ECSE 443 – Assignment 2

McGill University

Tristan Bouchard - 260747124

February 24, 2019

Question 1 – Curve Fitting

The first question in this assignment provided us with an array of data. We then had to attempt to fit different curve functions to the data while minimizing the Square Error. There were three types of functions to fit in this question: Polynomial, Exponential and Inverse, each of which presents its advantages and disadvantages. The results depended strongly on the data, as demonstrated in the sections below. The general approach I took to these problems was to first linearize the equation, such that I could use the **normal equations** method, using matrices.

Part a) Polynomial Fit

This section of the assignment required us to fit a second order polynomial, of the form $f(t) = a0 + a1t + a2t^2$ to the data provided. The method involves solving the following system:

$$\begin{bmatrix} 1 & t_0 & {t_o}^2 \\ \vdots & \vdots & \vdots \\ 1 & t_n & {t_n}^2 \end{bmatrix} \times \begin{bmatrix} a0 \\ a1 \\ a2 \end{bmatrix} = \begin{bmatrix} f(t_0) \\ \vdots \\ f(t_{n)} \end{bmatrix}$$

Where the first matrix, called *A*, contains all the sample time points provided. The second vector, called *a*, contains the coefficients of the polynomial. Finally, the last vector, called *y*, contains the data point associated with the time value in the A matrix. This system can be solved in the following manner:

$$A \times a = y$$

Multiplying on the left with the transpose of A:

$$A^T A \times a = A^T y$$

Solving for the coefficient vector, a

$$a = (A^T A)^{-1} * (A^t y)$$

Thus, solving for this overdetermined system yields the vector of coefficients that minimize the squared error. From the MATLAB Script I wrote, the equation along with the coefficients (rounded to the 10th decimal place) are:

$$y = 0.2509222625 - 0.1113459194t + 0.009651643t^2$$

Where

$$a = \begin{bmatrix} a0 \\ a1 \\ a2 \end{bmatrix} = \begin{bmatrix} 0.2509222625 \\ -0.1113459194 \\ 0.009651643 \end{bmatrix}$$

The fit yielded by these coefficients is as appears below, in **Figure 1**.

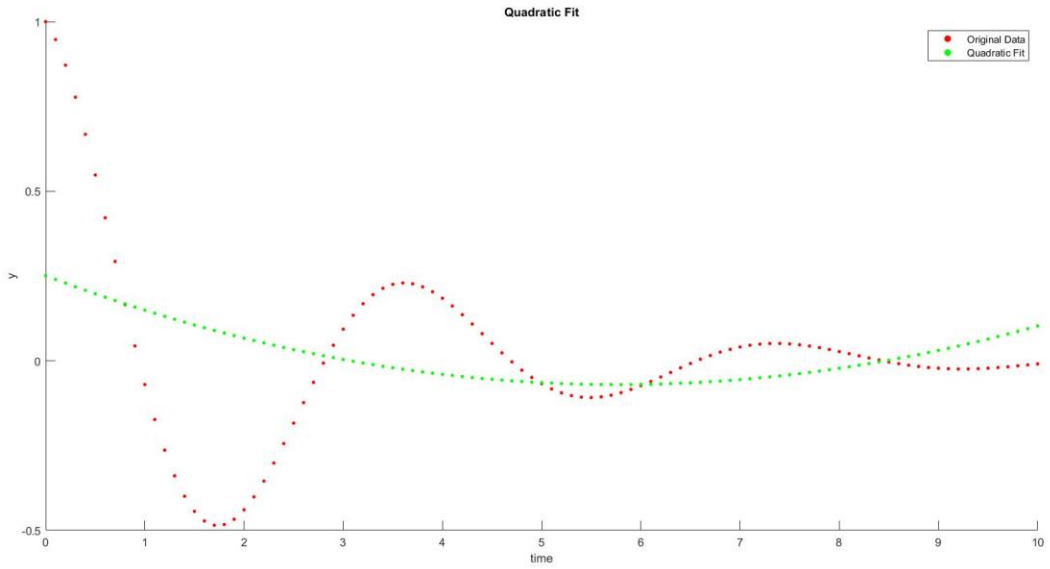

Figure 1: Quadratic Fit

The square error of this fit is of 6.3286. This was computed by subtracting each point of the quadratic function to the original data set, squaring each value, then adding them together.

