Computational Linear Algebra Coursework 1

George Hilton CID: 01494579

This document provides very detailed explanations for the written aspects of the coursework and includes all the plots produced by files in the repository. Some of the most important figures which are tabulated in the console have also been included; there are a vast number and don't serve much purpose being in the document but, of course, they can be viewed by running the code in the repository.

In the repository I have broken the coursework down into multiple files: q1.py, q2.py, q3.py, q4.py which implement the algorithms for each question and produce any necessary plots or prints; q1func.py, q2func.py, q3func.py, q4func.py which contain all the functions used to implement the algorithms in each question, these are called upon in the prior mentioned files and, test_q2c.py, test_q4c.py, the test files, written using PyTest which can be ran from the console.

1

a

Implemented in q1.py using q1func.py, I used, Modified Gram Schmidt as it is efficient and stable.

b

Both the matrices are very large, in particular Q, so I felt that plotting heat maps would provide more information than attempting to manually observe the values. As Q is extremely large I took 3 heat maps using all the columns and subsections of the rows with length 100, from the top, middle and bottom. The heat maps of Q (Fig: 1, Fig: 2, Fig: 3) reveal little about the matrix but, looking at R (Fig: 4), there are several interesting things. It appears that all but the first four rows of R are zero (not exactly equal due to rounding errors).

Due to the structure of R, we know that the reduced form of Q, $Q^\dagger$, is the first four columns of Q and the remaining columns are the nullspace, N. So Q

can be considered as follows

$$Q = (Q^\dagger | N).$$

c

As $Q^\dagger$ is made up of four columns, we know that these four columns span the column space of A. Thus, the rank of A is 4.

d

Let's also denote the first four columns of Q as q_1, q_2, q_3, q_4 and the i^{th} element of column j as $Q_{i,j}$.

Now, as the column space of A is spanned by the first four columns of Q, any column of A, say a_k, can be written as a linear combination of the first four columns of Q, i.e. $a_k = \sum_{i=1}^{4} \mu_i q_i, \quad \mu_i \in \mathbb{R}$.

If we consider the element in row i, column j of A, this can be written as $f_i(x_j)$ (from information given in the question, x_j is some pre-image). From our previous explanation we know that this element can be expressed as a linear combination of the elements in the i^{th} row of $q_1, ..., q_4$, i.e. $f_i(x_j) = \sum_{j=1}^{4} \lambda_j Q_{i,j}$, $\lambda_j \in \mathbb{R}$. So, $\{Q_{i,1}, ..., Q_{i,4}\}$ is a basis for $f_i(x_j)$ but, as i and j were arbitrary throughout, this holds $\forall i$. This also reveals that the dimension of the image of the functions used to create A, is four.

2

a

We have the problem that $Ax \approx b$ and want to find x such that the equation is as close to equality as possible. In lectures it was conjectured that to do so, we would solve the system $Rx = Q^* b$

Proof. Let $A = QR$ be the QR decomposition of A. Let $P = QQ^*$ then we know $Pv \perp (I - P)v$ for any vector v.

Also, we have $\|v\|_2^2 = \|Pv\|_2^2 + \|(I-P)v\|_2^2$.

Now, observe that

$$\begin{aligned}\|b - Ax\|^2 &= \|b - QRx\|^2 \\ &= \|(I - QQ^*)b + Q(Q^*b - Rx)\|^2 \\ &= \|I - QQ^*b\|^2 + \|Q(Q^*b - Rx)\|^2 \\ &= \|I - QQ^*b\|^2 + \|Q^*b - Rx\|^2 \\ \implies \min_x \|b - Ax\|^2 &= \|I - QQ^*b\|^2\end{aligned}$$

This occurs when $Rx = Q^*b$. □

b

From the information given, we wish to formulate a linear system relating polynomial coefficients and entries in the vector f. So, we want the rows of the L.H.S. to be $F(x_i)$ and the R.H.S. to be f_i. The system below does just that.

