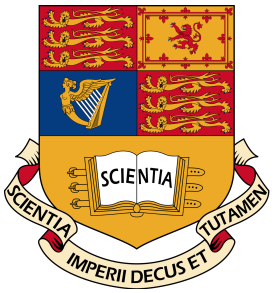
IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

M4N9 Project 1: Topographical Reconstruction of Tycho Crater using Moving Least Squares

Christopher McLeod

November 3, 2017



Abstract

This project makes use of Householder QR-decomposition to highlight the effectiveness of 'Least Squares' and 'Moving Least Squares' techinques in reconstructing 2D topographical information about the Moon's crater Tycho. The aim of this paper is to explore the quality and effectiveness of the two techniques by applying them to data.

Summary

FOREWORD

In the interest of giving the reader a clarity an appendix has been added at the end of the paper. Inevitably this project involved writing many many functions and scripts, and the process has involved augmenting, restructuring and renaming files and so a flow diagram of how these interact has been added. It is my hope that this gives a full picture of how the algorithms interact and feed into one another.

In the interest of making this paper as consise as possible, in general functions and scripts have been excluded from the final write-up. What is included is a description of the algorithms and programs, in terms of what function they perform. For the implementation, the code is provided alongside this paper with detailed comments.

1 INTRODUCTION: STATEMENT OF THE PROBLEM

The problem starts with two vectors, each with 1001 entries. One, henceforth refered to as h, contains the recorded elevation data of the tycho crater, and one (henceforth x) which is an equispaced vector detailing when elevation data was sampled.

The aim is to reconstruct the surface using polynomials of relatively small degree and ultimately compare the effectiveness of different methods, as well as exploring ways to reduce storage and improve the complexity and runtime of the reconstruction.

Generally, our initial aim is to find, for some specified n, a polynomial $p \in \mathscr{P}_n$, the set of polynomials of degree $\leq n$ such that the quantity

$$\sum_{i=1}^{N} \left| p(x_i) - h_i \right|^2 = \left\| p(x) - h \right\|_2^2$$

is minimised, where $p(x) = [p(x_1), \ldots, p(x_N)]^T$ and N is the number of data points in total (1001 in this case). However, later we will reformulate this aim so that we can construct a smooth surface which deals with noise better. The method is computationally more instensive and works by minimising local error at every data point. In this scenario, the aim is to acheive minimise the following over each $h_j \in h$

$$\min_{p \in \mathscr{P}_n} \sum_{i=1}^{N} \theta(x_j - x_i) \left| p(x_j) - h_i \right|^2$$

The methodology for both of these aims is in reformulating them into problems of the form

$$Ac = b$$

where $A \in \mathbb{C}^{m \times n}$ $(m \geq n)$, $c \in \mathbb{C}^n$ an unknown variable (i.e. what we're solving for in order to minimise our 'cost' equation) and $b \in \mathbb{C}^m$. From there, we seek to apply the Householder triangularisation (described below) to the matrix A in order to solve a potetentially overdetermined system of equations, by minimising the error or 'cost' in c

$$\min_{c \in \mathbb{C}} \| Ac - b \|_2^2 \qquad (\star)$$

2 THE LEAST SQUARES SOLUTION

We turn our attention to a global solution for the n^{th} degree polynomial $p^\star \in \mathscr{P}_n$ such that

$$\left\| p^\star(h) - h \right\|_2^2 = \min_{p \in \mathscr{P}_n} \left\| p(x) - h \right\|_2^2$$

i.e. over $p^\star(x_i)$ has the minimum Euclidean distance to h_i as is possible for an n^{th} degree polynomial for all $i = 1 \to N$.

Fix $n \in \mathbb{N}$ arbitrarily for the time being. We seek to reformulate the above cost equation, so we can solve the problem of finding $x^\star \in \mathbb{C}^n$ such that an equation of the form $(\star)$ is solved. For now, define a polynomial $p \in \mathscr{P}_n$ as

$$p(x) = c_0 + c_1 x + c_2 x^2 \cdots + c_n x^n$$

and let R be defined such that

$$R^2 = \left\| p(h) - h \right\|_2^2 = \sum_{i=1}^{N} [h_i - (c_0 + c_1 x_i + \cdots + c_n x_i^n)]^2$$

In the effort of minimising this quantity, we take partial derivatives and set them to zero

$$\begin{aligned}
\frac{\partial R^2}{\partial c_0} &= -2 \sum_{i=1}^{N} [h_i - (c_0 + c_1 x_i + \cdots + c_n x_i^n)] = 0 \\
\frac{\partial R^2}{\partial c_1} &= -2 \sum_{i=1}^{N} [h_i - (c_0 + c_1 x_i + \cdots + c_n x_i^n)] x = 0 \\
&\vdots \\
\frac{\partial R^2}{\partial c_n} &= -2 \sum_{i=1}^{N} [h_i - (c_0 + c_1 x_i + \cdots + c_n x_i^n)] x^n = 0
\end{aligned}$$

Rearranging to get the terms of h on the right-hand-side and rewriting in matrix form gives the equations

$$\begin{bmatrix}
n & \sum_{i=1}^N x_i & \cdots & \sum_{i=1}^N x_i^n \\
\sum_{i=1}^N x_i & \sum_{i=1}^N h_i^2 & \cdots & \sum_{i=1}^N x_i^{n+1} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{i=1}^N x_i^n & \sum_{i=1}^N x_i^{n+1} & \cdots & \sum_{i=1}^N x_i^{2n}
\end{bmatrix}
\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}
=
\begin{bmatrix} \sum_{i=1}^N h_i \\ \sum_{i=1}^N x_i h_i \\ \vdots \\ \sum_{i=1}^N x_i^n h_i \end{bmatrix}$$

Which one can verify is the the system

$$Ac = h$$

pre-multiplied on both sides by A^T for $c = [c_0, c_1, \ldots, c_n]^T$, where A is the $N \times (n+1)$ so-called Vandermonde matrix

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}$$

[1] The check is left to the reader.

