Alex Merino

MTH 4311

Dr. Pei Liu

Due: 2/27/2025

Part 1:

We are given two functions $f_a(t)$ and $f_b(t)$. We are asked to find a polynomial of degree N that fits the data points of $(t_i, f(t_i))$ for M datapoints. In our case, M != N. As shown below, we can rewrite the polynomial of degree N into matrix vector form:

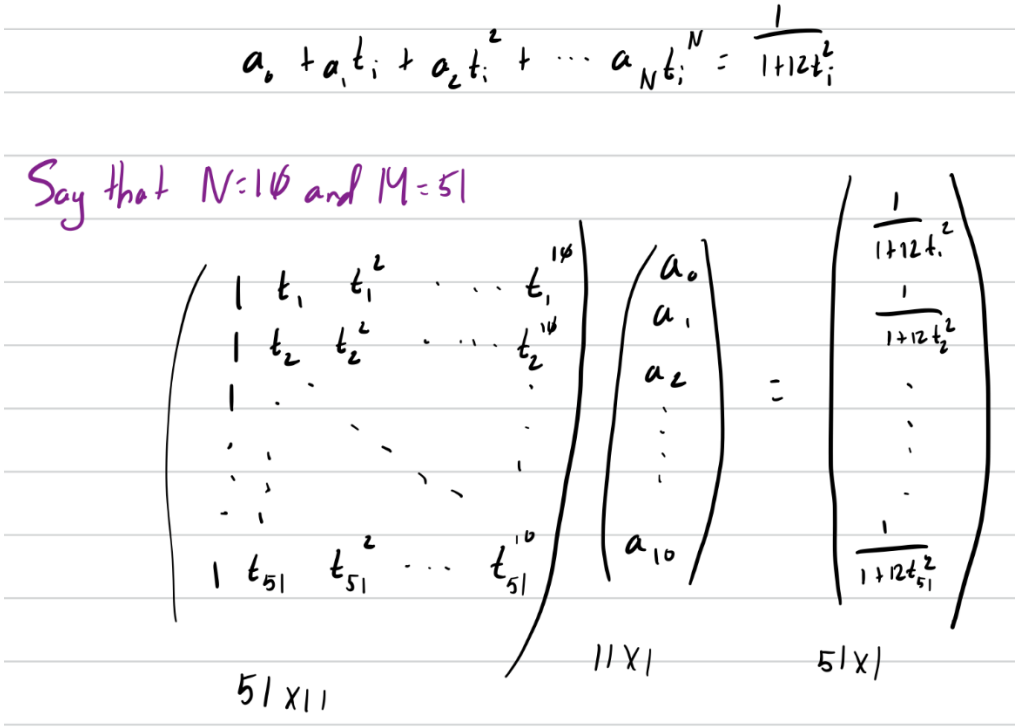
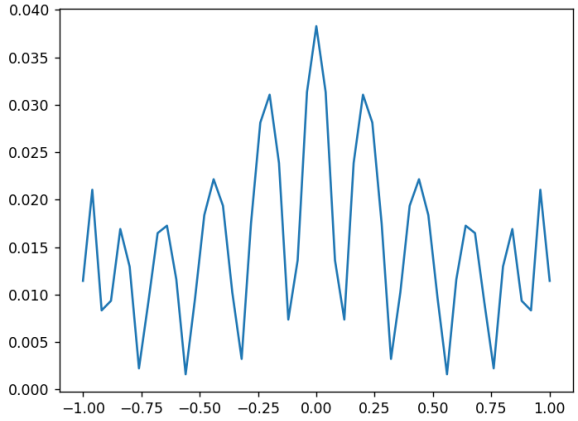


Figure 1: The mathematical work for putting a polynomial into matrix-vector form

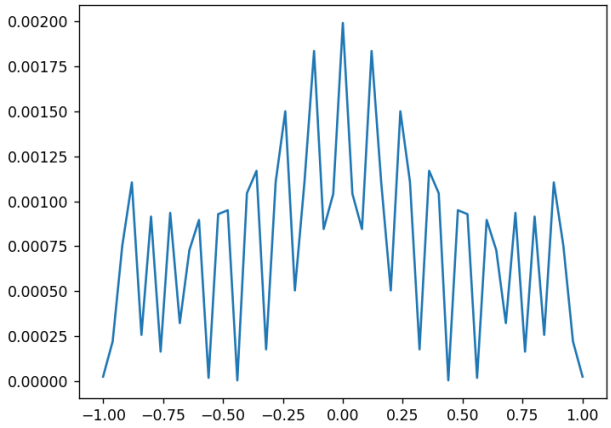
This is in the form of Ax=b where A consists of t_i values, b consists of the y-values of the function $f(t_i)$ at t_i and x consists of the coefficients of the interpolating polynomial. As shown, the matrix A is not square, so we must solve the system of equations using least-squares solution. This will solve for our coefficient vector x. Once, we receive the coefficients we can use them as our polynomial function (polyval) and get our approximation of each function.

For function $f_a(t)= 1/(1 + 12t^2)$, These are the plots of the error between the true points and the interpolating points for N = [10, 20, 30, 40, 50] with M=51 points. The expected errors of the

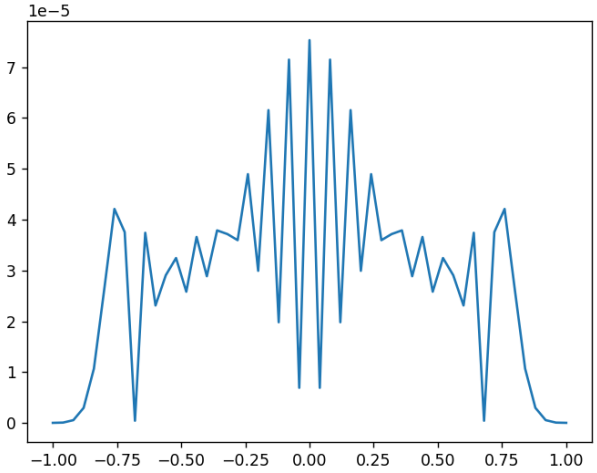
function should decrease with increasing N since the difference in N and M will be smaller and the least-squares solutions to be more accurate.