$$\begin{bmatrix} x_1^0 & \dots & x_1^m \\ x_2^0 & \dots & x_2^m \\ \vdots & \ddots & \vdots \\ x_{11}^0 & \dots & x_{11}^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{11} \end{bmatrix}$$

c

After running the code from q2func.py and q2.py in the repository we see the coefficients $a_0, ..., a_{10}$ tabulated in the console (Fig: 26) and a graph of the polynomial (Fig: 5) with the original data points marked on. We have a 10^{th} degree polynomial with non-zero coefficients which cleanly passes through all the data points. To check the polynomial is correct I added a test function to the repository, test_q2c.py. This checked whether the norm of my polynomial, $p(x)$, applied element wise to x, minus f was less than 1.0e-6, i.e. whether it was zero if we ignore rounding errors. Running the test in the console gives a pass and so, it is clear that the polynomial is interpolating the points accurately.

d

For this part I had to perturb the f_i values which we were initially given. Running the code in q2.py there are multiple realisations of the function with a variety of changes in the variables. In the console, tables of coefficients and distance between the coefficient vectors are printed and graphs are plotted with the original $p(x)$ and the perturbed version.

There are several interesting conclusions which we can draw. Firstly, in the case where we perturb only one of the values (Fig: 6), we see there is little to no affect when the perturbation is the machine epsilon. Even as the perturbations get much larger we see, from the plot and coefficients, that only the first portion of the graph has been perturbed (the norm output is less than one for all but the final case). Considering the case when we perturb 3 values (Fig: 7), we observe that this is more sensitive and, by $dx = 2^{-5}$, the norm difference value is 8 and, by $dx = 2$ it is over 100, with significant deviation from the original plot. Importantly, from the graphs we can see that as the number of points we perturb increases, the further along the x-axis the deviations travel. Perturbing 5 points (Fig: 8) provides the greatest perturbation and although it is still insignificant - like the others - near the machine epsilon value; we see vast difference across the full range of the graph when $dx = 2$. Now, as the number of perturbed points increases further (Fig: 9), we see a decrease in the 'level' of deviation. As, when we perturb lots of points, there is a form of a vertical shift of the original graph - this type of deviation is easier to visualise and understand. The case when $j = 11$ (Fig: 10) provides a very clear demonstration of this.

e

Here, I had to complete the same task as 2(c) but, this time interpolate with a polynomial of degree 7. In this situation - as there are more data points than variables - it is essential to use the least squares algorithm as it is unlikely there will be a polynomial able to fit the data perfectly. Instead, it is much more likely that our polynomial will be 'very close' to each of the data points. Running the code in q2.py a graph is plotted (Fig: 11) which has the original polynomial, $p_{10}(x)$ and our new polynomial of degree 7, $p_7(x)$ as well as the data points. In addition, the coefficients are tabulated in the console (Fig: 27).

This graph highlights exactly what we would expect: $p_7(x)$ is extremely close to the data points but actually passes through very few. It is important to note, that in some situations a lower degree polynomial is preferable. In this case we simply want to fit a curve to points however, if we wanted to plot a regression curve for real world data and make predictions $p_7(x)$ would be optimal. The shape of $p_{10}(x)$ with very high peaks between points suggests overfitting and it is much more likely that $p_7(x)$ would be optimal for extrapolation.

f

The next task was to investigate the affect of perturbing the f_i but in the case with $m = 7$.

I used the same function as the previous case but changed m to 7. We see a similar trend, for small values (Fig: 12), even much greater than the machine epsilon value, there is little to no deviation - particularly in the cases where fewer variables are changed. Once again we see a similar rise in the deviation as we increase the number of variables that are changed (Fig: 13); then, after 3, there is a gradual decrease (Fig: 14). There is a difference however, when we saw a progressive increase in the range of x which was affected in the $m = 10$ case, with $m = 7$ there is less uniformity and from the graphs, we see there is repeated vertical deviation downwards (Fig: 15). Unsurprisingly, there is much reduced overall deviation for the same values e.g. in the case when $j = 3$, $dx = 2$ we have a norm difference of 532.64... in the $m = 10$ case and a norm difference of 35.02... in the $m = 7$. Most likely this comes as a result of the overfitting with the degree 10 polynomial; leading to greater inaccuracies with perturbations as the approximation is more unstable.