Using the constructed notation, the problem is transformed to minimising the error in an n^{th} degree polynomial's approximation of data vector to minimising the error in an overdetermined linear system (as N is generally much larger than $n+1$), i.e.

$$\min_{p \in P_n} \left\| p(x) - h \right\|_2^2 = \min_{c \in \mathbb{C}^n} \| Ac - h \|_2^2 \qquad (\dagger)$$

From here, we use the fact that using Householder Triangularisation to give the reduced $Q-R$ decomposition leads to the least squres solution to this problem. i.e. say $\hat{Q} \in \mathbb{C}^{N \times N}$ and $\hat{R} \in \mathbb{C}^{N \times n+1}$ are the factors of A from the Householder Triangularisation, $A = \hat{Q}\hat{R}$, then

$$Ac = \hat{Q} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1,n+1} \\ 0 & r_{12} & \cdots & r_{2,n+1} \\ 0 & 0 & \cdots & r_{3,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{n+1,n+1} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} c = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_{n+1} \\ h_{n+2} \\ \vdots \\ h_N \end{bmatrix} := \begin{bmatrix} \\ H_1 \\ \\ \\ \\ \\ H_2 \\ \\ \end{bmatrix}$$

Defining $H_1 = [h_1 h_2 \ldots h_{n+1}]^T$ and $H_2 = [h_{n+2} \ldots h_N]^T$ and letting R_1 be the non-zero square upper-triangular $(n+1) \times (n+1)$ matrix so that $\hat{R} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, then the reduced $Q-R$ decomposition guarrantees that the quatity

$$\begin{aligned} \| Ac - h \|_2^2 &= \left\| \hat{Q}\hat{R}c - h \right\|_2^2 \\ &= \left\| \hat{R}c - \hat{Q}^\star h \right\|_2^2 \\ &= \| R_1 c - b_1 \| + \| b_2 \|_2^2 \end{aligned}$$

is minimised by solving $R_1 c = b_1$ [2]. The steps in the above use the fact that $\hat{Q}$ is unitary and the definitions $b_i = \hat{Q}^\star H_i$ for $i = 1, 2$ have been made.

To this end, one solves for the coefficients of $p^\star$ by using backwards substitution to sovle the system of equations given by

$$R_1 c^\star = b_1$$

where $c \in \mathbb{C}^n$ minimises †. From this we easily obtain $p^\star(x) = c_0^\star + c_1^\star x + \cdots + c_n^\star x^n$.

2.1 PART 1

The implementation of the least squares above solution is embodied within the function `lspoly(deg)`, which carrys about the above procedure. The function takes an argument 'deg' (the desired degree of the polynomial) and starts by creating the $N \times (deg+1)$[1] Vandermonde function, using the function to set $A =$ `vmon(x,deg+1)`.

Next the Householder Triangularisation is performed on A. The outputs $[V,R] =$ (house(A)) are an $N \times (n+1)$ matrix containing the vectors defining the Householder reflections, and the $(n+1) \times (n+1)$ upper-triangular matrix equivalent to R_1 in the model case above.

At this point, since $\hat{Q}$ has not been calculated explicitly, the function `Qadjh(V,h)` takes as its arguments, the Householder reflection vectors and the elevation data vector, then computes $Q^\star h$ and stores it in b.

At this point, we are at the stage where we must solve $R_1 c^\star = b_1$. So, the function `backsub(R,b)` performs the simple backwards substitution necessary to get the vector of monomial coefficients of the deg+1 degree least polynomial. Premultiplying this vector by A gives the N-dim vector of least squares approximations for h_i, $i = 1 \rightarrow N$

The script `lspolyplot` creates the below three plots of the least squares approximations against x for deg $= 5, 10, 20$

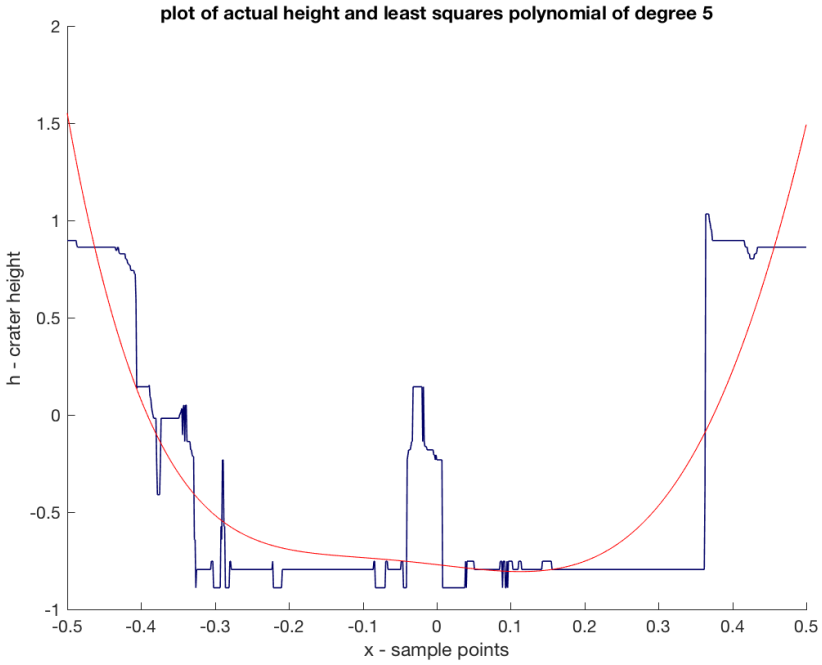


Figure 2.1: plot of x against elevation and lsp deg 5

And for those degree values, the script `makeTables` creates a table which details the six lowest monomial coefficients of the three least squares polynomials, outputting