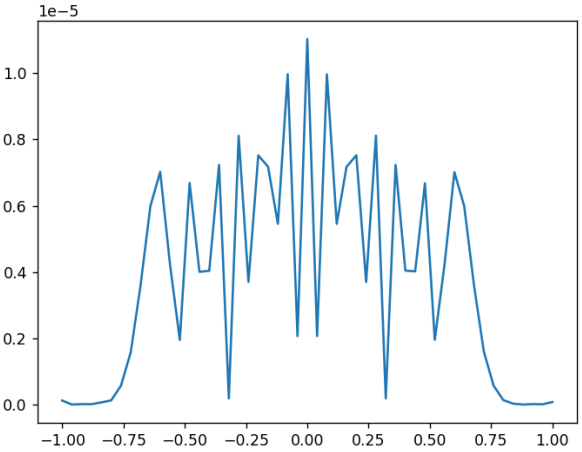
N=10



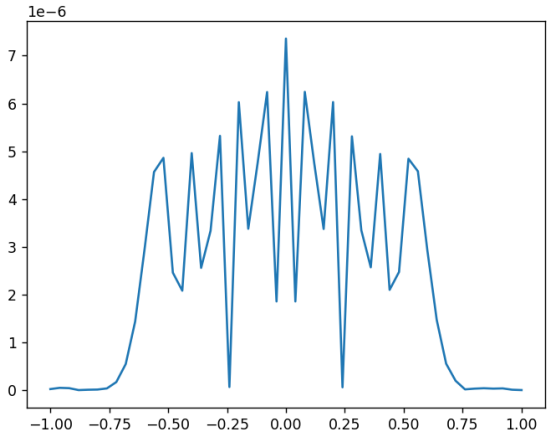
N=20



N=30

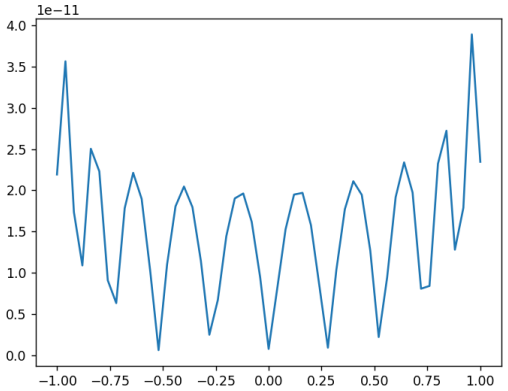


N=40

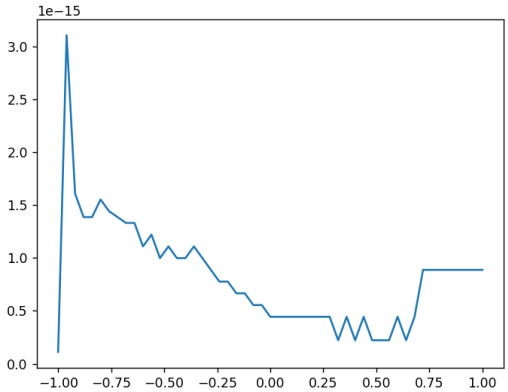


N=50

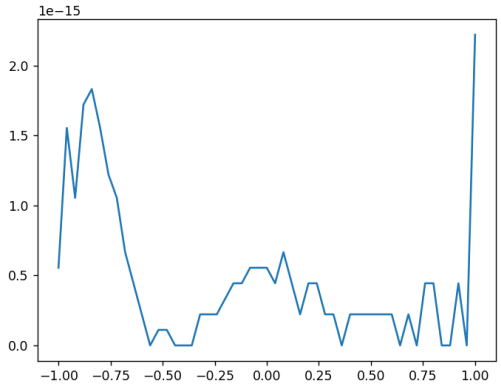
When plotting the true graphs, near the ends of the graph when N=10 and N = 20, there is some oscillatory motion in the line, which is why the values are higher closer to the ends compared to higher values of N. When we increase N the error decreases as shown by the scale each graph has in the top left of each image. This makes sense that the error decreases, because the higher the degree of the polynomial means we can fit the data better with our extra degrees of freedom.



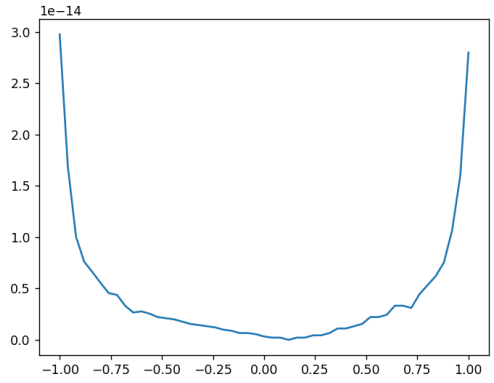
N=10



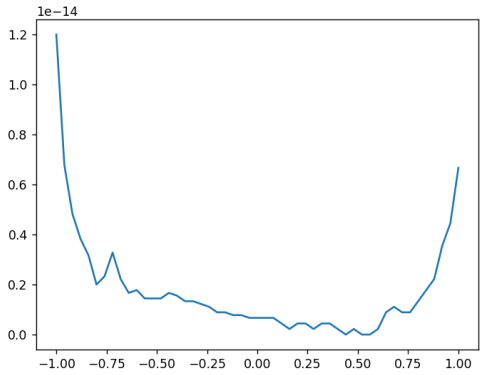
N=20



N=30



N=40

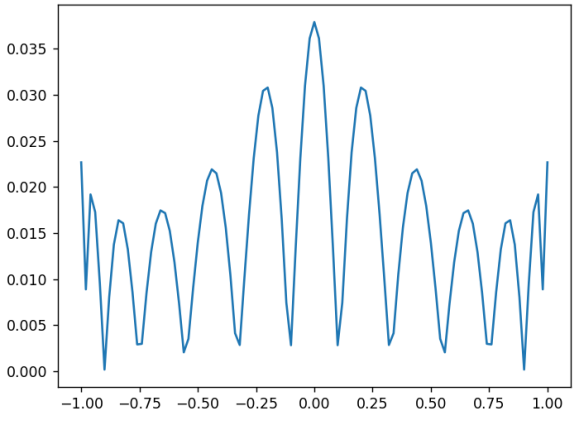


N=50

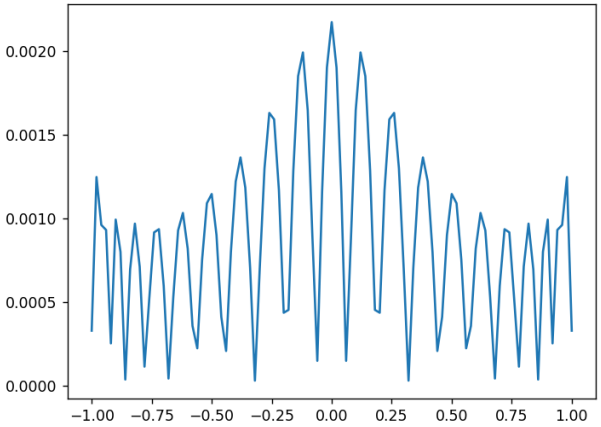
These graphs are the plots for the error of the second function $f(t_i) = e^t$. To get the polynomial coefficients for this equation we take the same steps as in **Figure** 1, but the vector b is the value of e^t at each t. The error graphs do get smaller in magnitude, which makes since because we are increasing our degrees of freedom. One major difference between the error graphs is that the ends of this plot are the highest in magnitude.

