Systems Biology II
Adapting classical fitting algorithms to cancer-damaged cell oscillations

Mathias D. Kallick
Department of Computer Science
Colby College
Waterville, ME, 04901
Email: mkallick@colby.edu

Neil K. Sefah
Department of Computer Science
Colby College
Waterville, ME, 04901
Email: nksefah@colby.edu

Dakota Thompson
Department of Computer Science
Colby College
Waterville, ME, 04901
Email: dthompso@colby.edu

***Abstract*—For this project, we were tasked with analyzing oscillations of Per2 and BMal1 in data recorded from cells damaged by cancer - specifically, we wanted to find the period of the oscillations. Michelle Farkas, from the University of Massachusetts Amherst Department of Chemistry, provided us data from cancer-damaged cells. This data was highly irregular and noisy, making a straightforward computational analysis impossible. However, by using a hand-made suite of data cleaning tools and a combination of broad peak searching and sensitive peak detection, we were able to create a piece of software that consistently finds all of the peaks and troughs in the data that was provided to us. This allowed us to produce a function of the period over time for individual datasets.**

I. Introduction

The study of circadian clocks within organisms has been a growing field in systems biology. We specifically looked at the oscillations of Per2 and BMal1 proteins in cells damaged by cancer. Michelle Farkas, from the University of Massachusetts Amherst Department of Chemistry, provided us data from cancer-damaged cells that were starved for 24 hours, 12 hours, and not starved at all. It's important to understand the period of these oscillations as knowing the period plays into understanding the mechanics of the lager celular systems. However, due to the nature of the data there were no consistently accurate automated peak detection system in place. The data has a couple of idiosyncrasies that made automated analysis harder - the oscillations are damped, there was a downwards trend in all of the datasets, and there was a lot of noise in the data. Furthermore, most period finding algorithms are designed to work for data that has a constant period, but we found that this data had a period that varied with time. Moving forward with the task of determining the period we proceeded with several initial steps. This included analysing smoothing techniques as we needed to smooth the data while ensuring that it remained as true to the actual data as possible through manipulations and derivatives. Additionally, the use of genetic algorithms to create a multiple cosines function for obtaining the periodicity was explored. The classic use of derivatives for finding peaks to calculate the period was also investigated for determining the best automated period detection tool.

II. The Need for Smoothing

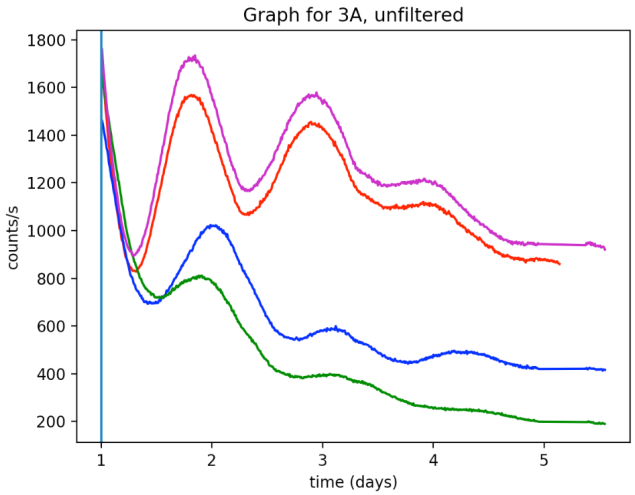

Fig. 1: Look at this mess. Look upon it, and weep.

III. Windowing

One of the core problems we needed to solve before tackling any others was making our data smoother. Initially, all of the data we worked with was extremely noisy, and yielded problems when we initially attempted to detect peaks or find accurate periods. In pursuing this, all other computation related to the project run would run much more efficiently and with less error. The primary method that we started smoothing with filters. The first that we got to work properly was the Savitzky-Golay filter, one developed by increasing the signal-to-noise ratio of the plot without entirely corrupting the signal. To do this it essentially generates new fitted subsets of the data according to an equation. While this method has been working pretty effectively for us so far, we wanted to find out whether or not we could find better results by trying to create new filters based on different functions. For this, we turned to window functions; methods that are designed to fit data

zero-valued/normalized according to a given interval, using an equation. These were normally used in signal processing as these algorithms are usually geared towards frequency responses and plots for proper spectral analysis (primarily detecting things like spectral leakage and other frequency/sinusoid related plots and problems). Yet, these algorithms could potentially be used to smooth any plot that was given as a signal, depending on its purpose.

