# Symbols, Patterns and Signals - CW1: An Unknown Signal

Matthew Crees - kv18821

## Introduction

The situation is serious. Our early warning device has received an unknown signal and it is up to me to figure out what it is. This report details the decisions I made when developing my program, my results and a critical evaluation of my work.

## Least Squares Regression

I have implemented the matrix form of least squares regression, which can be represented by this formula:

$$\mathbf{a_{LS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$\mathbf{a_{LS}}$ is a 1-diemnsional array that holds coefficient values which are used in the reconstruction of the line. In my program, these lines, and therefore the array, take three different forms:

- $a + bx$ with the array **[*a*, *b*]**
- $a + bx + cx^2 + dx^3$ with the array **[*a*, *b*, *c*, *d*]**
- $a + b\sin(x)$ with the array **[*a*, *b*]**

### Linear Regression

Initially, I designed my program to simply calculate the best fit using linear least squares regression. Although this wouldn't work for any of the more complex data sets, I wanted to have a good starting point to make sure that the core systems of my program were working correctly. These are:

- `leastSquares()` which runs the regression formula mentioned previously.
- `ySquared()` which calculates the residuals from given sets of training and test points.
- `plotBestFit()` which, plots the data onto a graph.

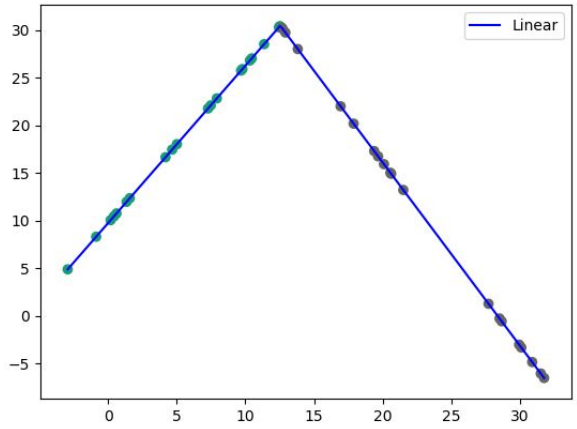


Fig. 1 - basic_2 results

**Fig. 1** shows the reconstructed lines fitting correctly on the basic_2 data set, with a total resonstruction residual of $6.47 \times 10^{-27}$ (2 d.p.).

### Extending to Polynomials

In order to extend the range of lines that my program could reconstruct, I had to alter most functions in my program so that they would differ depending on what line they needed to handle. This can be seen from the addition of `lineType` seen throughout.

The matrix **X** was expanded to include a column for the coefficients of each $x^n$ value in the polynomial. For example, expanding to cubics added two columns: $x^3$ and $x^2$.

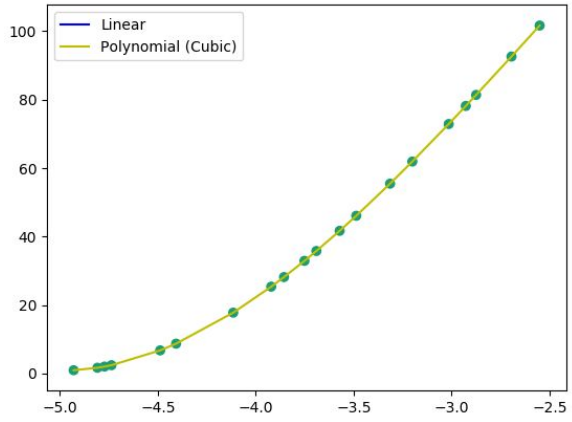


Fig. 2 - basic_3 results with cubic function

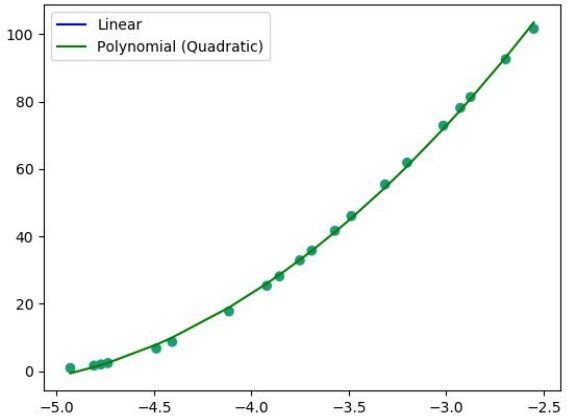


Fig. 3 - basic_3 results with quadratic function

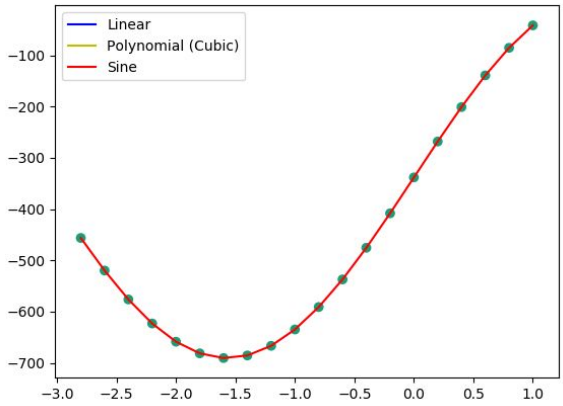


Fig. 4 - basic_4 results

I attempted polynomials up to degree 5, and found that 3 and higher seemed to yield the best results. **Fig. 2** shows the reconstructed lines fitting correctly when assuming the function is a cubic, while **Fig. 3** shows the contrast, where a quadratic function doesn't quite line up. This lead me to conclude that the polynomial function must be a cubic equation.