Part 1b:

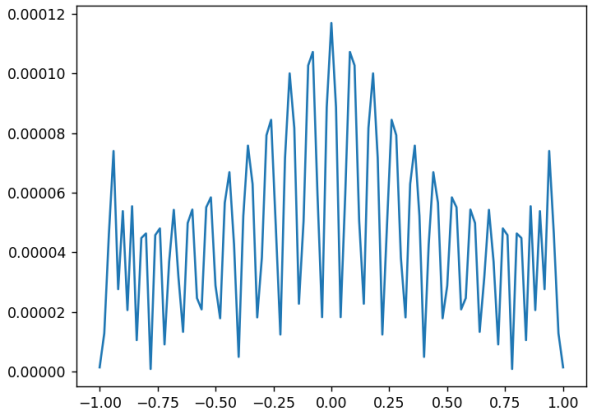
Instead of M=51 points, we will now use M=101 datapoints. This means our system of linear equations will have a matrix of 101xN, which is almost twice the size of the previous system. N will now be [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]



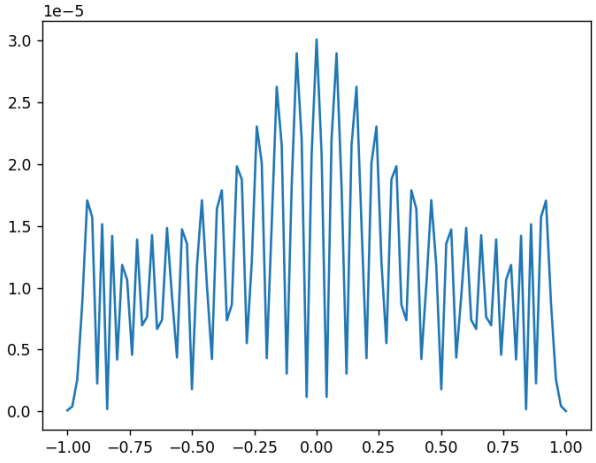
N=10



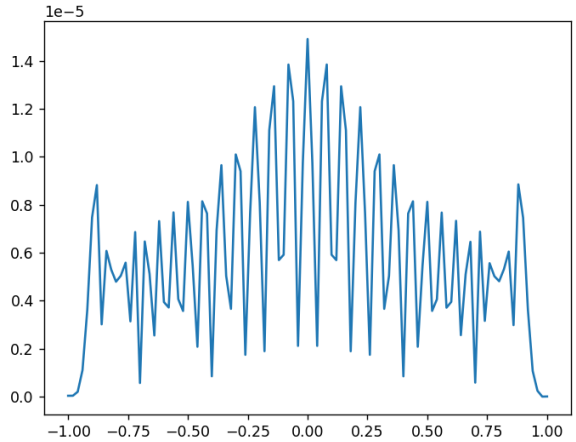
N=20



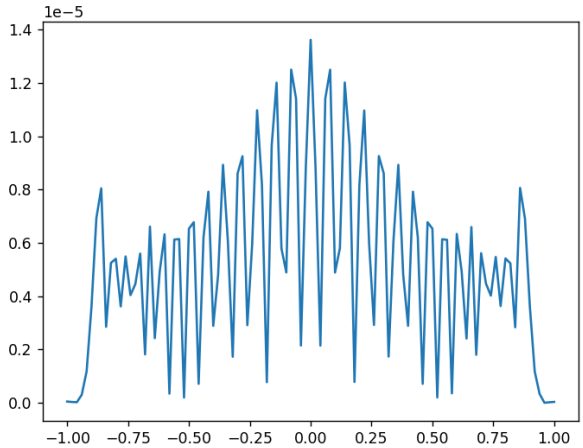
N=30



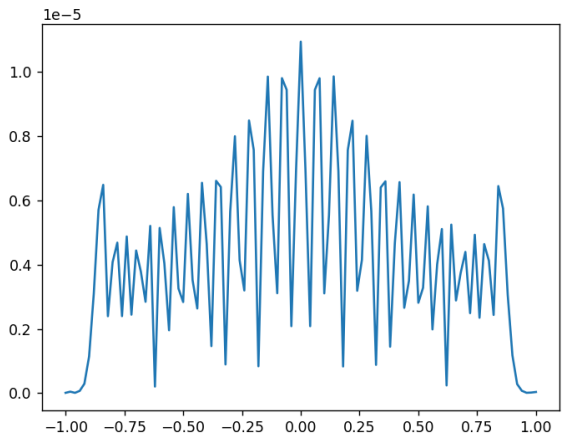
N=40



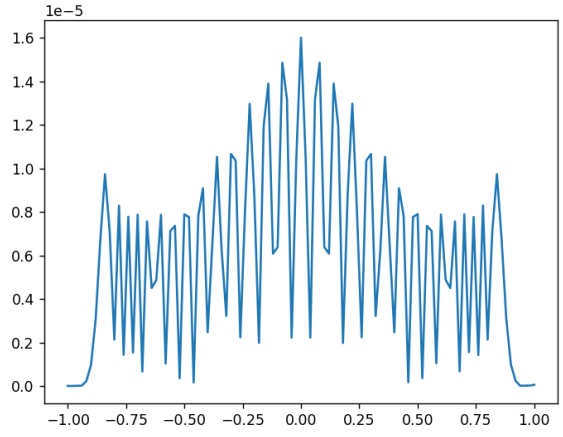
N=50



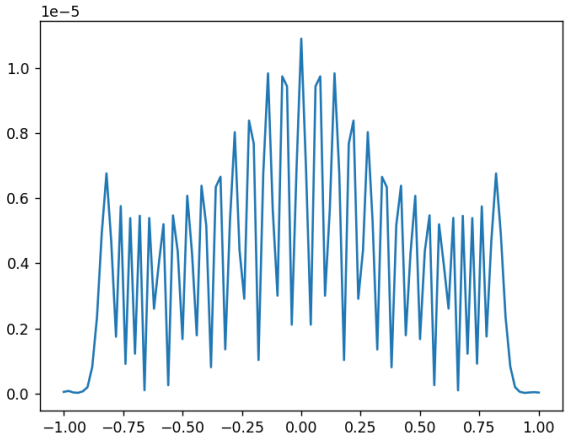
N=60



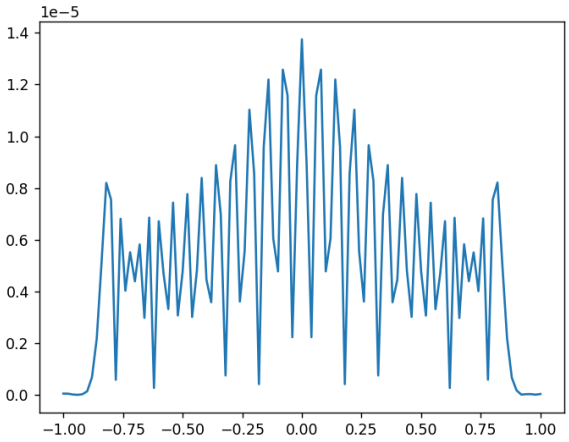
N=70



N=80

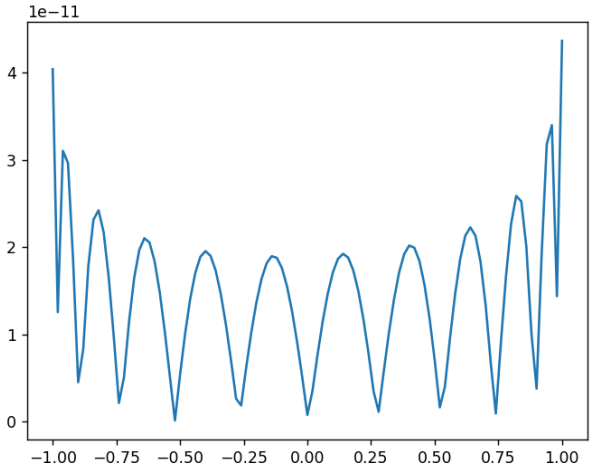


N=90

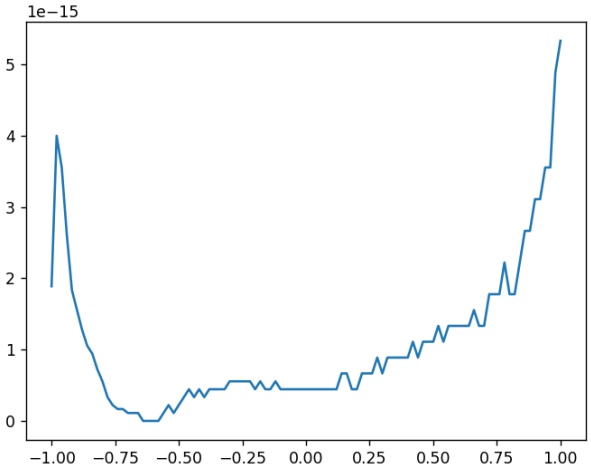


N=100

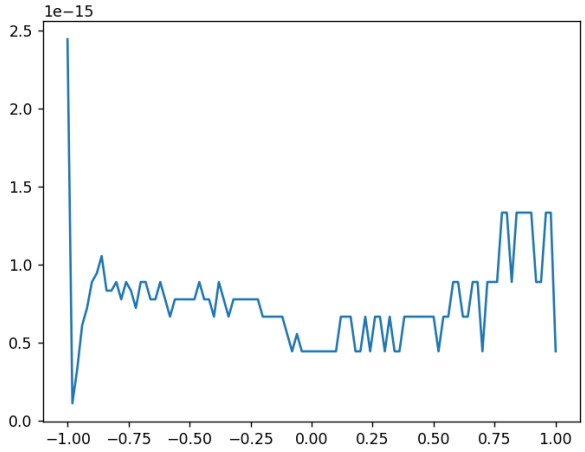
The ten error plots shown above are for all the N values. The error in the plots does not get better when you increase the amount of data points or increase the degree of the polynomial. This is because of the overfitting that is occurring with the increasing number of N values. When we had lower N values, the error was slightly decreased, but once it got to around $1.5 * 10^{-5}$ the error did not continue to decrease.