3

a

The first 3 plots (Fig: 16) generated by q3.py are to be used for analysis in 3a. We can see that for $M = 20, 50, 200$ the shape of each of the graphs is identical. I use the reduced version of Q: using Python slice notation to select all columns and the same number of rows as in R. This ensures we don't use the nullspace of Q and can analyses the information more accurately. Usefully, we also recognise that increasing M, in this circumstance, only acts to increase the number of data points when plotting the column graphs and so, we fix a large value of M in the next part. Looking at the lines plotted on the graphs we see they take the form of constant, linear, quadratic, cubic, quartic and quintic curves. From the construction of the Vandermonde matrix (which we performed QR decomposition on) we know that the first column consists of 1's; the second column is formed form equispaced points, the third is formed from equispaced points raised to the second power and so on, until the sixth column which is formed by equispaced points to the 5^{th} power. Due to the nature of Householder Reflectors, they will preserve and continue this structure into Q. Thus, we see that the first column of Q is constant, the second is linear, then quadratic, cubic, quartic and quintic which gives us the shapes seen in the plots.

b

For the next section I am investigating the comparison between plots of the last 5 columns of the QR factorisation of V when using Classical and Modified Gram Schmidt as well as Householder. I have fixed $M = 200$ as I previously mentioned. In the code there are plots for $p = 5, 15, 25, 50, 150$. For the case when $p = 5$ (17) we see the graphs are very similar between householder and GS (Gram-Schimdt), with the only difference being sign differences. This stems from the differences in the way Q is constructed for each method. When $p = 15$ (18) there is slight deviation between the Householder plot and the Gram-Schmidt plots; however, a similar oscillatory nature is present in both. The differences only arise due to the alternative decomposition methods. Same as before we see multiple high order polynomials and a reflection difference between Householder and GS due to sign changes. For $p = 25$ (19) this is the beginning of noticeable and important differences between the plots. Householder and Modified Gram-Schmidt have retained the same structure as before but, in this case, the Classical Gram-Schmidt algorithm appears to be plotting some repeats of curves. We observe that the Classical Gram-Schimdt algorithm, as expected, is very unstable and 'breaks' for larger matrices. The plots for $p = 50$ (20) and $p = 150$ (21) reinforce this trend: Householder and Modified Gram-Schmidt are plotting very oscillatory graphs, whereas the unstable Classical Gram Schmidt is unable to work correctly on much larger matrices. In these cases, we see that there are only two visibly distinct curves on the Classical Gram-Schmidt plot.

4

a

We can formulate the problem defined in the question as a least squares problem of the form

$$\min_{Bx=d} \|Ax - b\|^2.$$

Some notation:

$$Z = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix} = \text{Depth values}, \qquad b = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} = \text{Pressure values}.$$

To begin we must first break down, and understand the question. The first step is recognising that we are using polynomial interpolation to fit polynomials to the data. It states that the relationship between depth and pressure is different for depths less than or equal to one and depths greater than one. From this we can see that we need to find two polynomials to model this situation: $p_1(z), p_2(z)$ for $z \leq 1$ and $z > 1$ respectively with, $p_j(z_i) = y_i$.

$$\begin{aligned} p_1(z) &= a_0 + a_1 z + a_2 z^2 + ... + a_p z^p \qquad && z \leq 1 \\ p_2(z) &= b_0 + b_1 z + b_2 z^2 + ... + b_p z^p \qquad && z > 1 \end{aligned}$$

From this we can construct two linear systems, then bring them together as we did in question 2.

$$\begin{bmatrix} z_1^0 & \dots & z_1^p \\ z_2^0 & \dots & z_2^p \\ \vdots & \ddots & \vdots \\ z_r^0 & \dots & z_r^p \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_r \end{bmatrix}, \quad \begin{bmatrix} z_{r+1}^0 & \dots & z_{r+1}^p \\ z_{r+2}^0 & \dots & z_{r+2}^p \\ \vdots & \ddots & \vdots \\ z_n^0 & \dots & z_n^p \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_p \end{bmatrix} = \begin{bmatrix} y_{r+1} \\ y_{r+2} \\ \vdots \\ y_n \end{bmatrix}$$

Where z_r is the depth value equal to one.

