# 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—*

## I. Introduction

## II. The Need for Smoothing

## 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, but, 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. 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.

We had also tried to apply a few methods of dealing with outliers in the data, however, actual filters trumped those as well.

## 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 $\phi$s, and a $\tau$). 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, 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 $\tau/i$ 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, $\tau$, and then the rest of the periods are $\tau$ divided by an integer, $i$). An example of a function generated using mFourFit and a random set of parameters can be seen in Figure 1. 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.

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

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

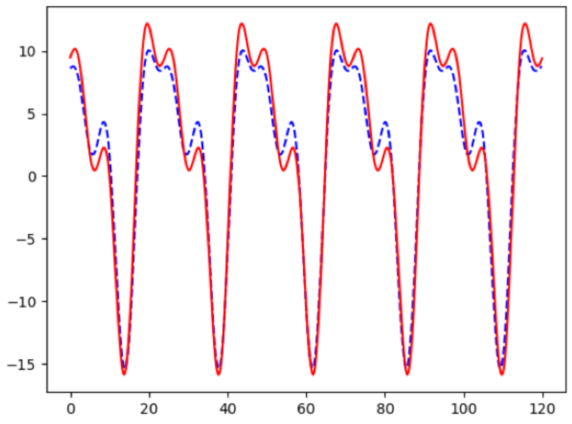

Fig. 1: A plot of artificially generated data that has then been fit to using the mFourFit equation.

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 2. 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.

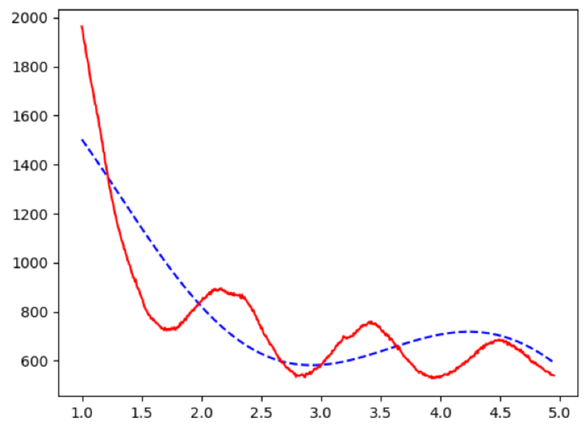

Fig. 2: A plot of our real data, and the fit we found using a genetic algorithm with FFT-NLLS. The fit does not get the individual oscillations, but it does find the general trend of the data.

## 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 **??**, 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 3) that is much, much easier to analyze. This result allowed us to improve our peak finding substantially.

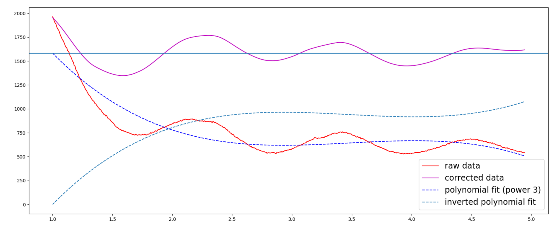

Fig. 3: Using the third-degree polynomial fit to find the downwards trend of our data, and correcting the data based on that downwards trend.

## VII. Peak Finding with Derivatives

## VIII. Our Peak Detection Process

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

## X. Conclusion

## XI. Acknowledgments

## XII. References