DDCS CW Report

Euan Bagguley

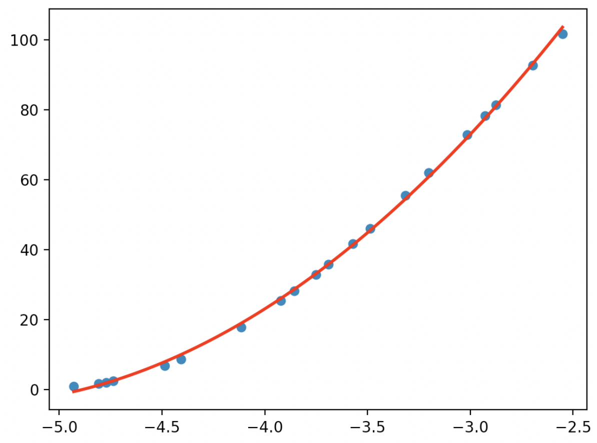
April 2021

1 Equations for linear regression

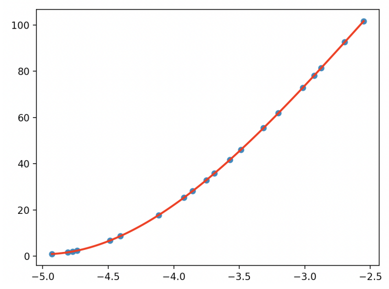
The equations I used for the linear regression were the least squares and squared error equations. The least squares takes in a matrix of extended x values (in the case of linear this consisted of a column of ones and a column of x values) and an array of y values. It returns an array of coefficients which are then used for the bias on the linear regression and the gradient of the line. I also used the squared error equation which takes in the predicted y and original y arrays from a segment and returns the summed, squared difference between each predicted and original y value, to calculate the error for each segment.

2 Justification for choice of polynomial order

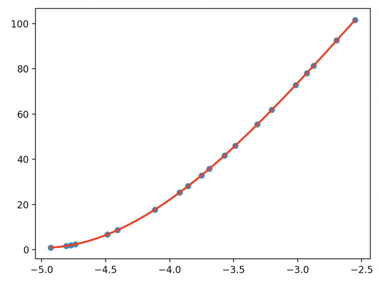
To decide the order of the polynomial function, I started by comparing polynomial plots of order 2 through to 5, with coefficients obtained from the least squares function for each order, on the file 'basic_3.csv'. When looking at the plots, as seen in Figure 1 below, it was evident that the order 2 polynomial was the worst fit, however the difference in the remaining orders was negligible. This was reinforced by the order 2 polynomial having the greatest summed squared error of 15.74 while orders 3 to 5 were very close with errors of 4.40-19, 1.42e-13, 7.15e-09 respectively. As the order 3 polynomial had the smallest summed squared error, this was my choice for the polynomial function as, by having the lowest order out of the three, there is no chance that the reason for it acquiring the lowest error be due to any overfitting. My decision was also backed up by testing the four functions on basic 4 where, again, the order 3 polynomial had the least squared error. To confirm my choice, later in the process, I ran k-fold cross-validation on the four different orders on a variety of the data set segments to decide which order had the lowest average error and the outcome was that the order 3 polynomial was always picked.