Now set the above matrices to $A^{\dagger}$ and $A^{\dagger\dagger}$ respectively and

$$A = \begin{bmatrix} \mathrm{A}^{\dagger} & 0 \\ 0 & \mathrm{A}^{\dagger\dagger} \end{bmatrix}, \quad x = \begin{bmatrix} a_0 \\ \vdots \\ a_p \\ b_0 \\ \vdots \\ b_p \end{bmatrix}$$

Now, we have our least squares system that we want to solve

$$Ax = b.$$

We need to consider the constraints on the question. It states that the pressure is continuous at a depth equal to one and that there is a jump in derivative at a depth of one. These can be interpreted through several equations

$$\begin{aligned} \text{Continuity:} \quad & p_1(1) - p_2(1) = 0 \\ \text{Derivative:} \quad & p_2^{'}(1) - p_1^{'}(1) = 5. \end{aligned}$$

Defining

$$B = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & -1 & -1 & \dots & -1 \\ 0 & -1 & -2 & \dots & -p & 0 & 1 & 2 & \dots & p \end{bmatrix}, \quad d = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

we can represent this system of equations in matrix-vector form as

$$Bx = d.$$

So we have a system to solve by least squares, $Ax = b$ and constraints which we need to enforce, $Bx = d$. Hence, the problem can be represented as

$$\min_{Bx=d} \|Ax - b\|^2.$$

b

We have a new situation now, $\min_{Cy_1=d,y_2} \|A_1y_1 + A_2y_2 - b\|^2$, under some transformation $Q^Tx = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$. We can convert this new problem into matrix vector form as follows and then manipulate for Q.

Firstly, note that $\begin{bmatrix} B \\ A \end{bmatrix}$x $= \begin{bmatrix} d \\ b \end{bmatrix}$ then we can write,

$$\begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix} x$$

$$\implies \begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix} Q^T x = \begin{bmatrix} B \\ A \end{bmatrix} x$$

$$\implies \begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix} Q^T = \begin{bmatrix} B \\ A \end{bmatrix}$$

$$\implies \left(\begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix} Q^T \right)^T = \begin{bmatrix} B \\ A \end{bmatrix}^T$$

$$\implies Q \begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix}^T = \begin{bmatrix} B \\ A \end{bmatrix}^T$$

So, if we take the QR decomposition of $\begin{bmatrix} B \\ A \end{bmatrix}^T$ then we get the desired Q and $R = \begin{bmatrix} C & 0 \\ A_1 & A_2 \end{bmatrix}^T$. We can write C explicitly as the 2x2 matrix contained in the upper left corner of R^T.

c

Our problem, $\min_{Cy_1=d,y_2} \|A_1y_1 + A_2y_2 - b\|^2$, can be solved by first computing y_1 and then y_2. We have values for A, B, d, b and want to find x.

We compute the QR factorisation of $\begin{bmatrix} B \\ A \end{bmatrix}^T$. Then obtain C as mentioned above. C is upper triangular so we can easily solve the system $Cy_1 = d$ for y_1.

Taking appropriate slices of R^T we obtain A_1 and A_2 and aim to solve $\min_{y_2} \|A_2y_2 - (b - A_1y_1)\|^2$. We use the householder least squares function to solve $A_2y_2 = b - A_1y_1$ for y_2.

Now all we need to do is compute $x = Q \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$.