However, most of these proved ineffective. The windowing methods we tested were primarily the ones most commonly used in signal processing/general usage. These included the Hamming window function, the Blackman Harris, the Flat-Top, and the Bartlett-Hann functions.

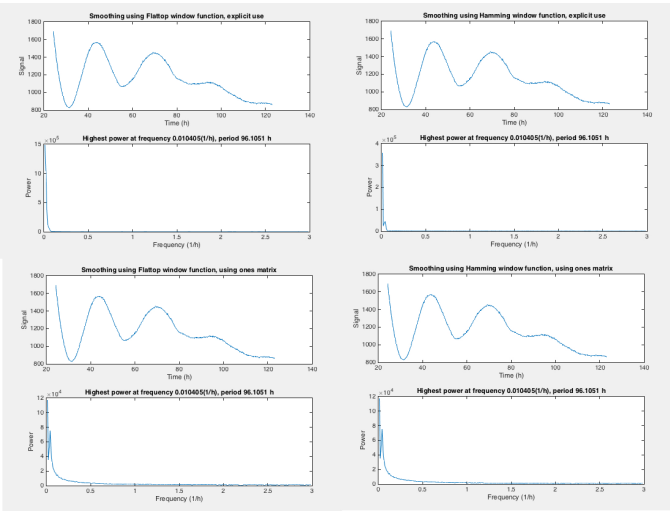

Fig. 2: Hamming and Flattop Window Test: Using MATLAB code, this shows us one of the sets of data, and shows the effect of using the Hamming and Flattop window functions to dampen it. The bottom result is supposed to show us the periods detected after the application of the algorithm, however, it is not detecting the periods properly, likely due to the cancerous data. This is further confirmed by the period being detected, which is much larger than it should be, as the periods should be 20-30h at best. The explicit use versus the ones matrix indicates if there was an additional matrix used in mitigating the effects of the algorithm on the signal. Explicit use meant no additional matrix was used, ones matrix indicates that ones matrix was used.

However, all of them yielded problematic results, as they would not give us accurate period detection, no matter what we tried. Considering the nature of these functions (with most of them being derivatives of a core few, which we tested for), we then concluded that these would be ineffective for our purposes, at least in the timeframe we were working in.

We had also tried to apply a few methods of dealing with outliers in the data, previously working with two methods for removing outliers based on standard deviation, and then a custom spike smoothing method which is intended to isolate where data spikes and then modify them according to the adjacent points in the plot, rather than an outright removal. This would have us then decide whether or not smoothing or removal is better for our data in the preliminary stages of our analyses. However, actual filters trumped those as well, but, this was not totally unexpected as the filters we were previously working with were industry standard for smoothing.

IV. Genetic Algorithms

One of the most common ways to tackle the period-finding problem is to do curve fitting. That is, find the correct parameters to a function in order to get the difference between that functions outputs and the true data as small as possible. Then, we extract the period from that function (since it should be easy to mathematically determine the period from a set function). Although this method works extremely well, it should also be clear that finding the correct parameters for the function is prohibitively time-consuming (since the potential options are infinite, and testing each set of parameters requires an evaluation of the function at every time step). However, given how common this approach is (and how common the parameter-finding problem is), a large number of algorithms exist to get approximate best parameters in more reasonable computation times. For this problem, we chose to use a genetic algorithm, which is one of the most common ways to do parameter-finding (also, Mathias had working code from Systems Biology I, so it was relatively easy to implement). Genetic algorithms are meant to emulate the way that evolution works (in the loosest sense of the word). To do this, they take a random sampling of the parameter space, to get the parameters for the first group of parents (each of which is simply a full parameter set for the function, so for example, for Equation 1, we would have a C, five As, five ϕs, and a τ). The next step is to evaluate the quality of each parameter set using a cost function (for us, this was simply a least-squares difference between our data and the function defined with each parameter set), and sort the parents according to the quality of their fit. Once the parents have been sorted, we take a subset of the parents according to a selection strategy. What we used for this project was tournament selection - for each parent that we pick, we first pick a random subset of the parents, then we take the best parent from that subset. For this version of a genetic algorithm, we pick the same number of parents as we had in the last round - we just pick the best parents multiple times, and exclude the worst parents, by design. This whole process is then repeated for either a set number of generations (in our case, 10-20), or until the error for the best parameter set becomes lower than some threshold. The best parameter set at the end of this process then becomes the parameters that we use for the function.