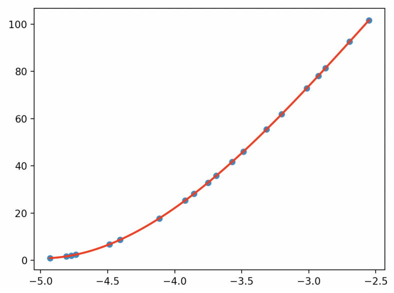
(a) Order 2 polynomial



(b) Order 3 polynomial



(c) Order 4 polynomial

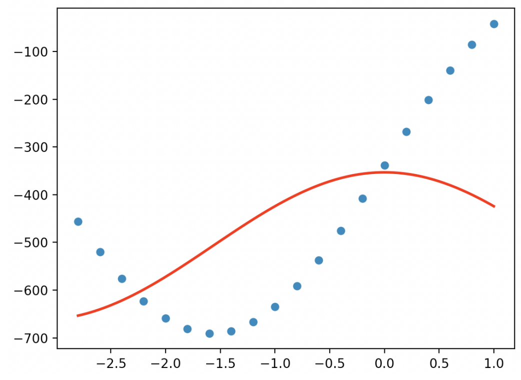


(d) Order 5 polynomial

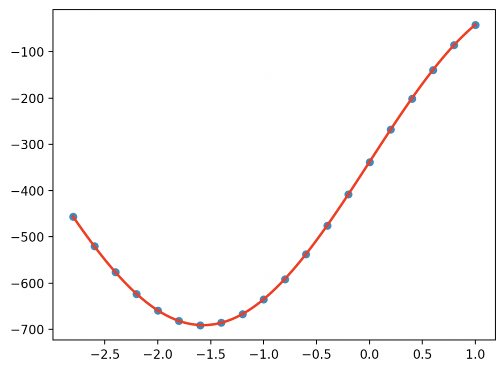
Figure 1: Tested polynomial plots on input ‘basic_3.csv’

3 Justification for choice of the unknown function

When trying my polynomial on the ‘basic_5.csv’ file, there was a lot of error and so I gathered that this was the unknown function. Looking at the plotted data points, it was clear that they had the form of a wave. To tailor to this, I first tried fitting a cosine wave with a coefficient and an added bias. As shown in Figure 2a, this was clearly not correct and looked to be half a period out of phase. I then changed it so it was a fitted sine wave with a bias and coefficient and this fitted perfectly (shown in Figure 2b). I then tested the sine plot on segments in the ‘noise’ and ‘adv’ files, to which it also fitted the data points with little error, justifying my choice.



(a) cosine wave Plot



(b) sine wave plot

Figure 2: Attempted plots to find the unknown function on ‘basic_5.csv’

4 Description of procedure of selecting between the linear/ polynomial/sine functions

To decide which line was rightly fitted for each segment, I implemented a function to use cross-validation in order to remove the over-fitting that occurred when only using the summed squared error to base a decision from. The function takes in the 20 data points for each segment and splits it into a testing and training set by removing 5 data points at random from the original array and adding them to a separate testing array. It then calculates the coefficients for each of the three possible lines, using the least squares method on the training array, and returns the summed squared error between the testing data points and their predicted values for all three. I then used k-fold cross-validation to repeat the procedure 100 times, returning the average summed cross-validation error of each of the three models on the line segment and used these three values to base the decision for the correct line for that segment off. In the deciding process I further added a bias towards the linear model by dividing the summed linear cross-validation error by 1.1 so that when the errors between the three different line choices were close, the program would favour the linear model, even if the error for the sine or cubic model were slightly less.
Randomly splitting the data values into a training group of 5 data points and a training group of 15 data points for cross-validation was optimal. When the training set was larger, the program would overfit more often. When the training set was smaller, the opposite would sometimes happen depending on which points selected for each set. For example, on the adv1 file, if the majority of the points selected for testing were located in the middle of the array, the segment would get predicted as a linear function. Making the split random each time ensured that each equation would be tested with a variety of different testing and training sets, making it obtain a more

reliable average cross-validation error.
The advantage of using k-fold cross-validation method to get an average of the errors meant that the program was more accurate in choosing the correct model for each segment. Before implementing this, the decision was based off the error from one cross-validation. This worked most of the time, however, depending on which 5 data points were selected for the test group, the program would sometimes overfit the regression on the linear segments and select the sine or polynomial function as, even while using cross-validation, the error that the function calculated on these segments was very close.
Adding a bias towards linear in the decision process aided the program to make the correct decision when dealing with noisy segments that were linear. Even with k-fold cross-validation, often on the first segment of 'adv_3.csv' file, the program would often overfit as the error between linear, sine and polynomial were very close. Adding a standard bias (subtracting a certain value from the linear error) did not work as it made the program chose linear on the polynomial segment in the 'basic_4.csv' file where there was little curve and no noise and so the errors between the two models were very low. Dividing the linear error by 1.2 avoids fitting linear regression to polynomial and sine functions, as proportionally the error for linear in these sections is always greater, while ensuring that linear regression is chosen over sine and polynomial models in segments were there is a lot of noise and the errors can be very similar. The advantage of this is that the program is less likely to overfit when predicting the regression for linear segments and acquires more reliable and consistent results. The downside is that if there were a segment that was polynomial or sine, but very close to a straight line, the program may fit a linear regression instead of a polynomial or sine regression. However, if this were the case, it would not matter as the segment would have to be extremely close to being a linear for this to happen and so a linear regression would still be a good estimated model for the segment.