# An Unknown Signal coursework report - Karolina Kadzielawa (zo19826)

## Introduction

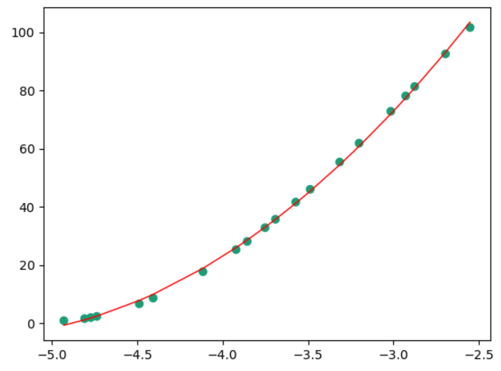
The aim of this report is to present the results of solution to the An Unknown Signal coursework, as well as outline the methodology used to fit the linear, polynomial and unknown function and describe measures taken to prevent overfitting.

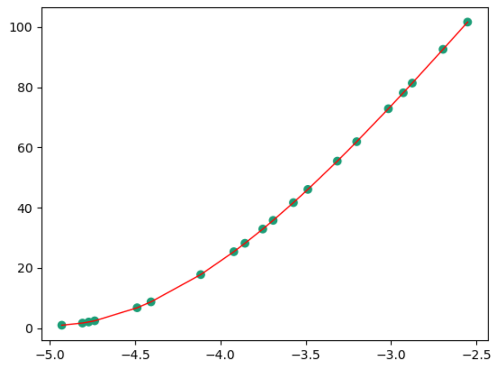
## The program structure

Given the file name as the first argument in the command line, the program reads it in using the `load_points_from_file` function provided in the `utilities.py` file. It then iterates over all the segments in the file, calculating the coefficients of the best fitting linear, polynomial and sinusoidal function using linear least squares regression. Then the cross validation error is calculated for each of those models and the function type with the lowest corresponding value is chosen for the segment. Having ranged over all of the segments the sum square error between the estimate and the actual dataset is calculated and displayed. If provided with an additional `--plot` argument the program outputs the plot of the fitting using the `view_data_segments` function from the `utilities.py` file.

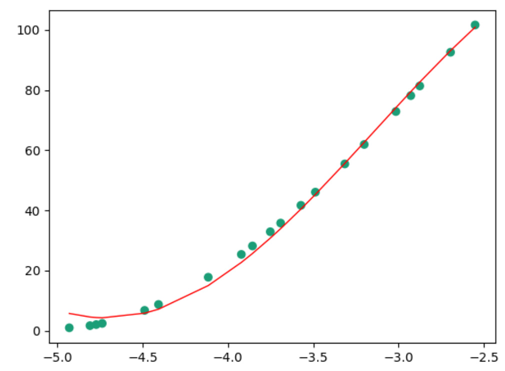
## Finding the degree of the polynomial function

Initially the solution could fit only a linear function: $y = a * x + b$. As a quadratic function $y = a * x^2 + b * x + c$ turned out to be a poor fit on the few segments it was chosen over linear on, I extended it to a cubic one $y = a * x^3 + b * x^2 + c * x + d$. Below is a comparison between the two, where the cubic one yields a minimal error.





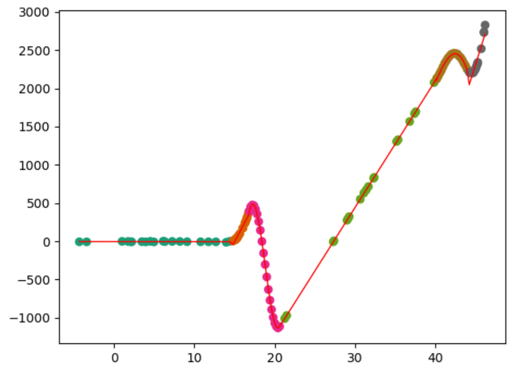
I have also tried using a quartic function, but the cubic function proved to be a better fit for example on the `basic` files, which contain less noise, meaning that an accurate fit is to be expected.

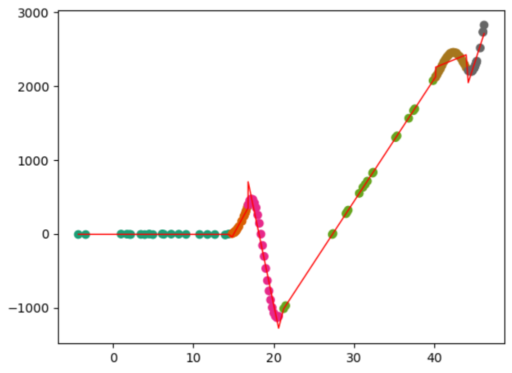


A few segments remained rather poorly reconstructed, however that was prior to finding the unknown function. I decided to move on to determining the unknown function to be able to assess whether the segments were in fact meant to be reconstructed by it or my choice of the degree of the polynomial function had been wrong. Having done that I was reassured in my choice.

## Finding the unknown function

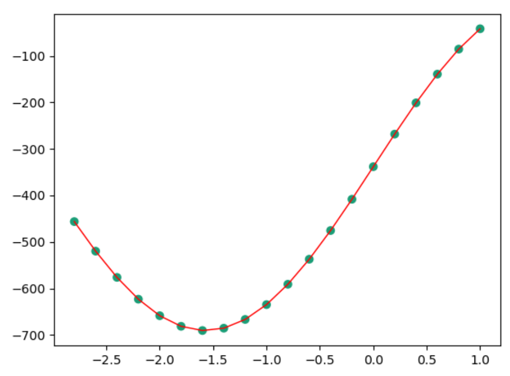
Having extended the program to fit a polynomial function, I moved to finding the unknown function. I made guesses of the function type based on my visual assessment of the shape of the available plots. I added print statements to identify the function type selected for each segment. The first guess was an exponential one: $y = a * e^x + b$. However, none of the segments in any of the test files matched the function type. What is more, the program started favouring the linear function more frequently than before. Below is an example of the difference between the version of the program which could choose only between linear and polynomial function types and one fitting all three.

