of SNR? If so, we can account for the bias by calculating δ and adjusting our estimate.

The second null point has also yet to be studied as thoroughly. While we expect it to behave similarly to the first but with the minimum as an overestimate, the full plot of average critical radius against TI isn't quite symmetric, and the critical radius tends to be higher around 832 than 416. It will be necessary to investigate if our methods are as successful there.

Once we can estimate the null points with a desirable accuracy, we can estimate T_{11} and T_{12}. This is as simple as dividing by ln(2). Naturally, the next step is to test out our methods with different values of T_{11} and T_{12} (and thus different null points). In fact, we've been using the same default values of c_1, c_2, T_{11}, T_{12}, T_{21}, and T_{22} this entire report, so different ones should be used in the future.

I'd also like to recognize that the TDA and the critical radius perhaps aren't the only methods that could be used to identify the null points. After all, our use of TDA is somewhat basic for much a powerful tool. I know that condition number is being studied by Dr. Spencer and Griffin. In addition, perhaps a clustering analysis method such as k-means clustering (with k = 2) could be used such that a metric analogous to the critical radius is defined as the distance between the centroids.

I'll also admit that the TDA section of this paper was quite minimal, and if desired, I'd be happy to write up a more detailed explanation of persistent homology and how to use Ripser and its usage in my code. If you're just looking for an straightforward introduction, this video helped me when I was learning.

At any rate, the work that has been done so far has good prospects that will hopefully achieve polished, robust results. Much of the necessary work has been done so far, and as stated in the introduction, the code is available for further research. If someone continues my work, they are welcome to use it as well as contact me about what I've done, what progress has been made, or anything else they need. My UD email is thecht@udel.edu, but since I'll stop using it soon, my personal email thecht319@gmail.com or new university email thecht2@illinois.edu would be better.