This was implemented in the q4.py using q4func.py for several values of p. The code plots the polynomials with the original data points overlayed and in the console, it prints the coefficients of the polynomials and the norm of $p(z)$ element wise minus y to measure deviation (Fig; 28). I also wrote test_q4c .py which is a test file, using PyTest that runs from the console. It compares the norm of errors for the approximation to the data points. Testing $p_1(x), p_2(x), \begin{bmatrix} p_1(x) \\ p_2(x) \end{bmatrix}$, we see passes in all cases with very low tolerances: we know from these tests that the curves are a very accurate approximation to the data points.

In addition, from the plots (Fig: 22, Fig: 23, Fig: 24) we also see that the curves give good approximations to the data points for both regions: depth ≤ 1 and depth > 1. The norm values are all relatively small (< 2) but, smallest for $p = 13$, so this curve most closely fits the data points. However, we can see in all the plots for $p > 5$ there is a sharp curve downwards at the right end - this is due to overfitting and suggests the polynomials could not be extrapolated well. In addition, for all cases, if we look at the coefficients, they are very large (some of order 10^{10}) and so, if we extend the range of the polynomial plot (Fig: 25) it blows up very rapidly. Hence, for this data set, the polynomials fit the data given very well, but would be very poor and making predictions for even greater depths.

d

For part d, I struggled to implement the theory: I think this is partly due to a lack of coherence with my formulation of the problem. I will express the ideas I did have.

We have a similar situation to 4a in the sense that we want to solve $\min_{Bx=d} \|Ax - b\|^2$. We can formulate the least squares aspect as

$$
\begin{aligned}
& x = \hat{C}(\theta) = a_0 + a_1\theta + ... + a_p\theta^p \quad 0 < \theta < \frac{2\pi}{M} \\
& \vdots \\
& x = \hat{C}(\theta) = l_0 + l_1\theta + ... + l_p\theta^p \quad \frac{2(M-1)\pi}{M} < \theta < 2\pi
\end{aligned}
$$

From this we can construct A and b such that $Ax = b$ (we are trying to solve for the coefficients)

$$A = \begin{bmatrix} V_0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & V_{M-1} \end{bmatrix}, \quad b = \begin{bmatrix} a_0 \\ \vdots \\ a_p \\ b_0 \\ \vdots \\ l_p \end{bmatrix}.$$

Where each V_i is the Vandermonde matrix corresponding to the system defined above, in the range $\frac{2i\pi}{M} < \theta_i < \frac{2(i+1)\pi}{M}$ for $i = 0, 1, ...M-1$.

Like 4(a) we now need to bring in the continuity and derivative constraints at each boundary. After some calculation we can write the continuity restraint for the V_i section as

$$\begin{bmatrix} 1 & \frac{2\pi(i+1)}{M} & \dots & \left(\frac{2\pi(i+1)}{M}\right)^p & -1 & \frac{-2\pi(i+1)}{M} & \dots & \left(\frac{-2\pi(i+1)}{M}\right)^p \end{bmatrix} \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_p \\ \lambda_0 \\ \vdots \\ \lambda_p \end{bmatrix}.$$

Similarly, for the derivative restraint

$$\begin{bmatrix} 0 & 1 & 2\left(\frac{2\pi(i+1)}{M}\right) & \dots & p\left(\frac{2\pi(i+1)}{M}\right)^{p-1} & 0 & -1 & 2\left(\frac{-2\pi(i+1)}{M}\right) & \dots & p\left(\frac{-2\pi(i+1)}{M}\right)^{p-1} \end{bmatrix} \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_p \\ \lambda_0 \\ \vdots \\ \lambda_p \end{bmatrix}.$$

It was at this point that I struggled to bring together the constraints for each Vandermonde matrix into more succinct matrix vector form. As such, this is regrettably as far as I could progress with 4(d).