[1] in the above $n \rightarrow$ deg

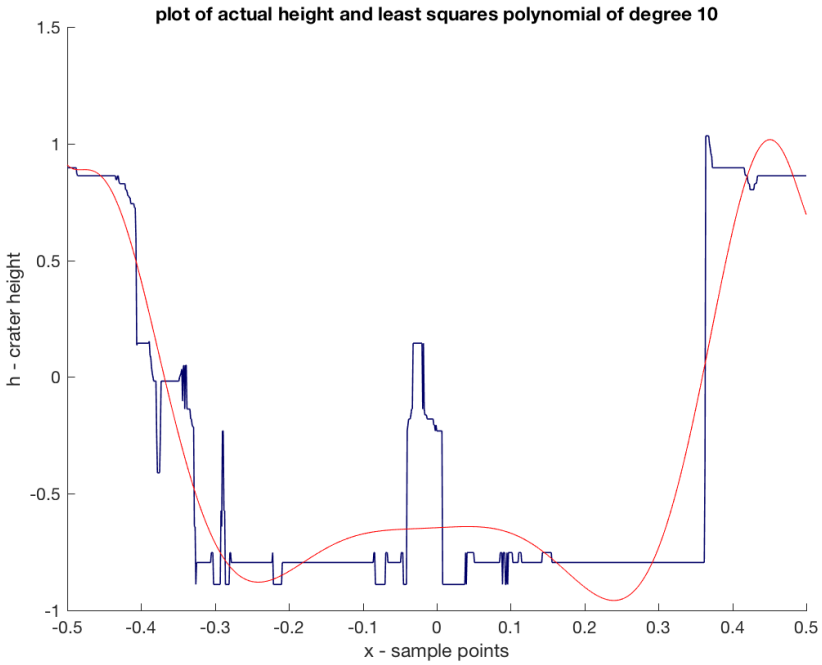


Figure 2.2: plot of x against elevation and lsp deg 10

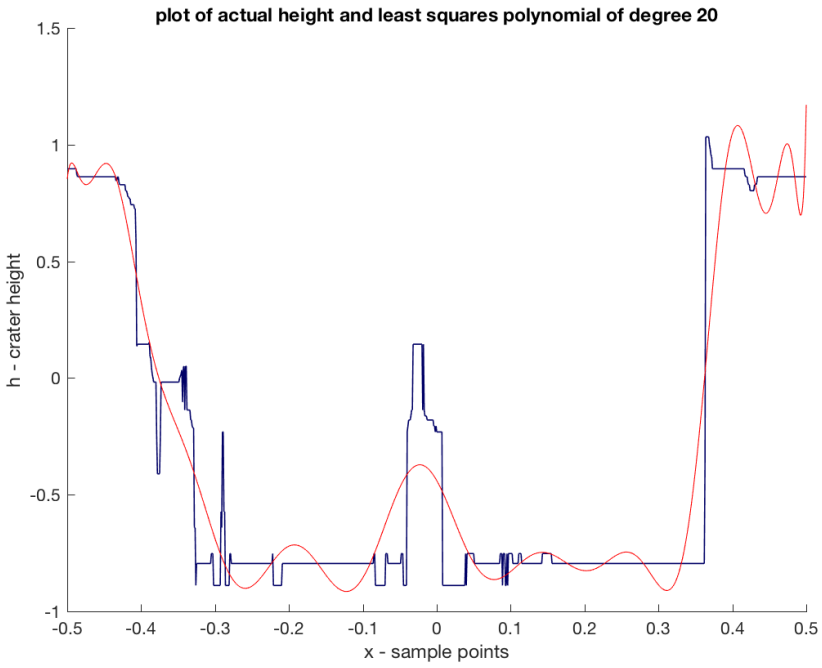


Figure 2.3: plot of x against elevation and lsp deg 20

deg	c0	c1	c2	c3	c4	c5
5	-0.77026	-0.4342	-0.32875	8.1133	37.996	-26.439
10	-0.64603	0.15433	0.24913	-15.936	-329.11	235.78
20	-0.44001	-5.6094	-94.455	1129.6	8037.1	-76082

Table 2.1: Least Squares Coefficients for varying degrees

deg	$\left\|p^\star(x)-h\right\|_2^2$
5	10.505
10	7.7858
20	5.8983

Table 2.2: Least Sqaures Errors for varying degrees

The blue curve shows the original data, and in each case the red shows the least squares polynomial for the given degrees. `makeTables` also provides a value for the 'error' $\left\|p^\star(x)-h\right\|_2$ in a table

One can see that as the degree of the least squares polynomial increases, the error in the approximation decreases. The plot of the degree 5 polynomial shows the most basic trend of the elevation data, that the crater has a big drop from the edges. This improves in the degree 10 polynomial, and some indication of the peak around the middle of the crater can be seen, though the fit here does not look good ,as it does not capture the sharp increase in elevation and instead the least squares polynomial has a small and widely increased elevation. The degree 20 polynomial fits the peak in the middle of the crater best, one can see indication of a peak here. However, the signs of overfitting start to become visible here outside of this region, for example, on either side of the peak where the prediction looks too volatile. At the edges, we see evidence of Runge's phenomenon occuring when the magnitude of the derivatives grows to quickly and having sample points equidistance leads to unwanted oscilliatory errors at the edges [3].

3 THE MOVING LEAST SQUARES SOLUTION

We now turn our attention from away from finding a global least squares polynomial to reconstruct the data, and towards minimising the approximation locally. Alexa et al. [4] provide a methodology for this. A simpler version of this is implemented: for any point $x_i \in x$ one conducts a weighted least squares, determined by the 'radial' Gaussian weighting function

$$\theta(r) = e^{-\frac{r^2}{\eta}}$$

taking $r = \left|x_i - x_j\right|$ over the $j = 1 \to N$, where η some variable. Note that $\theta(r) > 0$ for all r and θ decays as $\left|x_i - x_j\right| \to \infty$. We use the weighted least squares result at x_i for the i^{th} value in the mls approximation.

Weighting each point like this leads to a new optimisation problem. That is, at any point x_m, find the polynomial (of some prescibed degree n) that satisties the weighted least squares cost equation

$$\min_{p\in\mathscr{P}_n}\sum_{i=1}^{N}\theta(|x_m-x_i|)\left|p(x_i)-h_i\right|^2 \tag{$\star'$}$$

Much as in the case of the least squares solution, we seek to reformulate this as a problem of solving a cost equation of the form

$$\min_{c\in\mathbb{C}^n}\|Ac-b\|_2^2 \tag{$\dagger'$}$$

A similar derivation as in the least squares case allows a transformation of the problem to this case. Let the $p\in\mathscr{P}_n$ be abritrary, $p(x)=c_0+c_1x+\cdots+c_nx^n$ and define as follows

$$A\leftrightarrow\begin{bmatrix}\theta_1 & \theta_1x_1 & \cdots & \theta_1x_n\\ \theta_2 & \theta_2x_2 & \cdots & \theta_2x_n^n\\ \vdots & \vdots & \ddots & \vdots\\ \theta_N & \theta_Nx_2 & \cdots & \theta_Nx_n^n\end{bmatrix}$$

$$b\leftrightarrow\begin{bmatrix}\theta_1h_1\\ \theta_2h_2\\ \vdots\\ \theta_Nh_N\end{bmatrix}$$

$$c\leftrightarrow\begin{bmatrix}c_0\\ c_1\\ \vdots\\ c_n\end{bmatrix}$$

where $\theta_j:=\theta(\left|x_m-x_j\right|)$. From here we can check that we now have an equivalence between a problem of type ($\dagger'$) which is equivalent to our original problem of type ($\star'$). That is, for the defined values

$$\min_{c\in\mathbb{C}^n}\|Ac-b\|_2^2=\min_{p\in\mathscr{P}_n}\sum_{i=1}^{N}\theta(|x_m-x_i|)\left|p(x_i)-h_i\right|^2$$