$$error = \sum_{i=0}^{n} \left(Original_{Sample_i} - fit_{function_i} \right)^2$$

Equation 1

Such, considering that there are 101 data points in the set, we can say that this fit is relatively good. The main reason that the polynomial fit is good in this situation is its ability to conform well to high frequency changes, where high variability is present in a small portion of the domain of the function. However, the polynomial's behavior over the entire domain of the function, especially in terms of the steady state behavior, if often less than ideal. This can be noticed for $t = [9, \infty)$, where the polynomial fit begins to diverge. However, the square error here is very low because the data only goes up to $t = 10$.

1.b) Exponential Fit

This section of the assignment required us to fit an exponential type function to the provided data. The exponential function was of the form:

$$f(t) = a_0 t^{a_1}$$

My approach to linearizing this equation was simply to take the natural logarithm of both sides, then solve for the linearized coefficients using the normal equations method:

$$\ln(f(t) = \ln(a_0 t^{a_1}))$$

$$\ln\big(f(t)\big) = \ln(a_0) + \mathrm{a}_1 \ln(t)$$

Then, solving for $\ln(a_0)$ and $a1$ simply required placing the coefficients in the matrices as follows:

$$\begin{bmatrix} 1 & \ln(t_1) \\ \vdots & \vdots \\ 1 & \ln(t_n) \end{bmatrix} \times \begin{bmatrix} \ln(a_0) \\ a1 \end{bmatrix} = \begin{bmatrix} \ln(f(t_0)) \\ \vdots \\ \ln(f(t_n)) \end{bmatrix}$$

Then simply take the exponential of the result received from $\ln(a_0)$ which solves for a_0.

$$a = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0.072688765244001 \ + \ 0.294592186754102i \\ -1.109434267418071 \ + \ 0.275376751272871i \end{bmatrix}$$

Who, when placed in the original equation, yield the following:

$$f(t) = (0.072688765244001 \ + \ 0.294592186754102i)t^{-1.109434267418071 \, + \, 0.275376751272871i}$$

Then, the curve of the above equation, when superposed to the original data can be observed in **Figure 2** below, which demonstrates the real parts of the equation above.

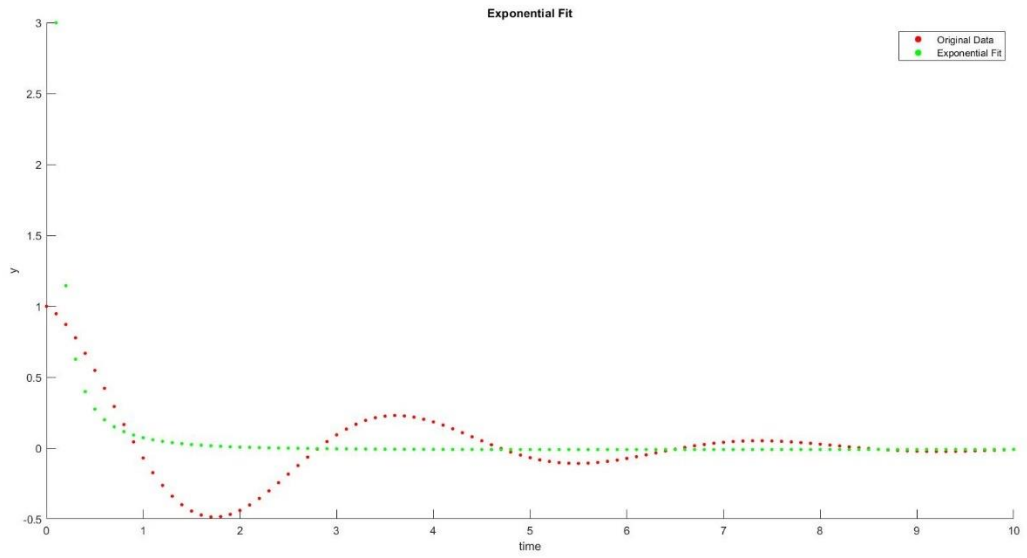

Figure 2: Exponential Fit

What is interesting to observe in this equation and fit is first that the coefficients are complex. This is due to the phase correction of the regression curve. Another interesting observation to make about this curve is the steady state behavior, which is very similar to the original data and tends to the same value as time nears $t = 10$.