5 Gallery

Figure 1: Heat Maps of subsections of Q

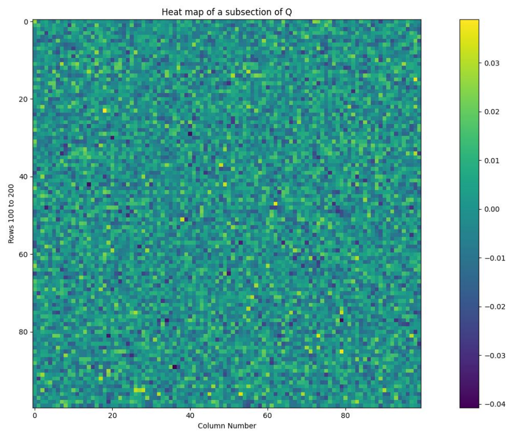


Figure 2: Heat Maps of subsections of Q

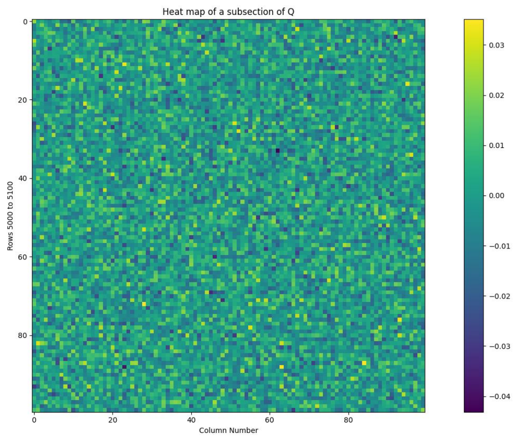


Figure 3: Heat Maps of subsections of Q

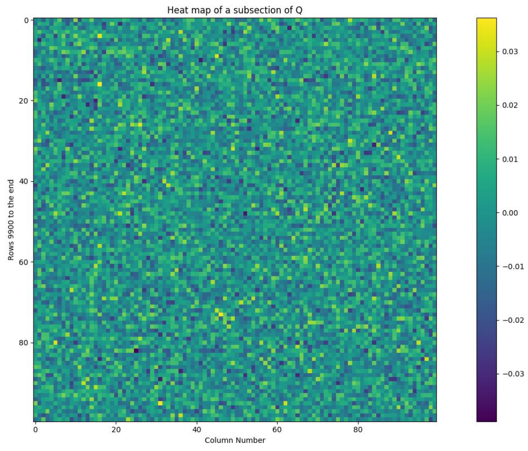


Figure 4: Heat Map of R

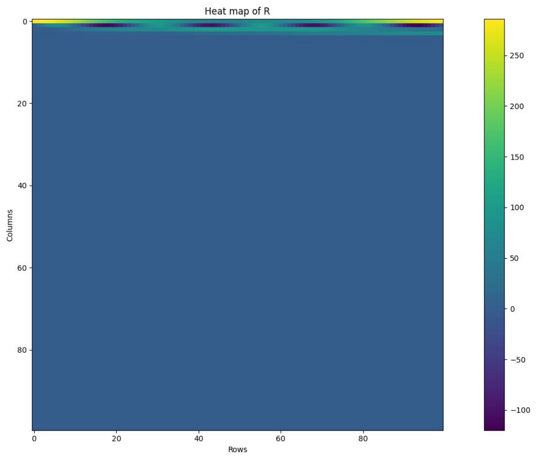


Figure 5: Plot of p(x) with data points

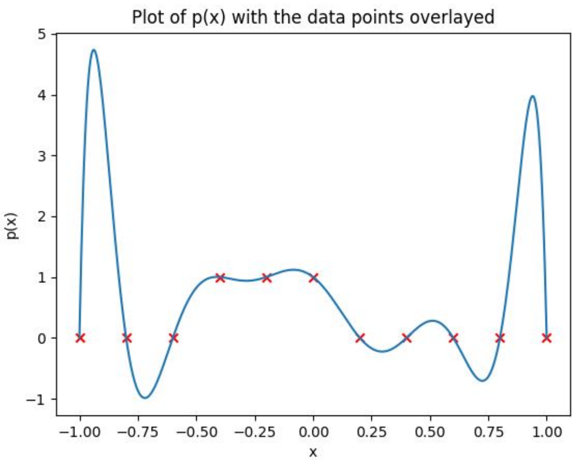