for every $x_m\in x$.
One can verify this, since

$$\|Ac-b\|_2^2=\left\|\begin{bmatrix}\theta_1 & \theta_1x_1 & \cdots & \theta_1x_n\\ \theta_2 & \theta_2x_2 & \cdots & \theta_2x_n^n\\ \vdots & \vdots & \ddots & \vdots\\ \theta_N & \theta_Nx_2 & \cdots & \theta_Nx_n^n\end{bmatrix}\begin{bmatrix}c_0\\ c_1\\ \vdots\\ c_n\end{bmatrix}-\begin{bmatrix}\theta_1h_1\\ \theta_2h_2\\ \vdots\\ \theta_Nh_N\end{bmatrix}\right\|_2^2$$

$$
\begin{aligned}
&= \left\| \begin{bmatrix} \theta_1(c_0 + \cdots + c_n x_n^n) \\ \theta_2(c_0 + \cdots + c_n x_n^n) \\ \vdots \\ \theta_N(c_0 + \cdots + c_n x_n^n) \end{bmatrix} - \begin{bmatrix} \theta_1 h_1 \\ \theta_2 h_2 \\ \vdots \\ \theta_N h_N \end{bmatrix} \right\|_2^2 \\
&= \left\| \begin{bmatrix} \theta_1(p(x) - h_1) \\ \theta_2(p(x) - h_2) \\ \vdots \\ \theta_N(p(x) - h_N) \end{bmatrix} \right\|_2^2 \\
&= \sum_{i=1}^{N} \theta_i \left| p(x) - h_i \right|^2
\end{aligned}
$$

Hence, by minimising over the $c_i \in^n$ we get that

$$\min_{c \in \mathbb{C}^n} \|Ac - b\|_2^2 = \min_{p \in \mathscr{P}_n} \sum_{i=1}^{N} \theta(|x_m - x_i|) \left| p(x_i) - h_i \right|^2$$

As hoped.

3.1 PART 2

With this in mind, we set out to solve another (overdetermined) linear system. As the Here is a detailed explaination of the steps of the algorithm for fixed n, running j from 1 to N:

- Fix x_j
- Weight the $N \times (n+1)$ degree Vandermonde matrix A, such that $A_{ij} = \theta(\left| x_i - x_j \right|) x_i^{j-1}$ so if $\theta_i := \theta(\left| x_j - x_i \right|)$ and we call WA the 'weighted' Vandermonde matrix

$$WA = \begin{bmatrix} \theta_1 & \theta_1 x_1 & \cdots & \theta_1 x_n \\ \theta_2 & \theta_2 x_1^2 & \cdots & \theta_2 x_n^n \\ \vdots & \vdots & \ddots & \vdots \\ \theta_N & \theta_N x_1^2 & \cdots & \theta_N x_n^n \end{bmatrix}$$

- Weight the elevation vector. If we call this Wh, then $[Wh]_i = \theta_i h_i$, so that

$$Wh = \begin{bmatrix} \theta_1 h_1 \\ \theta_2 h_2 \\ \\ \vdots \\ \\ \theta_N h_N \end{bmatrix}$$

- Solve the system $WAc = Wh$ for c, using the methods in the least squares solution giving the coeficients i.e.
  - Perform the Householder Triangularisation on WA to acheive V and R the Householder reflections and upper triangular
  - Solve the equation $\hat{Q}\hat{R}c = Wh$ for c using `Qadjh(V,Wh)`
  - Right-multiply c by A (the unweighted Vandermonde) to find the weighted least squares polynomial
- store the j^{th} component of the weighted least squares vector in the j^{th} component of the moving least squares vector.

One can see that in the after some preparation, in the fourth bullet point, the least squares solution is implemented. Knowing this, the least squares algorithm is augmented into `MLSpoly(deg, eta, epsilon)`. `MLSpoly` beigns by initialising the MLS vector, *MLSp*, as a length N column vector of zeros and then enters a for loop, running through the entries of x. In the for loop, it makes a call to `weightVec` and stores the result in a variable w, which achieves

$$w = \begin{bmatrix}\theta_1, & \theta_2, & \cdots, & \theta_N\end{bmatrix}^T$$

From there, a larger matrix is made from w, called W, which essentially turns creates matrix which is 0 everywhere but the diagonal, which has the values of w.

$$W = diag(w) = diag(\begin{bmatrix}\theta_1, & \theta_2, & \cdots, & \theta_N\end{bmatrix}^T)$$

From here, `MLSpoly` passes deg and W to the function `weightedls`, which calculates `WA = W*A` and `Wh = W*h` and carries out the same calculation as in the least squares solution, this time A is replaced by WA and likewise, h by Wh. The last line of the for loop sets the i^{th} value of the output vector to the i^{th} vector of the output vector from (assuming the loop is on its i^{th} iteration).

3.2 PART 3

It is worth taking a closer look at the way in which `weightVec` is built. Given that it it is called N times and for each of these loops, it could easily do N calulations each time, the number of FLOPs is $O(N^2)$.
By introducting a tolerace value ϵ, we can speed up the calculation of the weight vector by negating, for each x_i, the calculation of the weight function $\theta(|x_i - x_j|)$ if it is known before hand that

$$\theta(|x_i - x_j|) < \epsilon$$

To this end, the `weightVec` algorithm is built as follows. For fixed x_i

- Calculate the maximum distance x_j is allowed to be from x_i, which solving $\theta(r)$ for (r) gives $\theta(r) > \epsilon$ if $r \geq \sqrt{-\eta log(\epsilon)})$. Store $r_max = \sqrt{-\eta log(\epsilon)})$

- Use the fact that x is a monotonic vector to calculate the maximum x_j either side of x_i, i.e
  - $x_{max} = min(x_i + r_{max}, x_N)$
  - $x_{min} = max(x_i - r_{max}, x_1)$
- Use a logical index to store the values of x_j that produce $\theta(|x_i - x_j|) \geq \epsilon$, specifically those $y \in x : x_{min} < y < x_{max}$, store this in nz_{slice} named for the 'non-zero slice' taken from x.
- run through the non-zero slice and fill in θ_j in the output vector, in the j_{th} position.