V. Zielinski et al. - Multiple Cos Fitting

Given that this approach is the tried and true approach to period-finding, we decided to use it as a starting point for this project. Zielinski, Moore, et al. published a paper detailing various methods of period finding for circadian data [1], from which we took our first function, which is called mFourFit. This function (seen in Equation 1) makes the assumption that

any periodic data can be represented as a linear combination of cosines with different periods. It simply adds together j cosines, where cosine i has period τ/i and amplitude A_i. It also makes the assumption that it is possible to model the data using only periods that are multiples of each other (so there is one base period, τ, and then the rest of the periods are τ divided by an integer, i). An example of a function generated using mFourFit and a random set of parameters can be seen in Figure 3. Zielinski also described another function, FFT-NLLS, which can be seen in Equation 2. FFT-NLLS is very similar to mFourFit, but it doesnt assume that the periods are multiples of each other. Instead, it uses j individual periods that are all found during the parameter-finding phase. This potentially provides a better fit to messier data, but at the same time it dramatically increases the computation cost of parameter-fitting.

$$f(x) = C + \sum_{i=1}^{5} A_i \cos(\frac{2\pi(t-\phi_i)}{\tau/i}) \tag{1}$$

$$f(x) = C + \sum_{i=1}^{5} A_i \cos(\frac{2\pi(t-\phi_i)}{\tau_i}) \tag{2}$$

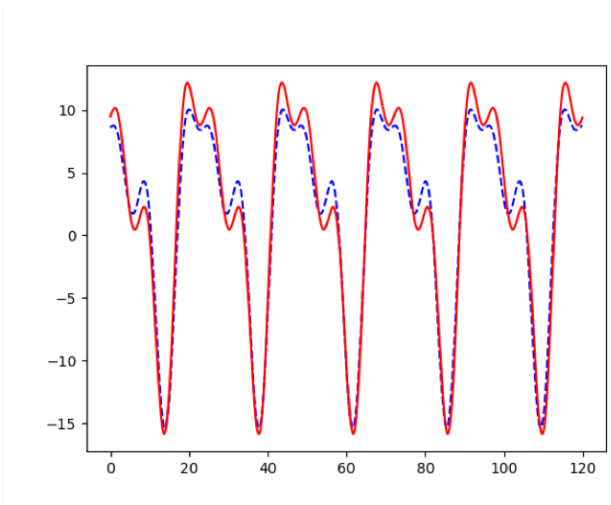

Fig. 3: A plot of artificially generated data that has then been fit to using the mFourFit equation.

For this project, we used a genetic algorithm to fit parameters for both FourFit and FFT-NLLS. Unfortunately, it quickly became clear that despite running our genetic algorithms for 10-15 hours, we still found that the fitting would simply fit a curve through the middle of our oscillations, picking up the general trend of the data, but not the individual oscillations (that is, totally missing the peaks and valleys). An example of this for FFT-NLLS can be seen in Figure 4. FourFit looks exactly the same, but trains faster because there are significantly less parameters. This was a truly discouraging result, given that this was the most well-known way of producing the periodicity of data. That said, we were still able to get some interesting information with this fitting, since the fitting would still recognize the overall trend of the data. The downwards trend of the data was a big part of the messiness that made this data hard to work with, so knowing the trend would allow us to correct that trend and work with easier data.