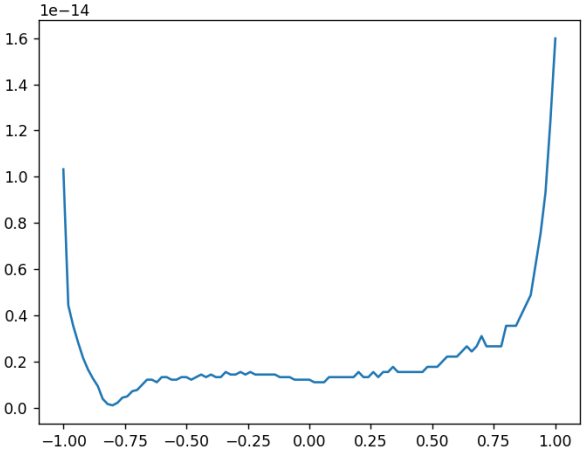
N=10



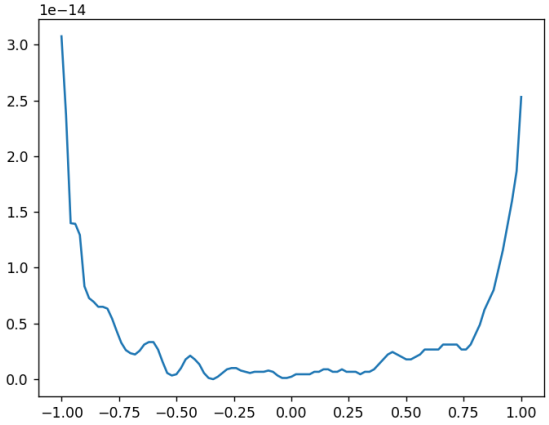
N=20



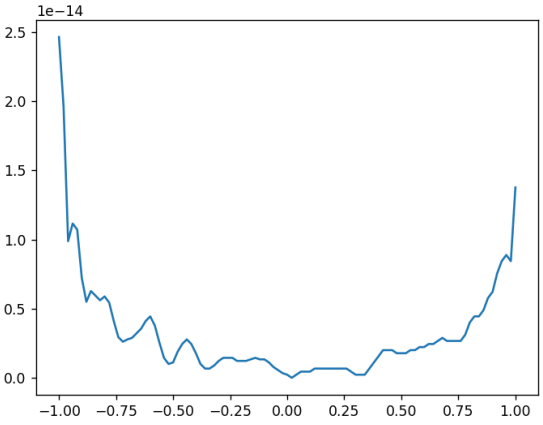
N=30



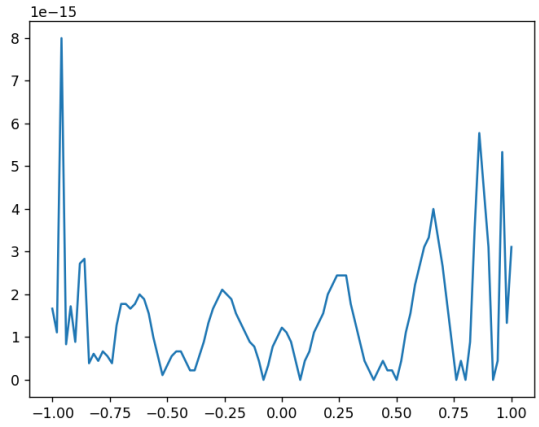
N=40



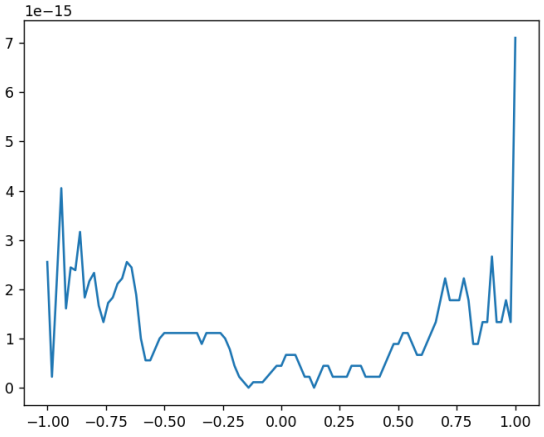
N=50



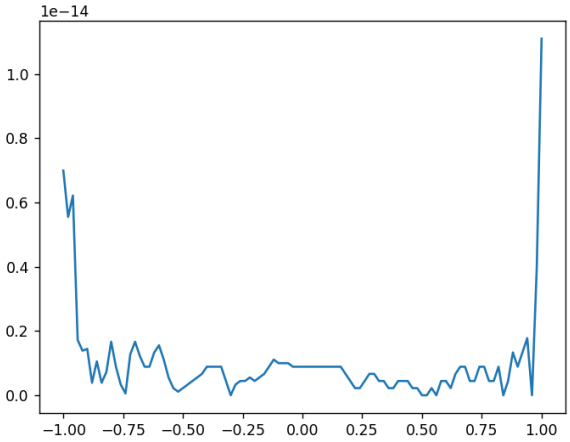
N=60



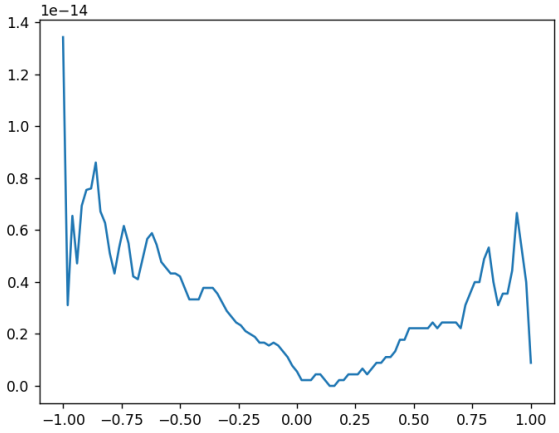
N=70



N=80



N=90

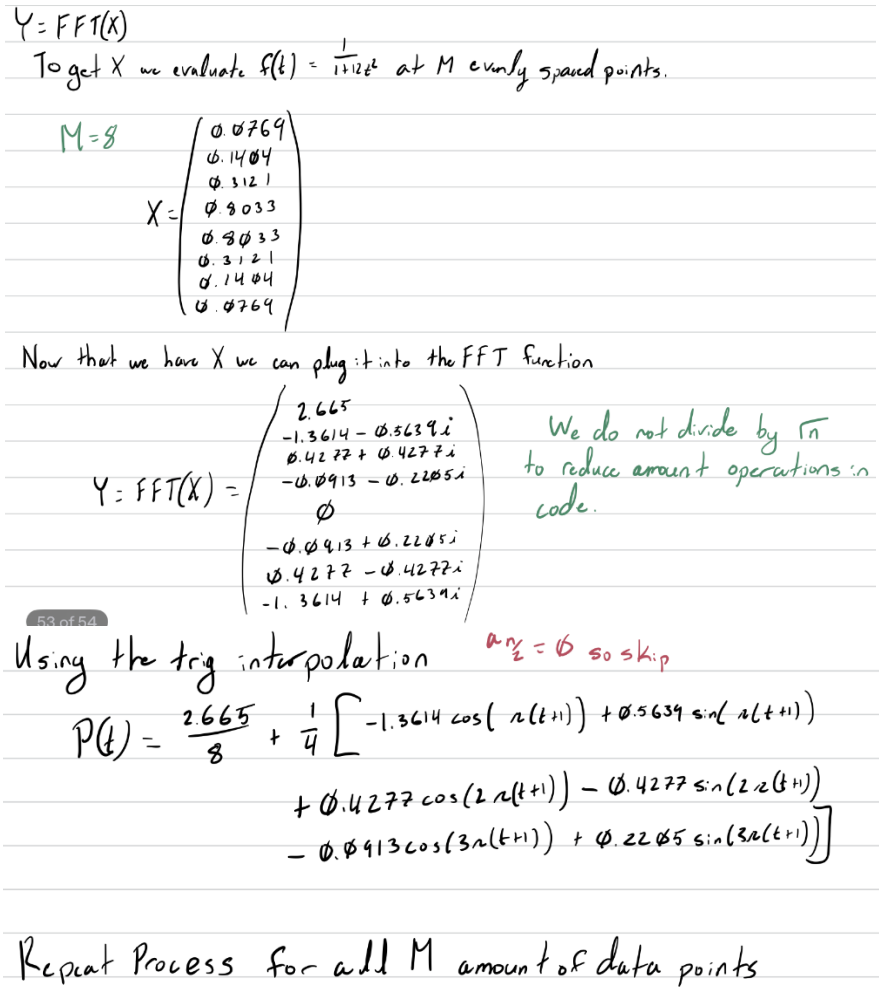


N=100

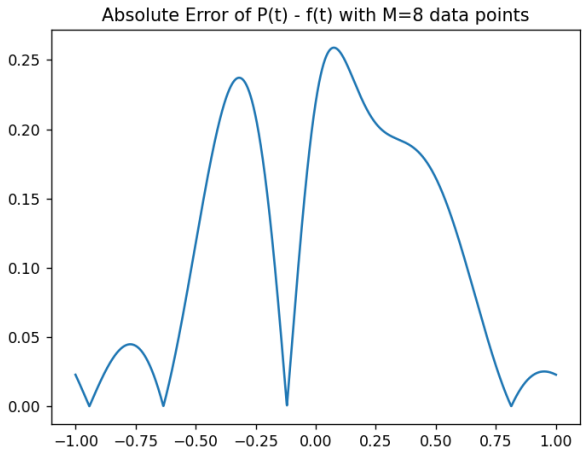
With the second function, e^t, the errors did not tend to decrease either. Again this is probably due to overfitting the actual data with large degree polynomials. The end points are still with the largest error to help fit the center of the data better, this was also seen in other interpolations such as Chebyshev.

