Numerical Programming
Computational Project 2

Input:
An image of a function with added noise.
Functionality:

- Extract function-related information from the image;
- Approximate function using splines;
- Approximate function using least squares;
- Detect peak.
- Detect secondary peak.

Part 1
Extracting information

We were asked to extract information about the function from the image. Method `extractAllInfo(image, threshold)` gets all the points of the function. The logic for this method is that we transform the given image in grayscale[1]. Now all the colors are from white(255) to dark gray(0). Then we traverse the image and get all the points that are smaller than the *threshold*. In our case with the given image, since the background is white we take a *threshold* of 150. So we get points that are grayer and not white. Finally, it returns *points_list* which is all the points with format **(y,x)**.
Sadly, the drawback of the previous method is that we get all the points, even the ones that we do not need, therefore here comes the other method: `filterInfo(points)`. We filter the list returned from *extractAllInfo* and filter it, and get all the points with unique Xs. So now we get a much clearer list of all the points.
Method `construct(image, points)` is to then create an image that displays a clearer function for personal uses. You pass the image from which you started and then points extracted from previous methods.

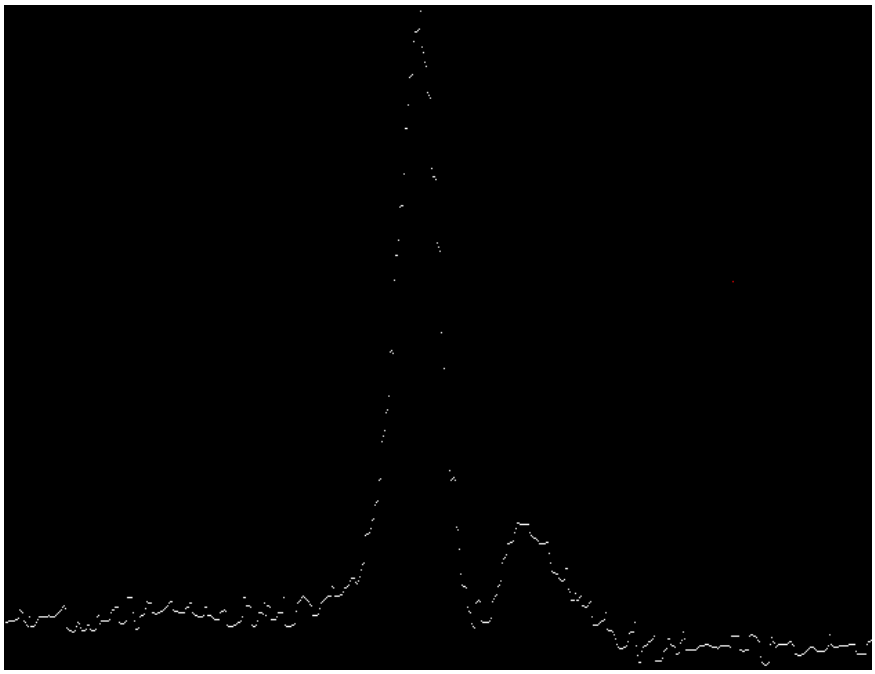

*Photo 1. Output of **construct***

[1]gray scale is done by multipling red, green and blue by coefficients, **GR = 0.299R + 0.587G + 0.114B**

Part 2

Approximation with splines

Method `Jacobi(A, b, x0, t, iter=300)` is based on the Jacobi method which is an iterative algorithm for determining solutions of a strictly diagonal system of linear equations. The logic goes like this we take A matrix, b, and x0 vector. Then we decompose A = D+L+U then the solution is obtained iteratively by $x^{k+1}=D^{-1}(b-(L+U)x^{k})$. The algorithm is repeated until convergence. We use this method later in the cubic spline algorithm.

`extractMatrix(matrix)` method takes some matrix and returns its upper, lower and middle diagonals. What we need in method `thomasAlgo(matrix, vector)` that gets matrix and vector and then extracts from the matrix, and then applies Thomas algorithm

`cubicSpline(x, y, tol=1e-100)` The method gets x and y points and then we generate a diagonal matrix, we apply Jacobi's method. In return we get coefficients of degree 1 - b, degree 2 - c, and degree 3 - d.

Part 3

Approximate with least squares

In this part, we only have one method which `leastSquare(degree,y_points, x_points).`
We pass the degree of differentiation and pass the x and y of the function. The method gives us the plot of a function and also the least square regression to see it clearly. In the photo below we can see our function and also the regression of degree 1.