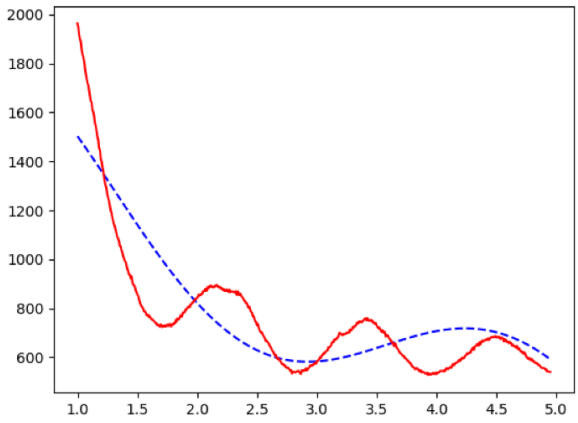

Fig. 4: A plot of our real data, and the fit we found using a genetic algorithm with FFT-NLLS. The fit does not get the individual oscillations, but it does find the general trend of the data.

VI. Correcting the Data with Polynomial Fit

When we first realized that we could use the genetic algorithm to correct the data, we were very excited to have solved one of our major issues with the data. However, we soon noticed a major issue - running a fitting algorithm for 10 hours on a single dataset is not the most efficient way to clean data. At that point it is cheaper and easier to have a human do it. Our solution to this problem was to look for a simpler fitting algorithm. After testing a few, we settled on simply using a third-degree polynomial fit to find the trend of the data. This is a much lower level fitting algorithm, and it does not fit complex data nearly as well as the FourFit or FFT-NLLS functions, but our datas downwards trend is pretty simply, so it works very well for that, and, importantly, it fits quickly enough that it is not a noticeable slowdown in our code. As can be seen in Figure 5, the third-degree polynomial fit finds the trend in our data perfectly. If we then invert the fit, and add it to the raw data, we get a corrected dataset (Figure 5) that is much, much easier to analyze. This result allowed us to improve our peak finding substantially.

VII. Peak Finding with Derivatives

Standard peak finding techniques have traditionally relied on the first derivative to find peaks and valleys. This process looks for where the first derivative crosses zero and depending on the direction a peak (positive to negative) or valley (negative to positive) is found. When applying this technique the first derivative correctly detects most peaks and valleys. However, due to damping and the downward trend of the data it fails to

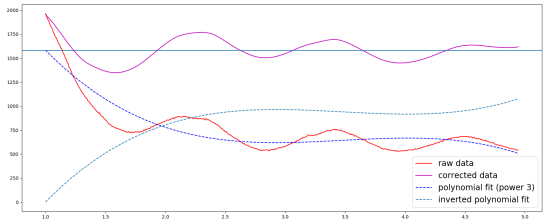

Fig. 5: Using the third-degree polynomial fit to find the downwards trend of our data, and correcting the data based on that downwards trend.

find peaks that optically appear to be significant in the systems period. This can be seen in Figure 6 as in between the first and last valley peak pairing there appears to be an oscillation in the data that doesn't fully pass the first derivatives threshold. Thus to find the optically noticeable peaks we needed a more sensitive measure for which we turned to the third derivative. The Third derivative operates in the same manner as the first derivative for finding peaks and valley but it also looks at the curvature of the data as well. The third derivative correctly detects peaks and valley including the visually significant peaks and valleys. However, it also detects peaks and valleys where there don't appear to be any significant to the periodicity which can be seen in Figure 7. This signified a hybrid method with a sensitivity greater than the pure first derivative and less than the pure third derivative would be required.