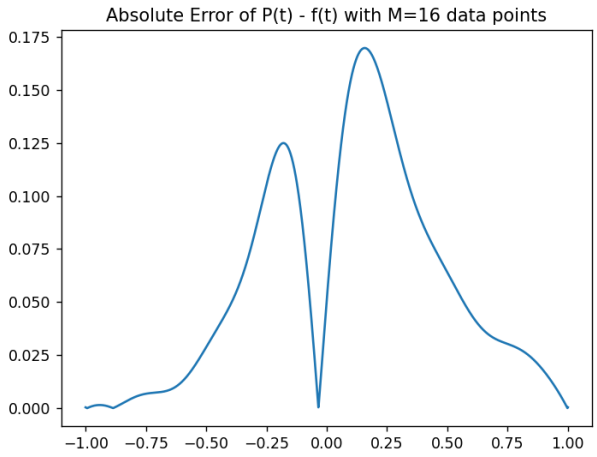
Part 2:

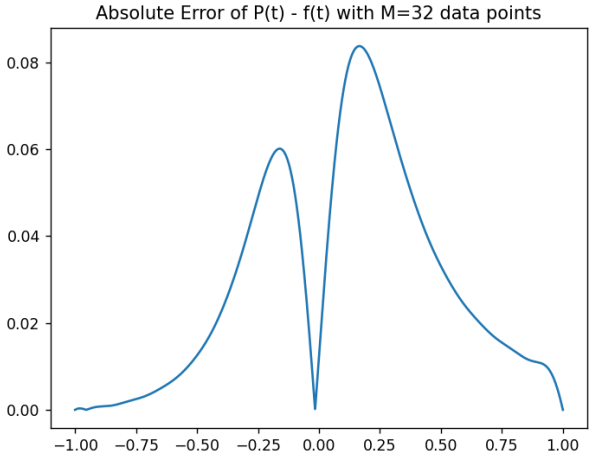
To find the trigonometric interpolation of the function f(t) we need to create a trigonometric function P(t). This function requires the coefficients of the FFT of the t values we want to approximate with. Below is a worked out example of what we want to do:

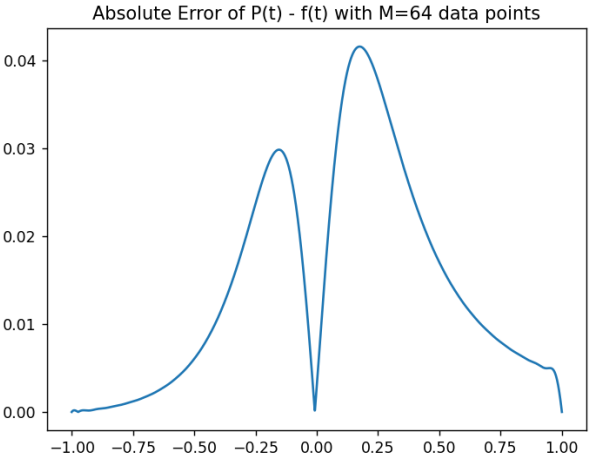


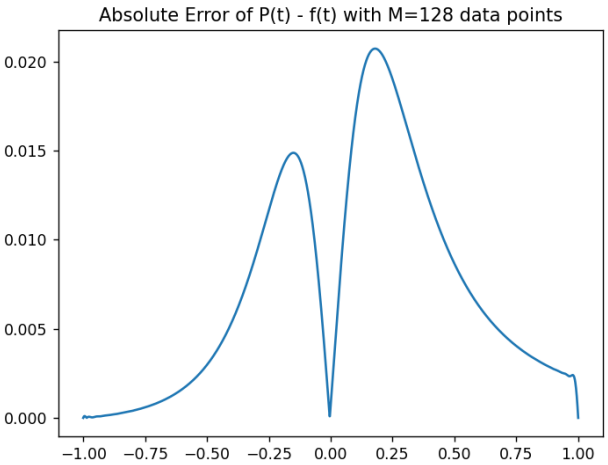
After implementing this in code, the following error graphs are output as a result:

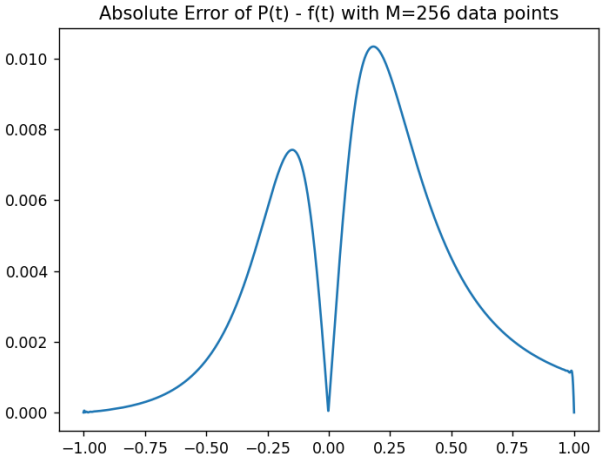






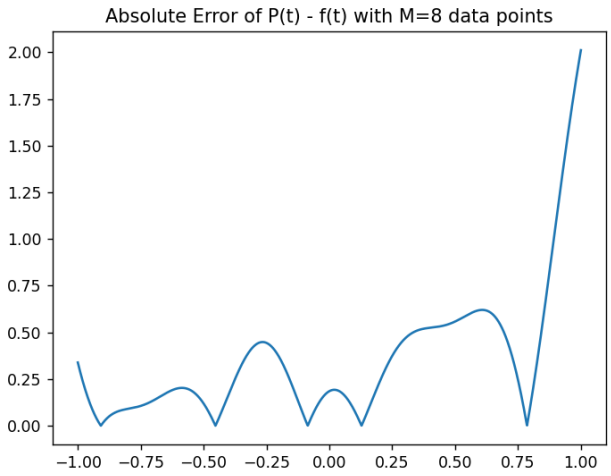


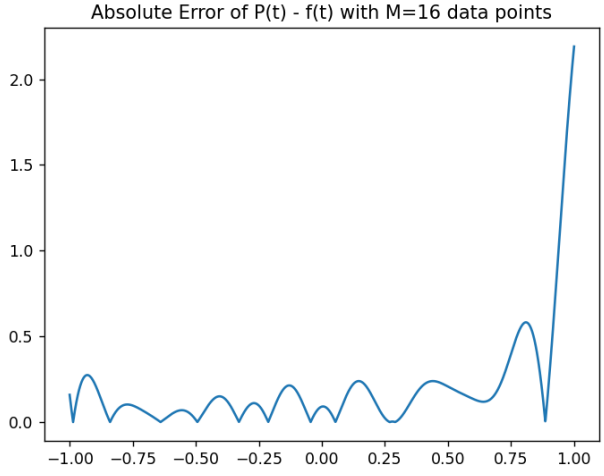


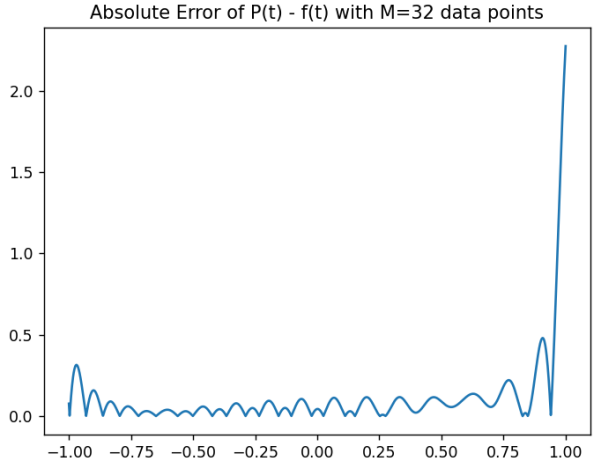


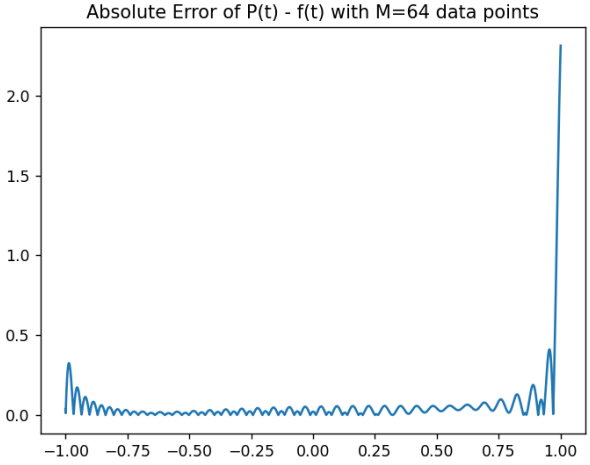
As we increase the amount of data points we would like to use in the FFT the error between the function decreases. This decrease makes sense because the more information we have about our function, the larger amount of trigonometric functions we can use. This is ideal for interpolation compared to polynomial interpolation because the more trig functions that are added, the complexity/degree of the function does not increase.

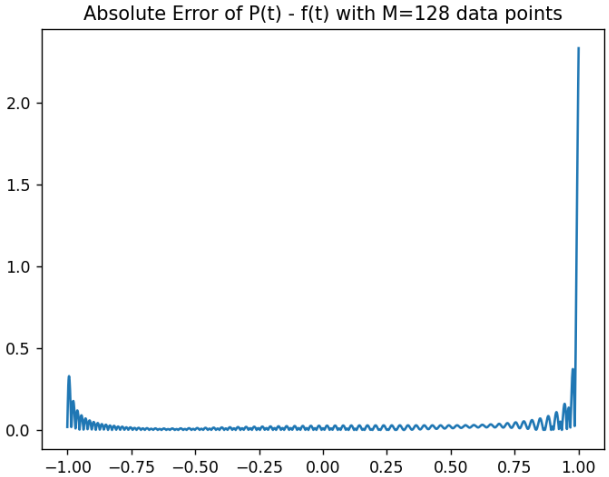
For the function e^t, the error plots are show below:

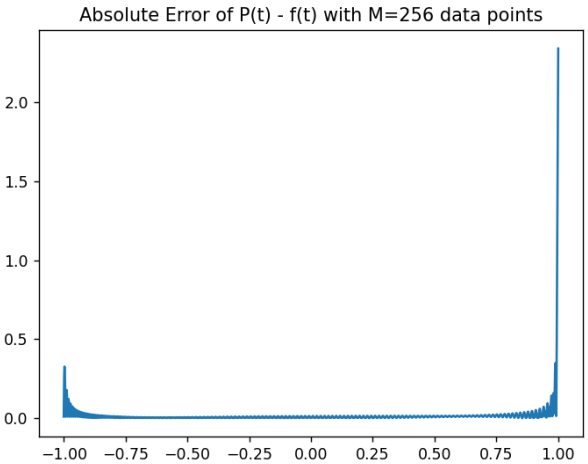








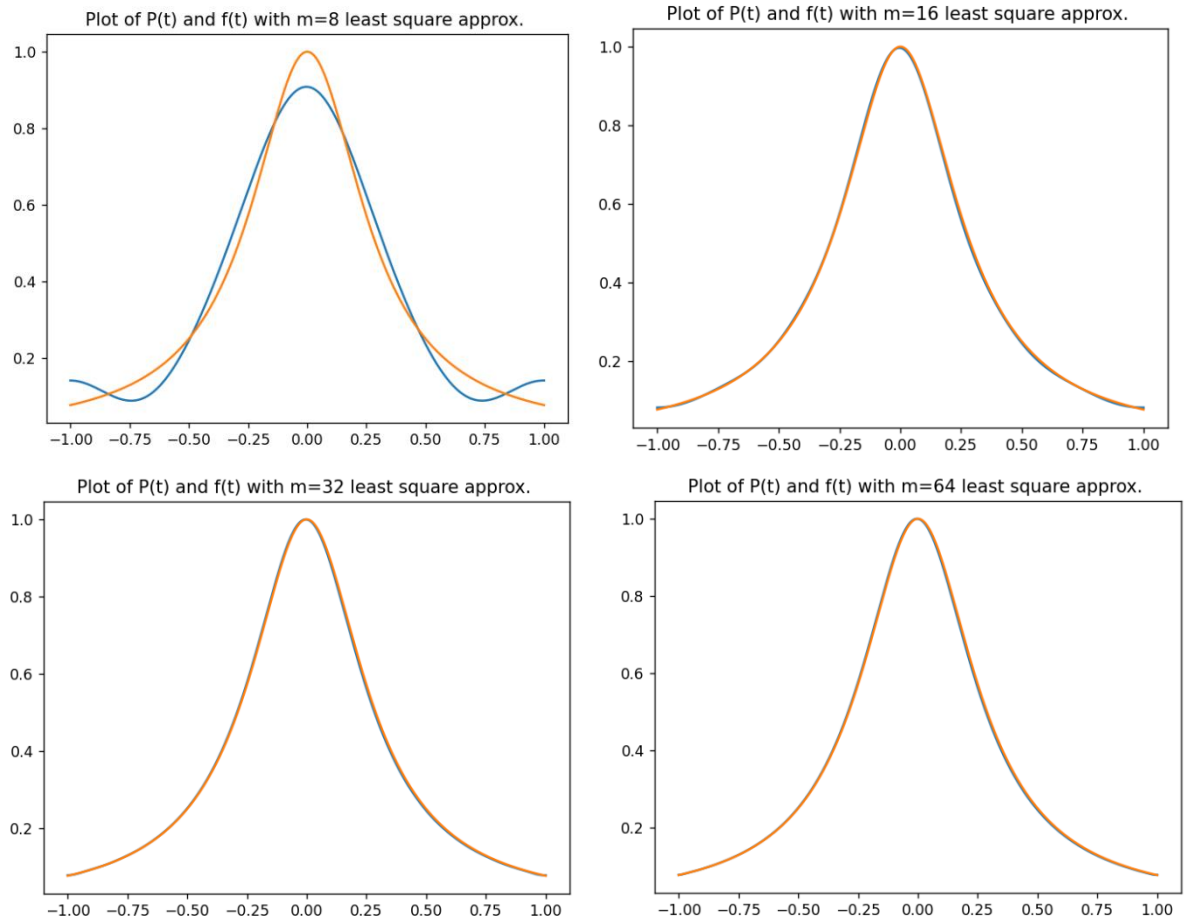


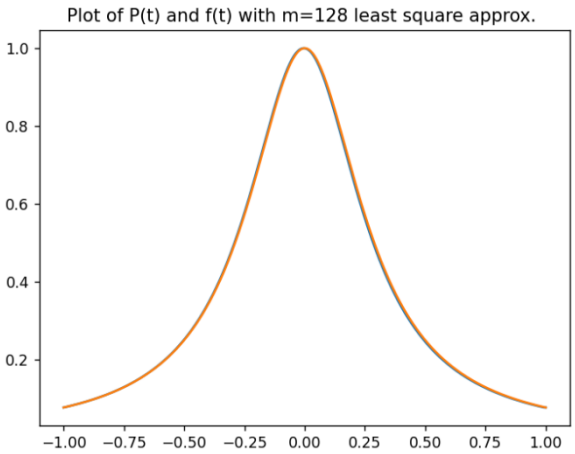


It is known that e^t is not an oscillatory function. It is always increasing, which means to trigonometrically interpolate the function, there needs to be larger error and the ends of the function. At fewer points the trigonometric interpolation does very poorly, but with more points, the middle part of the curve is approximated very closely, still with oscillatory movement though. At the ends the function makes a large jump to restart the period, this type of error can also be seen in Chebyshev interpolation.

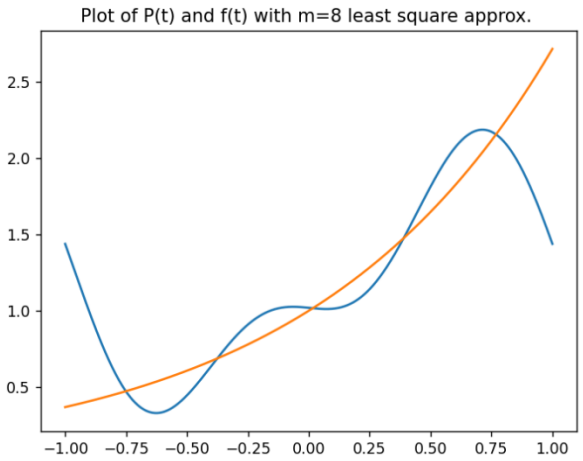
Part 3:

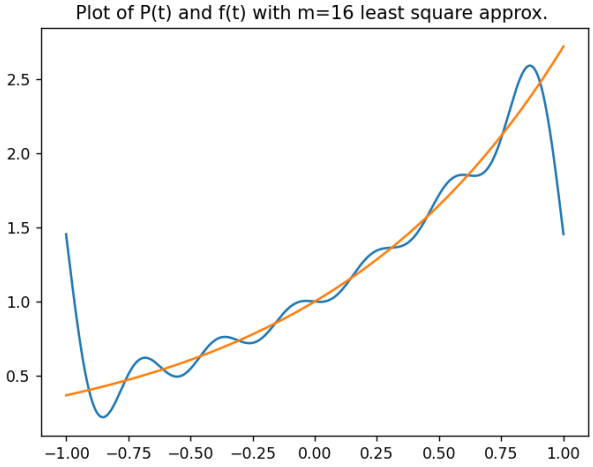
To do least squares approximation, the only difference in working out the problem is that any n in the summation and in the initial cosine term will change to m as in slide 40 of the FFT powerpoint from class. I made a small modification in the trigonometric interpolation code