$$error = 7.5619$$

The error computed, according to **Equation 1**, is a little bigger than the quadratic fit. This may be cause by the fact that the exponential regression follows the variation if the function less well that the quadratic, especially with respect to the extrema that the quadratic deals with very well.

1.c) Inverse Fit

The final part of this question involved fitting an inverse function to the data, where the equation is of the form

$$f(t) = \frac{1}{a_1 + a_0 * t}$$

The strategy I made use of was to, as in the previous question, linearize the equation and solve using the normal equations method. First, I linearized by inverting the equation:

$$f(t)^{-1} = \left(\frac{1}{a_0 + a_1 * t}\right)^{-1}$$

$$\frac{1}{f(t)} = a_0 + a_1 * t$$

Then, solving for the coefficients by making the data in the *y* vector inverted, in the following manner:

$$\begin{bmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1/(f(t_0) \\ \vdots \\ \frac{1}{f(t_n)} \end{bmatrix}$$

Solving as in a similar manner to above, the coefficients obtained are:

$$a = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 2.403590506890193 \\ -1.744846214839930 \end{bmatrix}$$

Which yields the equation

$$f(t) = \frac{1}{2.403590506890193 + \ -1.744846214839930t}$$

When plotted against the original data, as in **Figure 3** below.

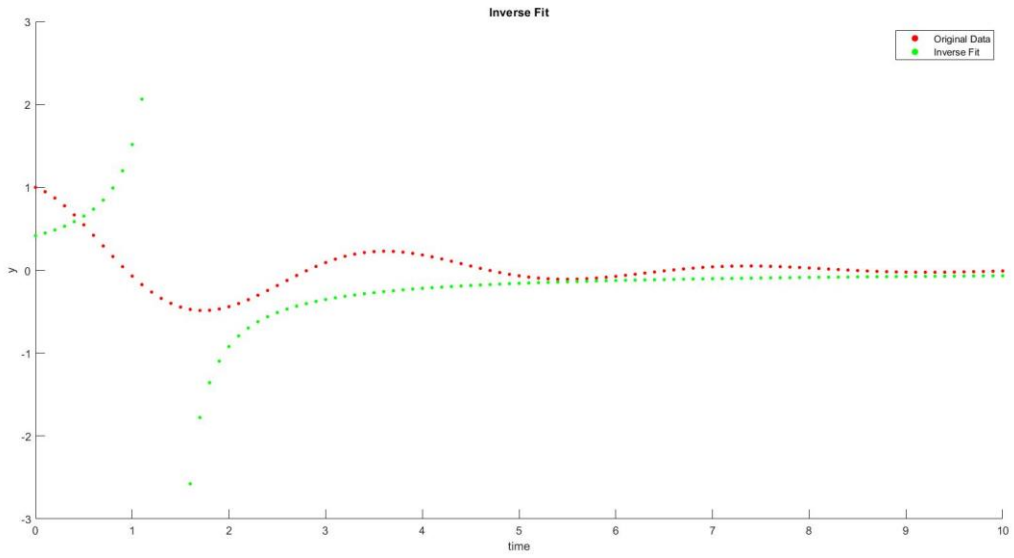

Figure 3: Inverse Fit

The square error of this regression is calculated to be $error = 743.4615$. This error is very large, which most definitely has to do with the asymptotic behavior right in the center of interest of our data. The values effectively go up to infinity, then back down to minus infinity, which throws off the error by a lot. However, it is interesting to note that as time progresses, the regression curve closely hugs the data curve, and has a transient response which is very similar to our data.