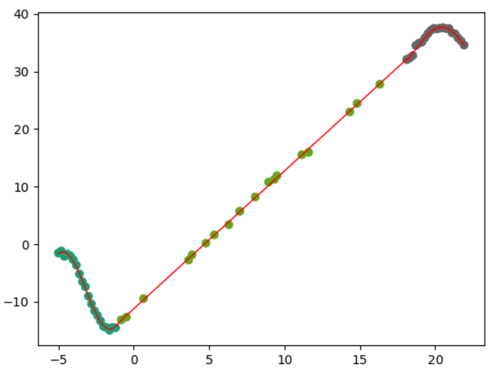


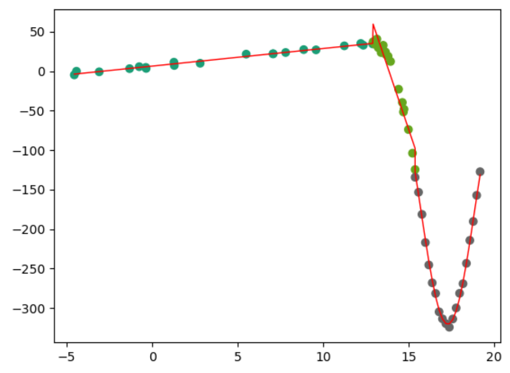


I have not been able to identify the cause of this unexpected behaviour.
The second guess of the unknown function type was a sinusoidal one: $y = a * sin(x) + b$. This time the thirst option was being selected very frequently and the errors on the sections it

covered were low. Visually it appeared to be an accurate guess too.







Again however, the polynomial function was being chosen less frequently than before. Regardless, the sinusoidal function appeared to be an accurate fit.

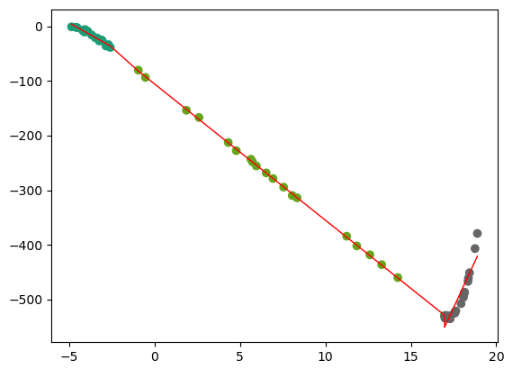
## Choosing between function types

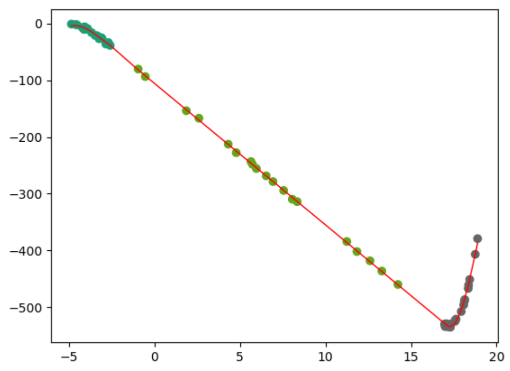
The appropriate function type for a given segment is chosen based on the cross validation error performed on the three possible fits. The method for obtaining the cross validation error is outlined in the following paragraph.

## Preventing Overfitting

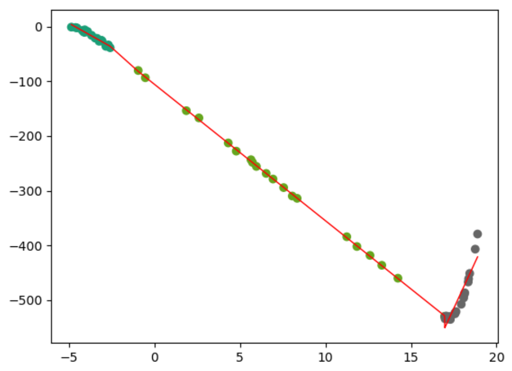
The solution uses repeated k-fold cross validation in order to minimize overfitting. The kfold function takes in the number of folds: k, the list of X parameters, the list of Ys and a string denoting the function type. First the points are randomly shuffled. Then the segment is split into k slices. The fit is trained on one of them, while the rest serves as test slices. This happens k times, each time the test slice is a different one. At each iteration the error is appended to a list. The mean of those errors is returned. A separate function repeats k-fold cross validation n times and returns the mean of the obtained cross validation errors.

The k value was set to 5. A larger k value would reduce the bias in the model's estimated performance, however it would increase the variance too. Below is an example between the fit at k = 10 and k = 5 with the former performing significantly worse.





Similarly, a low k value produces a poor result as the bias is too high. For k = 1:



Instead in order to reduce the noise in the estimated model performance the process is repeated n times and the mean of the performance is returned. The value of n has been set to 10. This comes at a cost of evaluating many more models. With 5 folds and 10 repetitions 50 different models are being fitted and evaluated. As the dataset size is relatively small the computational cost is of little concern. Furthermore, despite the sum squared error of the model being slightly higher than the lowest error returned without repetition, unlike in that case the value is consistent. What we lose in accuracy we gain in reliability, which in the fictional scenario outlined in the coursework description is a reasonable tradeoff.