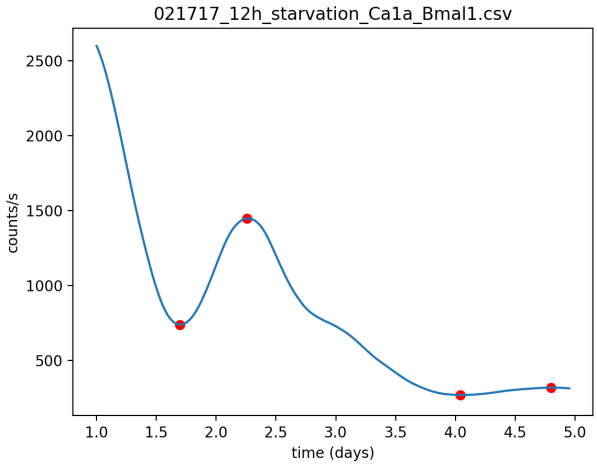

Fig. 6: Using the first derivative to find peaks.

VIII. Our Peak Detection Process

The final peak detection method utilized correcting data through polynomial fitting, the first derivative, and the third derivative. First the data was smoothed using the Savitzky-Golay filter to remove noise from the data. A third degree polynomial was fit to the dataset, thus identifying the major damping trend. This third degree polynomial was then inverted

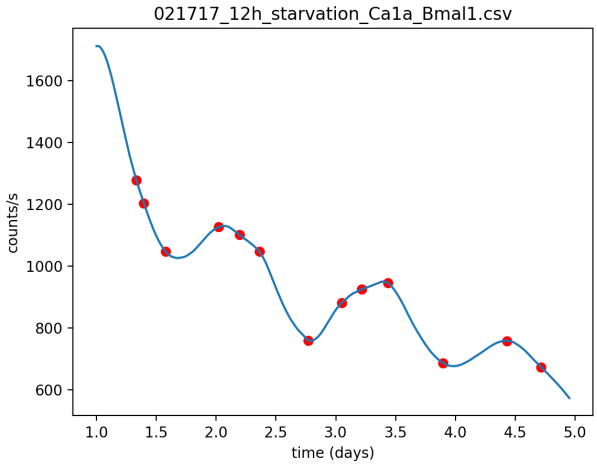

Fig. 7: Using the third derivative to find peaks.

and added to the actual data thereby removing the overarching damping and creating a corrected dataset. By removing the overarching damping trend optical peaks became more susceptible to detection by the first derivative. The peak and valley identification subsequently began by using the first derivative to identify peaks and valleys in the corrected dataset. The average time between peaks and valleys was then calculated and if a gap between peaks and valleys larger than 1.33 times the average distance was detected it was marked as missing a peaks and/or valleys. The third derivative was then applied to the uncorrected data strictly within identified gaps to find the remaining peaks and valleys. The peaks and valleys initially found with the first derivative of the corrected data and those found using the third derivative on the uncorrected data in identified gaps were then consolidated on the uncorrected data to return the set of final peaks and valleys. With the final set of peaks and valleys the period could then be calculated. This process is shown in Figure 8.

IX. Using Peaks to Find the Period as a Function of Time

Having found the peaks and troughs, we need to find the period of the oscillations in the data (since that was our original goal). Normally, the way to turn a set of peaks and troughs into a period would be to find the difference in time between each successive peak and trough and then double that time (since the period is defined as peak-to-peak time, not peak-to-trough time). However, our data does not have a consistent period, so doing that would provide a misleading result. In order to remedy that, we calculated our period for each pair of peaks and troughs (so for peak one to trough one, then from trough one to peak two, and so on). We then plotted this against the time that the peaks were found at (See Figure 9). This gives us an idea of the period at discrete points throughout the data, but we wanted to see how the period changed continuously over time as the oscillations occurred. In order to do this, we simply plotted a third-degree fit to