Question 2 Root Finding using the Bi-Section Method

This section of the assignment required us to find the roots of a discrete data set using the bi-section method. This method is a discretization of the Newton method, which allows it to work for discrete data. The bi-section method is a more computationally efficient method than the Newton method, simply due to the simplicity of the computations (sums, subtractions, multiplications, additions instead of derivatives).

The function for which to find the roots was given as:

$$f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27}$$

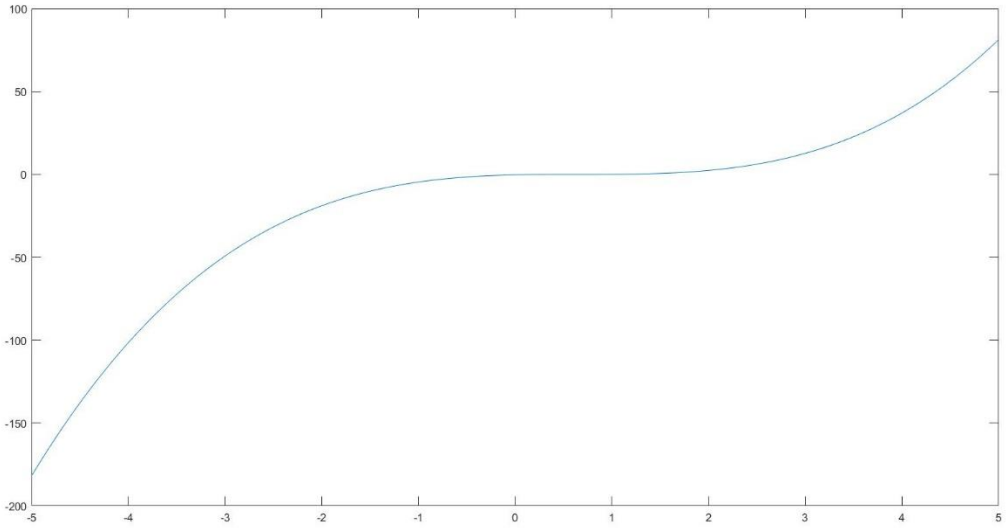

Figure 4: Function to compute roots

The bi-section method is as was explained in the class notes: We begin by having a fixed search window, in which we verify that the sign of the function differs at both endpoints. Due to the Middle Value Theorem, we assume that at least 1 root can lay within this interval. Thus, every iteration of the method slices the search area in half, until the required precision is obtained.

My strategy for this, as it is a cubic expression, that we expect three roots. Thus, the bi-section method must be carried out three times, to compute each of the three roots, which could be repeated unique or even complex. Complex roots cannot be found using the bi-section method, but repeated roots can be found using a little bit of fiddling around with the algorithm.

Having computed the roots of the function using a built in MATLAB function, I knew that the roots of the function were repeated. Since repeated roots are shared to the derivative of a function, I computed the discrete derivative of the function, and attempted to see if the root found using the bi-section method was also a root to the derivative. If it was, I would then remove the root from the function, by dividing the original function by the pole. This strategy yielded the following roots:

Root	Bi-section Method	MATLAB
1	0.666666673496366	0.666666666666667
2	0.666666673496366	0.666666666666667
3	0.666666673496366	0.666666666666667

Table 1: Roots of the function

The roots are repeated, with multiplicity 3. Using the derivative method described above, the algorithm was able to compute the repeated roots correctly. The roots fall within the tolerance value of $1 * 10^{-8}$.

Question 3 – Oscillation Frequency using the Secant Method

This section of the assignment posed a problem whose solution was not as straight forward as simply applying the formulas seen in the class slides. Basically, we were given data, and we asked to determine the oscillation frequency of this data.

My strategy was simple: knowing that an oscillating signal repeats every period and assuming the signal to be periodic, I would simply measure the distance in seconds between two peaks of the function. Then, finding the frequency of the data is trivial:

$$f = \frac{1}{T}$$

So then, the problem was posed: How do I find the exact points at which the signal is at a peak? Observing the data represented in **Figure 5** below, the points to choose seemed to be obvious.