This can largely cut down the number of FLOPs `weightVec` carries out. By making the scrifice of doing 4 preparatory calculations, we cut down the size of loop. For example, if $\eta = 10^{-3}$, $\epsilon = 10^{-6}$ then $r_{max} = 0.1175$, so that taking the most disasterous case, $i = 501$, exactly half way through the length of x, we have $x_{max} = 0.1175$ and $x_min = -0.1175$ which gives us a for loop as long as

$$\texttt{length}(nz_{slice}) = 235$$

and so, instead of doing $N * N = 1002001$ calculations in creating the weight vectors for every x_i, we have *at most* (given that we've considered the worst case) $N * 235 = 235235$, approx. a quarter of the calculations.

3.3 PART 4

The script `MLSpolyplot` creates the below plots, where $\epsilon := 10^{-6}$. In the first three plots η is fixed at 10^{-3}, so that the effect the degree has on the fit of the polynomial can be explored. The degree is allowed to vary from 1 to 3 to 10
Immediately, one notices that even for the degree 1 reconstruction, the fit is much more accurate than in the least squares solution. (To varying extents) all plots give a fairly good an indication of the peak, which gets better as the degree increases. The width of the peak is more accurate that in the least squares solution, showing that the moving least squares is more sensitive to sharp changes in elevation. This is undoubtedly because of the fact that it finds the solutions locally, and so can rapid changes in the data more accurately. While the degree 10 polynomial captures the peak most accurately, we see Runge's phenomena occuring at the edges once again. This occured more rapidly than in the least squares because having so many polynomial approximations means the magnitude of the derviatives of the overall curve grows faster.
The last three graphs produced fix the degree at 3 and allow η to take values 10^{-2}, 10^{-3} and 10^{-4}, which produces the following

The first thing to note is that η is much more sensitive variable than the degree, since small peturbations in η lead to large changes in output. The second thing to note is that these small peturbations, decreasing η leads to drastically better fit. For the three values of η we see a much more accurate reconstruction of the data than for the three degree values. We also have no issues with Runge's phenomena, as we can restrict the degree of the polynomials.