to allow for an optional input m, which when changed from above 0, it will do the least squares approximation with the first m terms. Below are the errors for function $f_a(t)$. For all plots, the true function is in orange and the approximated function is in blue.

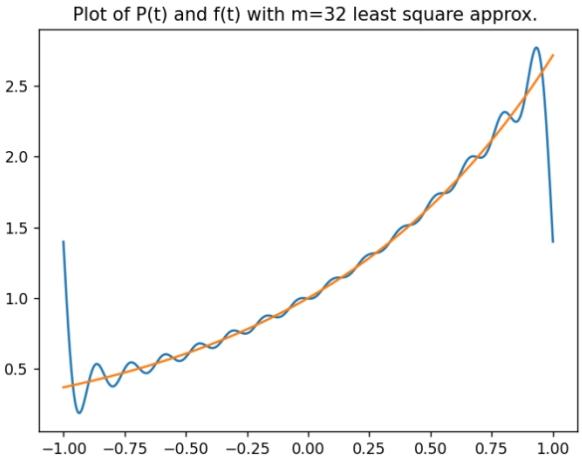


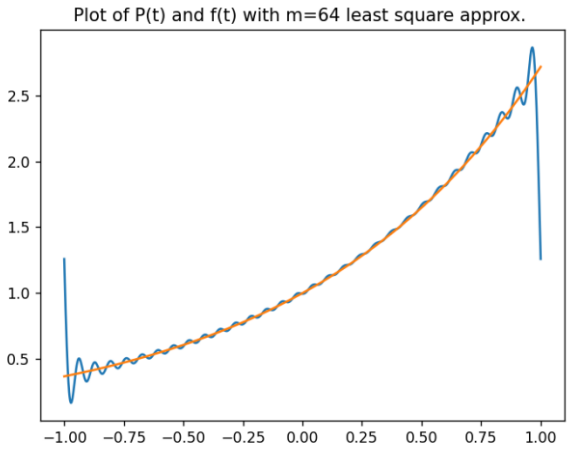


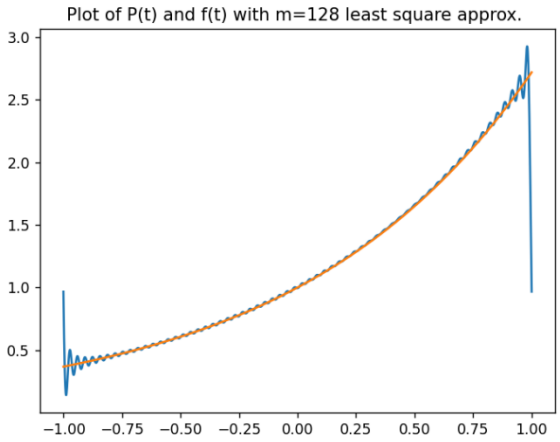
As we can tell by viewing the graphs, as the m value goes up, the approximation goes closer to the true value. This makes sense because with the more trigonometric terms we have the more robust and accurate the approximation can be.











The function, e^t, as we mentioned above is not an oscillatory function which means that the approximations will never be close to perfect. But in these graphs we can see how the function approximates it and the size of the oscillations are smaller with the larger amount of m values we have. This follows the same logic as the other function with adding more trigonometric terms.