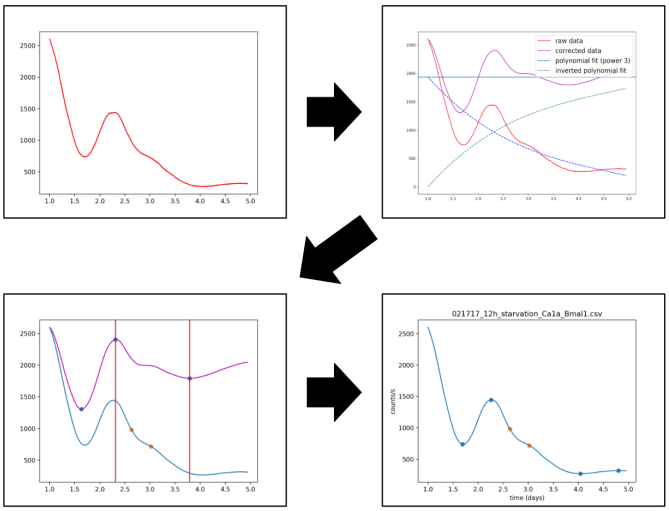

Fig. 8: Our final peak and valley detection process. This process worked consistently for every dataset that we tried it on.

the discrete period measurements (again, see Figure 9), which allows us to get a sense of the instantaneous change over the data collection window.

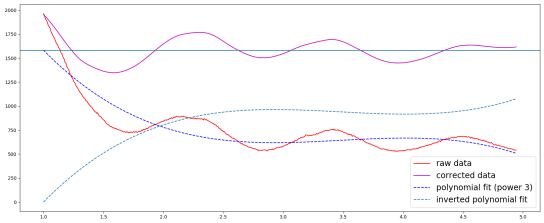

Fig. 9: A plot of the found period over time of one of our datasets. We also plotted the third-degree polynomial fit of the found period data in order to get a continuous function of the period over time.

X. Conclusion

By using a hand-made suite of data cleaning tools and a combination of broad peak searching and sensitive peak detection, we were able to create a piece of software that consistently finds all of the peaks and troughs in the data that was provided to us. We ended up determining that finding the period in data like this is extremely difficult, particularly because noise makes the derivatives useless, the downward trend of the graph makes some peaks not show up in derivative analysis, and the damping of the oscillations over time makes the last part of the data very hard to analyze. We also noticed that the fact that the period changes over time makes the coding problem harder, because we cannot simply assume that the peaks will be consistently spaced throughout the data, so placing constraints can end up being harder. However, by smoothing the data using the Savitzky-Golay filter, we removed our issues with noise entirely. We tried using a Genetic Algorithm to fit a combination of multiple cosines to the data, using both the mFourFit (Equation 1) and FFT-NLLS (Equation 2) equations. However, this simply provided us with a general trend fitting, and not a oscillation fitting. We were able to use polynomial fitting to get the same general trend fitting as the Genetic Algorithms, but with a much, much faster fitting time (10 hours compared to ¡1 second). Using that trend fit, we corrected the downwards slope of the data. We then used first-derivative peak detection to cast a wide net and get the most obvious peaks. By using the mean of the time between peaks, we were able to find places where there should be missing peaks (anywhere the time between found peaks was bigger than 1.3 times the mean). Inside of those areas, we then used the much more sensitive third-derivative peak detection, which consistently found all of the missing peaks and troughs that first-derivative didnt find. Having a consistent way to find the peaks and troughs in the data allowed us to plot the period over time for all of the data sets that we were given.

Acknowledgment

The authors would like to thank Stephanie Taylor for all of her amazing help guiding us through this project. We would also like to thank Michelle Farkas, from the Department of Chemistry at University of Massachusetts Amherst for the data.

References

[1] Zielinski T, Moore AM, Troup E, Halliday KJ, Millar AJ (2014). *Strengths and Limitations of Period Estimation Methods for Circadian Data*. PLoS ONE 9(5):e96462. doi:10.1371/journal.pone.0096462

[2] Eric Jones and Travis Oliphant and Pearu Peterson and others (2001–). *SciPy: Open source scientific tools for Python*. "http://www.scipy.org/", [Online; accessed 5/8/2018]

[3] Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing In Science & Engineering, IEEE COMPUTER SOC, 10.1109/MCSE.2007.55.

[4] (2017). *MATLAB Optimization Toolbox*. The MathWorks, Natick, MA, USA.