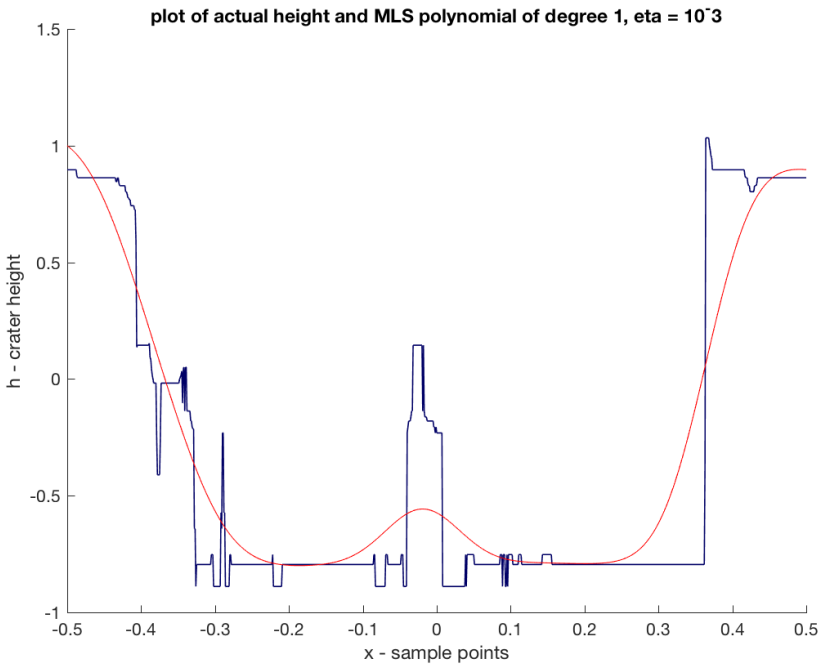


Figure 3.1: plot of x against elevation for deg=1, eta = $10^{-}3$

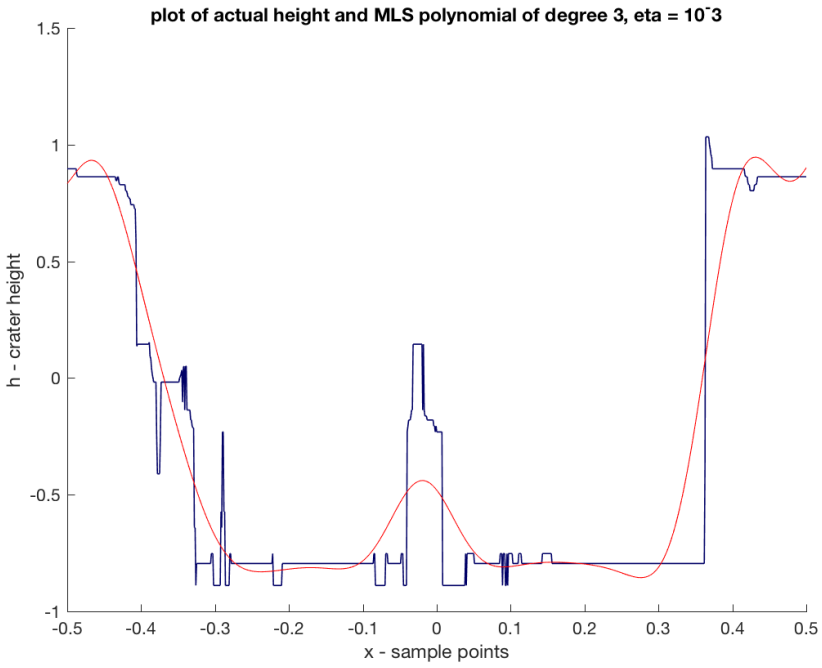


Figure 3.2: plot of x against elevation for deg=3, eta = $10^{-}3$

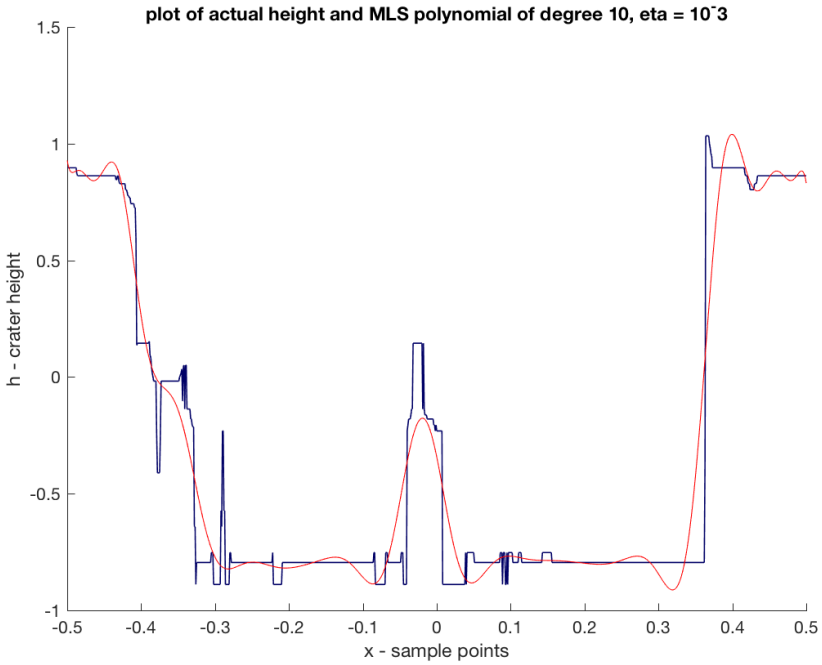


Figure 3.3: plot of x against elevation for deg=10, eta = $10^{-}3$

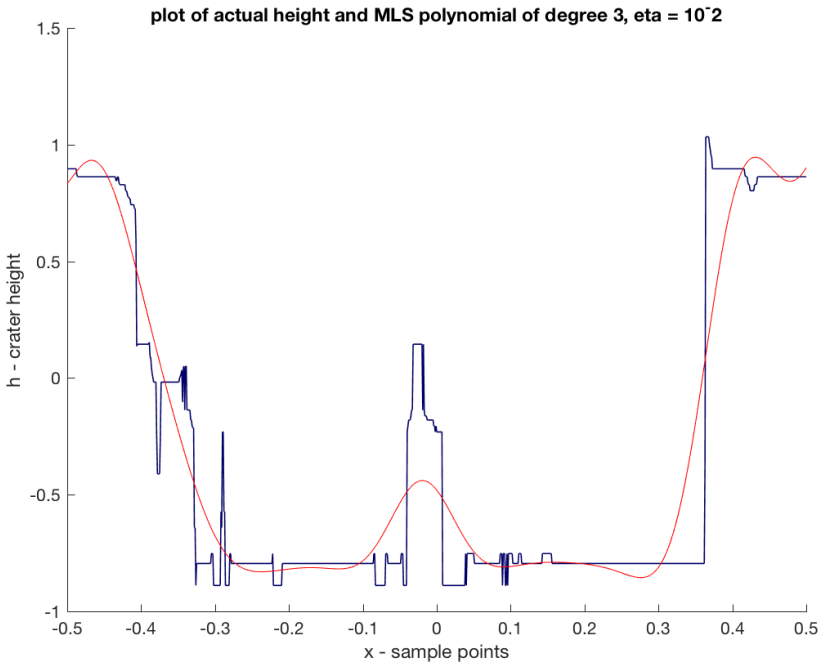


Figure 3.4: plot of x against elevation for eta = $10^{-}2$, deg = 3

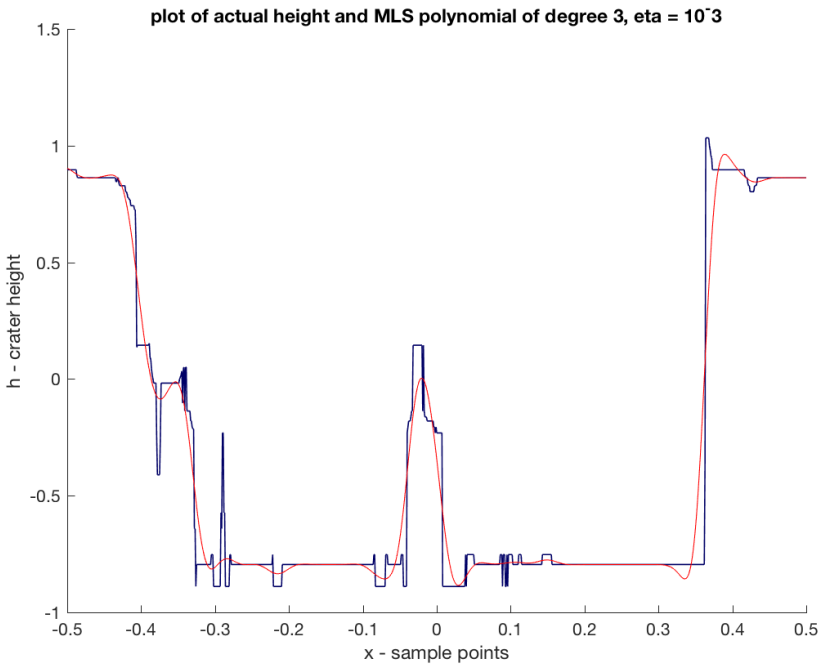


Figure 3.5: plot of x against elevation for eta = 10^-3, deg = 3

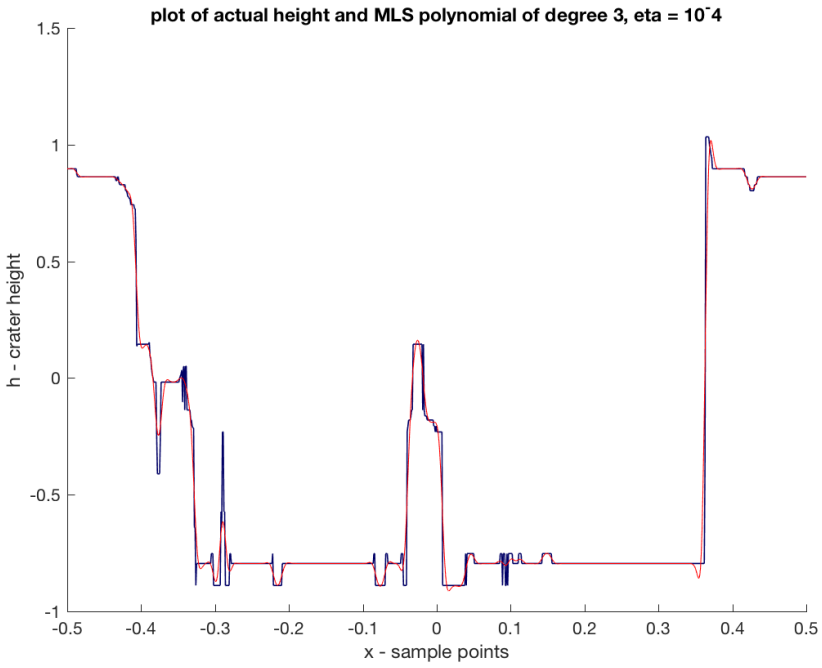


Figure 3.6: plot of x against elevation for eta = 10^-4, deg = 3

deg	$\left\| p^{\star}_{n,3}(x) - h \right\|_2$
1	4.5577
3	3.6557
10	2.8024

Table 3.1: MLS Error for Varying degrees

$\eta = 10^{-s}$	$\left\| p^{\star}_{3,s}(x) - h \right\|_2$
10^{-2}	6.4977
10^{-3}	3.6557
10^{-4}	2.039

Table 3.2: MLS Error for Varying η

To highlight this comparison between the effect of η and degree, `makeMLStables` produces the following tables

(where $p^{\star}_{n,s}(x)$ is the n^{th} MLS polynomial for $\eta = 10^{-s}$)

So the error decreases for the degree 3 polynomial by decreasing η at a rate relatively faster than when the degree is increased.