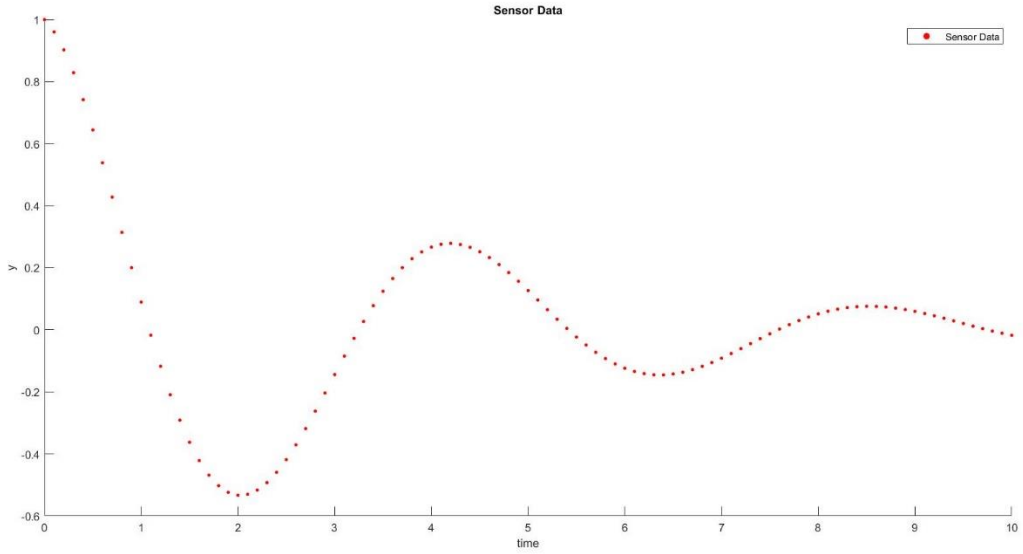

Figure 5: Question 3 sensor data

Such, I chose to measure the distance between the two negative peaks of the data, as they were well defined and fell right in the middle of the range, where the data was not too attenuated. Such I opted to look for the roots of the derivative of the data, which would be zero when the peaks were at their maximum negative value.

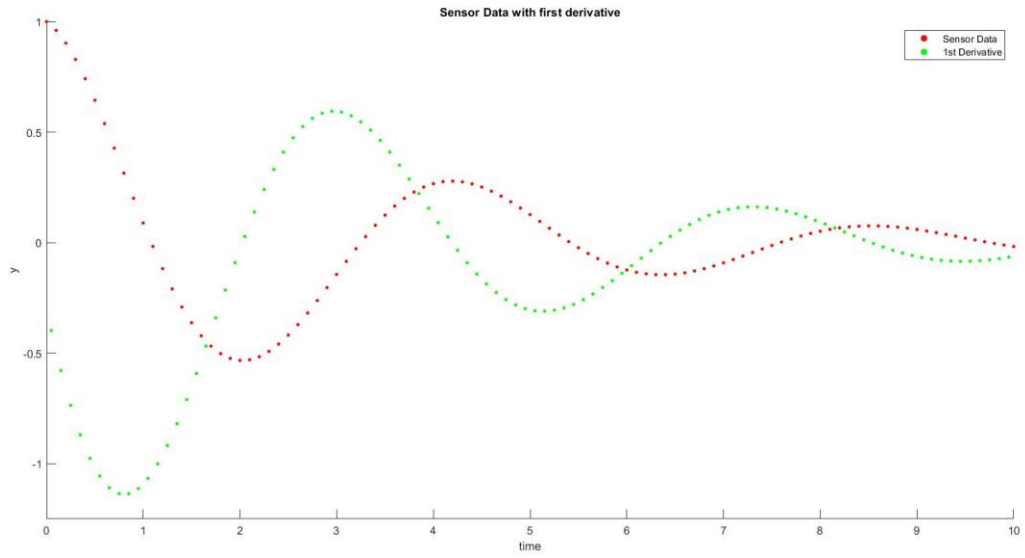

Figure 6: Question 3 data and first derivative.