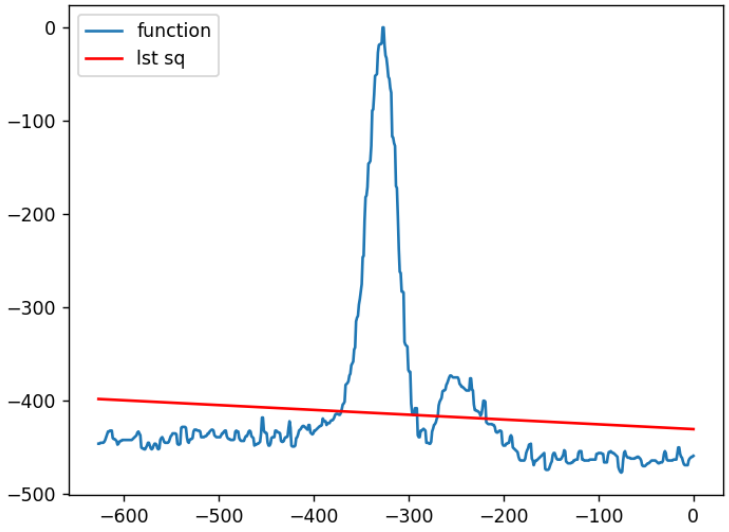

*Photo 2. Output of **leastsquare***

Part 4 & 5

Peak Detection

To Detect peaks in our function we have a method `peakDetection(y, l, threshold, inf)`. The logic goes like this: some datapoint is given n number away from moving mean then we detect the peak. The method takes four inputs:

- Y - datapoints
- L - how many last observations we should use to check new datapoint
- Threshold - score at which we detect the peak
- Inf - influence of new peaks on the mean and standard deviation. (from 0 to 1, default equals 0)

To dive into details of how we check each data point is that we compare the absolute meaning of datapoint minus the previous avg(mean of the previous *L* number *newY* (`inf * y[i] + (1 - inf) * newY[i - 1])` up until now) to *threshold* multiplied to *stdev* (standard deviation of last L number of newY). If the absolute meaning is smaller then we do not have a peak, otherwise, we get the peak.
Now, for our function we run `peakDetection(y, 30, 4.63)` and this gives us:

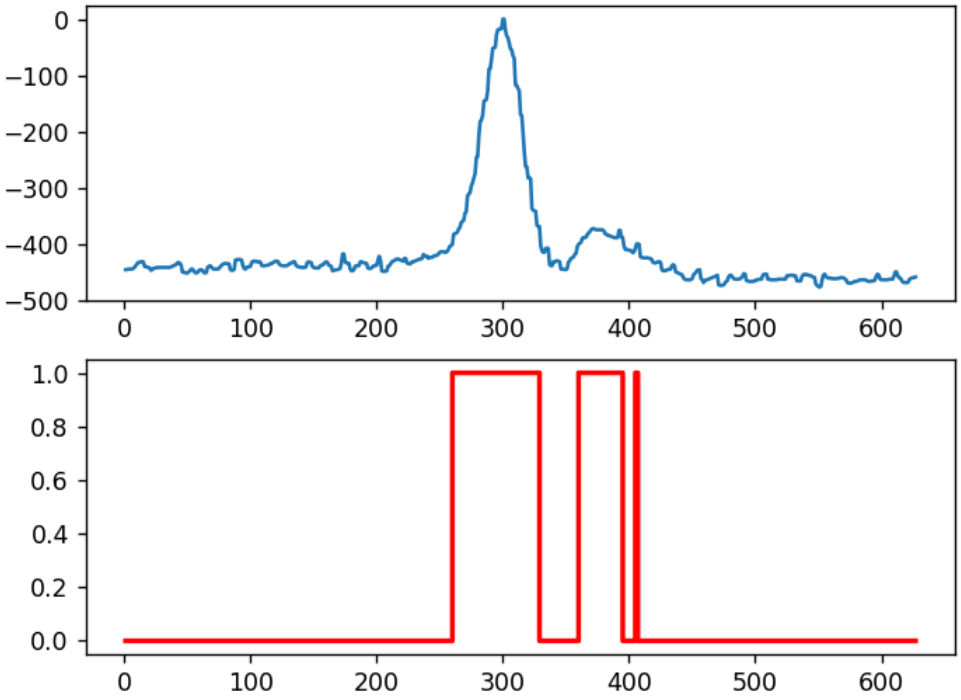

*Photo 3. Output of **peakDetection***

We see that it detected the biggest peak in the middle and also the smaller peak and a little peak near the second one. This function detects the first peak and also the secondary peak.