Although the error for the mls of degree 10 where $\eta = 10^{-3}$ is similar to the error of the approximation of degree 3 with $\eta = 10^{-}4$ one can see graphically that the fit is much better in the latter case.

We have seen that the results of the MLS are generally more desirable that those of the least squares approximation, but of course it comes at a price as it is more computationally intensive. If we let the number of FLOPs that the least squares algorithm needs be C, then the number of FLOPs the MLS algorithm requires, F is roughly

$$F = N(W + C + N(N-1)(n+1) + N(2n+1))$$

Where W is the number of calculations required to calculate the weight vector and the last two terms acocunt for the additional calculation of WA and Wh in addition to the least squares algorithm C which is performed upon this. Of couse, the multiplication of the sum comes from performing the weighted least squares at every point, of which there are N. For a better look at what W is, its easier to analyse if we fix the problem for $\eta = 10^{-3}$ and note that that increasing (resp. decreasing) η means increasing (resp. decreasing) the number of calculations. From before, we have the size of nz_{slice} to be around a quater of N, so, approximately

$$W = N/4 + 5$$

where 5 accounts for the preparatory floating op.s to filter x before applying θ, adding this

in and factoring where possible, we roughly have

$$F = CN + N^2((N-1)(n+1) + 2n + \frac{11}{5}) + 4N$$

So the least scales by the size of the data and incures an added cost of some quadratic in N.

4 FILTERING: MLS FOR SAMPLED DATASET

In Alexa et al. [4], their aim is to reduce the dataset of the scanned surface, potentially removing noise, making for smoother reconstructions as well as saving memory on storage. Now the attention turns to performing a similar action on the elevation data. The aim now is to reduce the number of elevation data points by which we calculate a weighting in the MLS procedure. That is, we take a subset $S \subset x$ then we now seek to minimise the cost equation for each $x_j \in S$

$$\sum_{x_i \in S} \theta(x_j - x_i)\big|p(x_i) - h_i\big|^2$$

For the following, we fix the degree of the MLS approximation at 3 and $\eta = 10^{-3}$. We investigate on taking linear subsets (x_i, h_i) of equispaced points, as well as taking samples that bunch in one particular region.

`reduced_MLSpoly` is the function written to take an equispaced sample x depedent on how large the user wants the sample to be. This is done by augmenting the original MLS alorithm, noting that to minimise the above cost equation, is to minimise over the full MLS cost equation if we redefine

$$\theta(\big|x_j - x_l\big|) = \begin{cases} \theta(\big|x_j - x_l\big|), & \text{if } x_l \in S \\ 0, & \text{otherwise} \end{cases}$$

and so, we carry out an augmented MLS procedure. This procedure creates a new vector x_{red} which has the same size as x, but has zeros at the x_k if $x_k \notin S$. It uses this to create a more sparse weight vector, and the MLS procedure is carried out from here. At every point, the coefficients are left multiplied to give the the WLS for those points that we consider weighting. A new function `red_weightVec` is augmented from `weightVec` and does the job of weighting non-zero entries in a simplified way, so as not to overcomplicate the situation and make the algorithm robust.

4.1 PART 5

Here are plots comparing the actual data for equidistant sample of sizes $n = 500, 100, 25$and20 generated by `red_plot`

(blue: original elevation data, orange: full MLS solution, red: reduced MLS)

What we see from this is that the sample reconstructs the orginal solution pretty well if we