Like the bi-section method, the secant method is also a discrete method of root finding based on the Newton method, such we can apply it to our discrete data. For the first root, we begin at a time $t = 1.25$ and an end time of $t = 1.55$. This yields our first root, at time $t = 2.026572010325451$. Then, using $t = 5.55$ and $t = 5.85$, we can obtain the second root at $t = 6.359337415439188$.

Therefore, the frequency of the signal is calculated in the following way:

$$f = \frac{1}{T}$$

$$f = \frac{1}{6.359337415439188 - 2.026572010325451}$$

$$f = 0.230799479431716\ Hz$$

Question 4 – Function Intersection

This section of the assignment required us to find the intersection of two lines in order to determine the operating frequencies at which both curves meet and both relationships are satisfied. The question requires the use of the Newton method, as well as having a precision of $1 * 10^{-8}$.

The equations for the curves are as follows, and they can be seen graphed together in **Figure 7** below:

$$P(w) = 100(1 - e^{-0.56*w})$$

$$E(w) = w^2 - 5t + 30$$

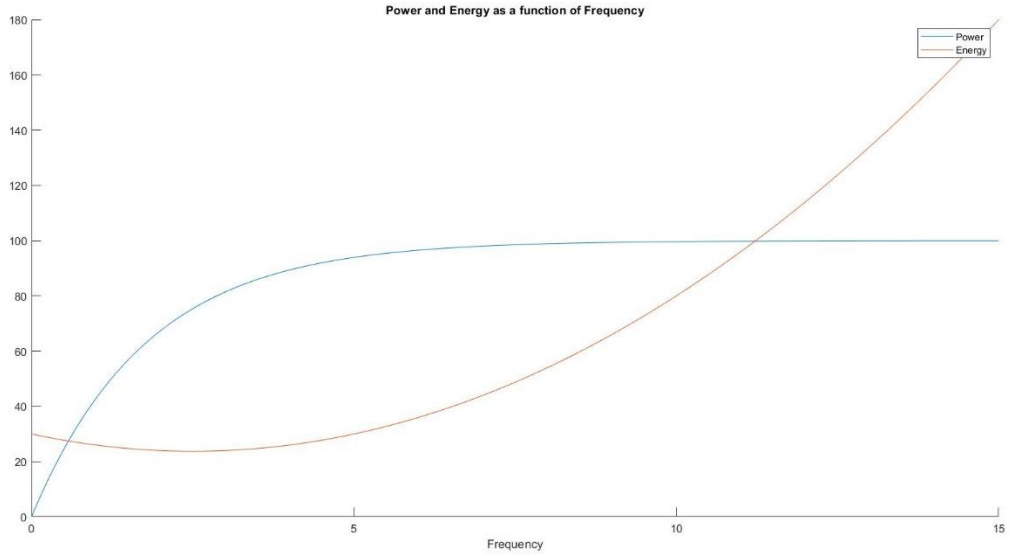

Figure 7: Power and Energy functions for question 4

The method for this problem is simple, as the Newton method can be used with symbolic MATLAB functions. All we must do is program the Newton method algorithm:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Then, we must subtract the signals from each other, such that:

$$sub(w) = E(w) - P(w)$$

This allows us to set the equation to zero and find the roots, which is where the two lines will intersect and both relationships agree. The Newton method is ideal in this case, as we have access to the formula of both curves.

Thus, starting with the frequency value $w = 0$, we obtain the first root at $w = 0.573322845282937\ kHz$. Then, moving our starting frequency along to X = 10, we can then find the next root at $w = 11.222086510495158\ kHz$. As can be seen with the amount of precision of the computation, my results are precise up to $1 * 10^{-8}$.

Appendix

Please see the attached README.txt file to find the MATLAB Live-Script file, as well as the functions used to compute the results in this report.