## Deducing the Unknown Signal

When first trying to deduce what form this signal was taking, I decided that it was best to take a step back and consider what information I could already observe from the data sets that I had been given. In particular, I found that none of the lines appeared to tend towards any plateaus. This means I could rule out the signal being logarithmic. The lines also never reached any asymptotes, meaning it couldn't be an exponential function.

After this, I decided to look into the trigonometric functions. As with exponential functions, this signal couldn't be a tangent as there are no asymptotes. However, the data did seem to fit with both sine and cosine. I decided to choose sine, but this choice was arbitrary and sine and cosine would both yield correct results, just with different coefficients.

**Fig. 4** shows the line fitting correctly to basic_4 with the sine function, and has a residual of $2.76 \times 10^{-13}$ (2 d.p.).

## Minimising the Residual

In order to minimise the residual, I first had to split the data into 20-point chunks to draw separate lines for each (this can be seen in **Fig. 1**). I then used take-one-out cross validation on each chunk in order to calculate which type of line would give the smallest residual values.

**Take-one-out cross validation:**
A single point is taken out to be used as a test point and the remaining nineteen points as training data. The line would be plotted with these training points, and then the y-squared value from that line to the remaining test points it taken. This was repeated for each of the twenty points and the values were summed up. Whichever line type gave the lowest total sum (and therefore the minimum residual) was chosen as the best fit.

## Combatting Overfitting

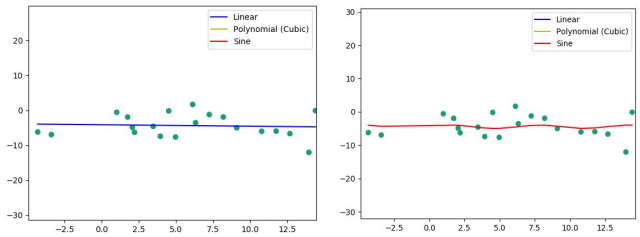


Fig. 5 - First chunk of adv_3 with (left) and without (right) naïve regularisation

As seen in **Fig. 5**, my solution at this stage still had the problem of overfitting. When there was noise in the data, such as in adv_3, lines that were meant to be linear (see **Fig. 7**) were instead being plotted as another type that yielded slightly lower residuals.

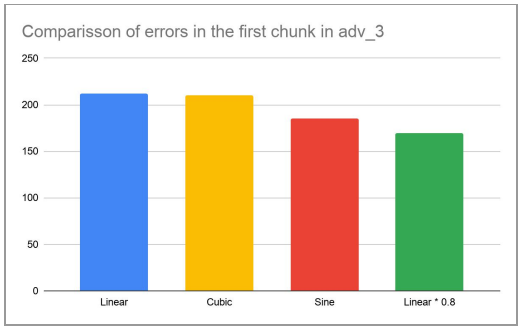


Fig. 6 - Error values in the first chunk of adv_3

In order to solve this overfitting, I decided to implement some naïve regularisation. This involved including an additional check when deciding which line type gave the best residual values: cubic and sine residual values must be lower then 80% of the linear values. As displayed in **Fig. 6**, results that are very close to the value of linear will no longer be chosen.

## My Results

I am showcasing **Fig. 7** as it best shows that all three types of line are trialed and the most appropriate one is chosen to be plotted. Although adv_3 does contain some noise, it can be seen here that the lines don't overfit.

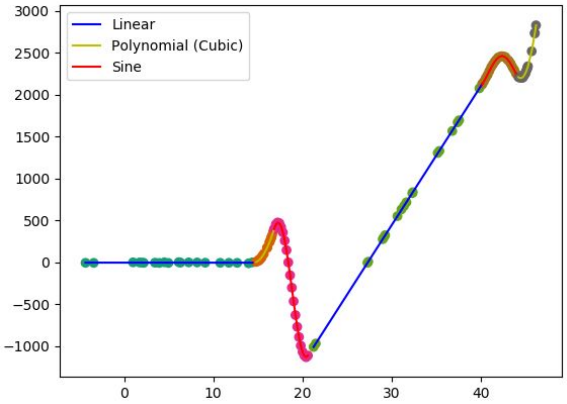


Fig. 7 - adv_3 plotted graph

| File Name | Total Reconstruction Residual |
|---|---|
| basic_1 | 1.681062589025976e-27 |
| basic_2 | 6.467081701001855e-27 |
| basic_3 | 2.9225629318655025e-18 |
| basic_4 | 2.761249399100871e-13 |
| basic_5 | 2.4960821218641167e-25 |
| adv_1 | 199.72561401537627 |
| adv_2 | 3.685132050447601 |
| adv_3 | 1058.4800032475885 |
| noise_1 | 12.207460140137103 |
| noise_2 | 849.5527462320149 |
| noise_3 | 482.909050785291 |

Fig. 8 - All the total reconstruction residuals

**Fig. 8** shows that the basic plots have close to zero residual values, resulting in the lines going through the points perfectly. The other data sets contain noise, meaning their resulting residual values are much larger. However, thes evalues are still small, and can be seen graphically in **Fig. 7**; the functions align with the data accurately.

## Critical Evaluation & Extensions

One way I could have extended my program is with the inclusion of more techniques to combat overfitting, such as more complex regularisation. However, I think that this may not be necessary as the naïve regularisation that I use already seems to work well enough.

I could also have used k-fold cross validation, instead of my take-one-out method. This would likely find better values for the coefficients as it could try more combinations of testing and training data. However, a downside to K-fold cross validation is that it may impact the efficiency of my program. Many more training and test sets need to be trialled, resulting in my program being slower to run.

A final improvement that I would like to have made is the way I handled the line types. I feel that passing through which line is which has resulted in a fair bit of needless repetition in my code.