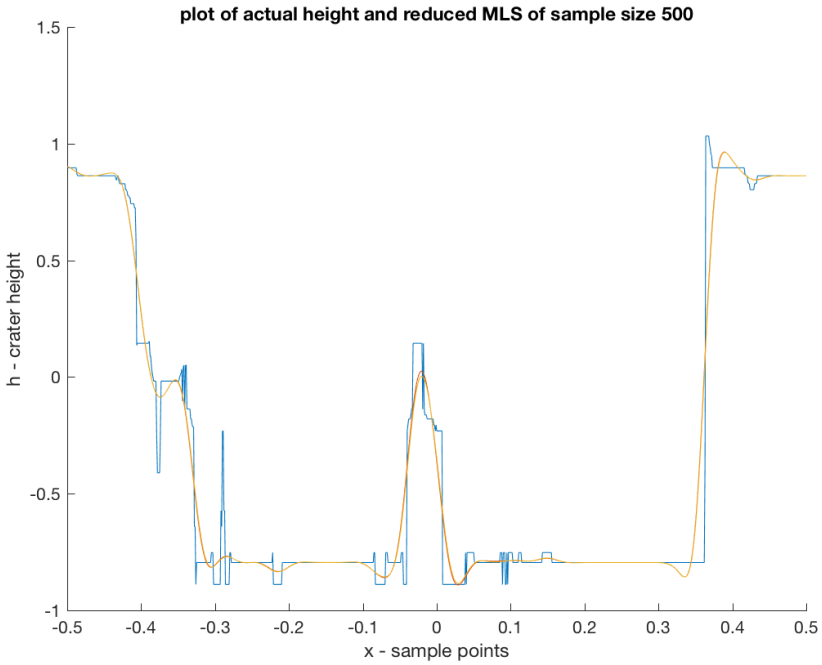


Figure 4.1: plot of x against elevation for full MLS and size 500 sampled MLS

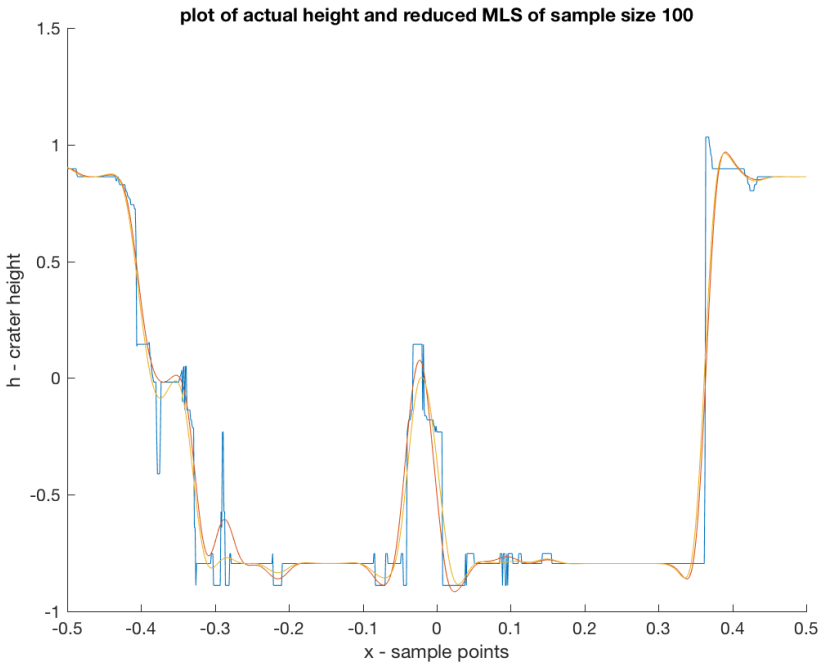


Figure 4.2: plot of x against elevation for full MLS and size 100 sampled MLS

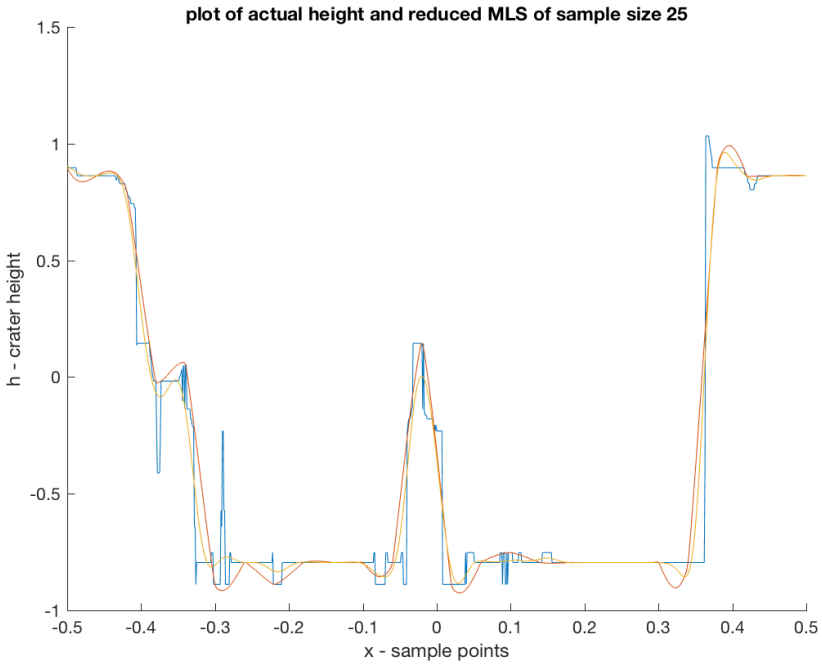


Figure 4.3: plot of x against elevation for full MLS and size 25 sampled MLS

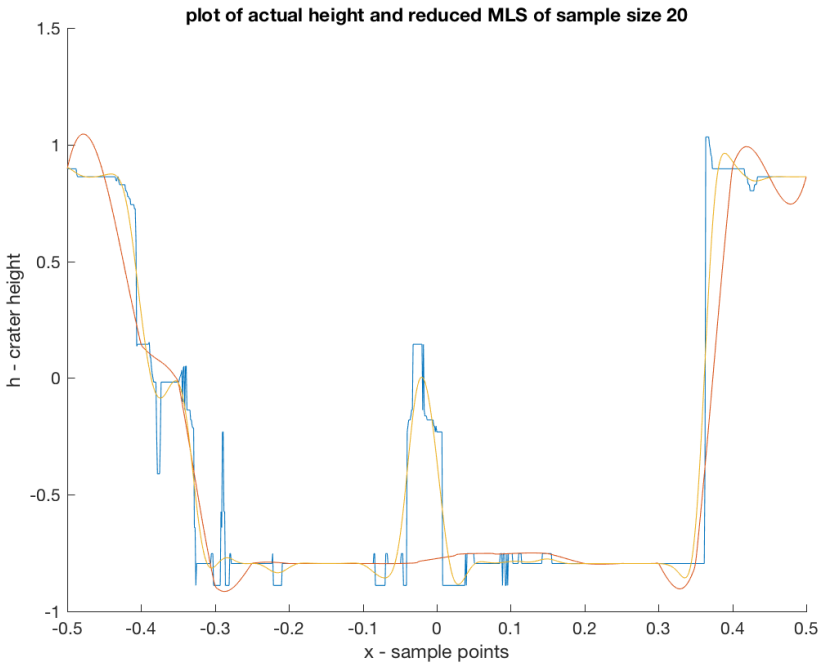


Figure 4.4: plot of x against elevation for full MLS and size 20 sampled MLS

half, or even tenth the sample size over which we weight. However, the reduced MLS seems to fall apart when taking a sample size between 25 and 20. This is because η and the number of points in the sample are closely related. If the sample size becomes too small, eventually the distance between the $x_k \in S$ (the sample set) becomes larger than r_{max} and so the reconstruction begins to fail if n drops below some point. Thus it is important to be aware of the point when the sample set starts limiting the neighbourhood of the x_j around which θ_j values are non-zero in enough quantity that solving the MLS system for $WAc = Wh$ does not output garbage.

REFERENCES

[1] Weisstein, Eric W. “Least Squares Fitting–Polynomial.” From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html

[2] Lee, Do Q. “Numerically Efficient Methods For Solving Least Squares Problems” (2012)
https://www.rose-hulman.edu/ bryan/lottamath/leastsqrs.pdf

[3] Epperson, James F. “On the Runge Example” University of Georgia, Athens 1987 Available at
https://www.maa.org/sites/default/files/pdf/upload_library/22/Ford/Epperson329-341.pdf

[4] Alexa, Mark et. al "Computing and Rendering Point Set Surfaces" TU Darmstadt, Tel Aciv University, ATT Labs 2003 Available at
http://ieeexplore.ieee.org/document/1175093/

APPENDIX

LIST OF FIGURES

LIST OF TABLES