Figure 6: Comparison of plots with perturbations, j=1, m=10

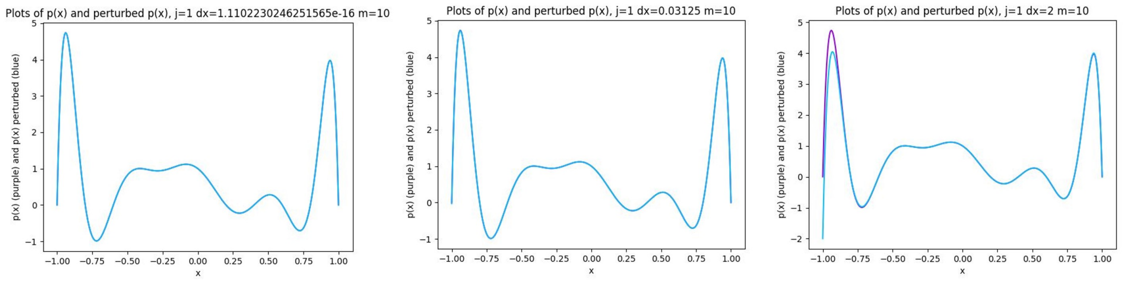


Figure 7: Comparison of plots with perturbations, j=3, m=10

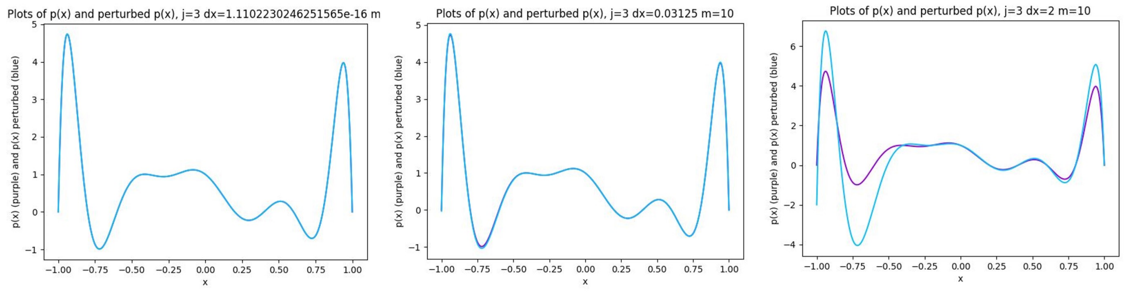


Figure 8: Comparison of plots with perturbations, j=5, m=10

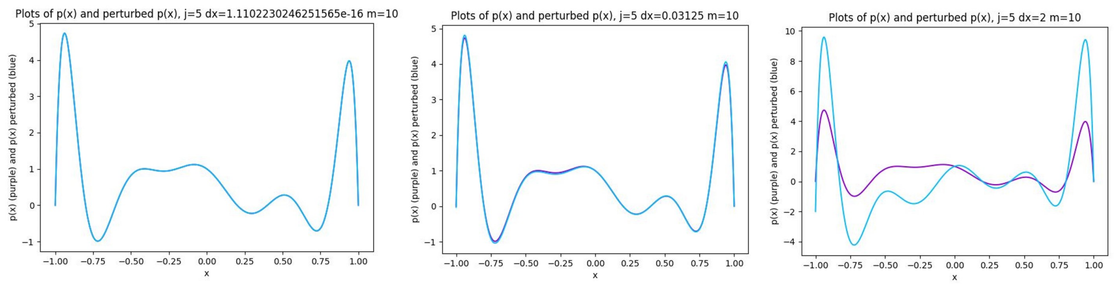


Figure 9: Comparison of plots with perturbations, j=7, m=10

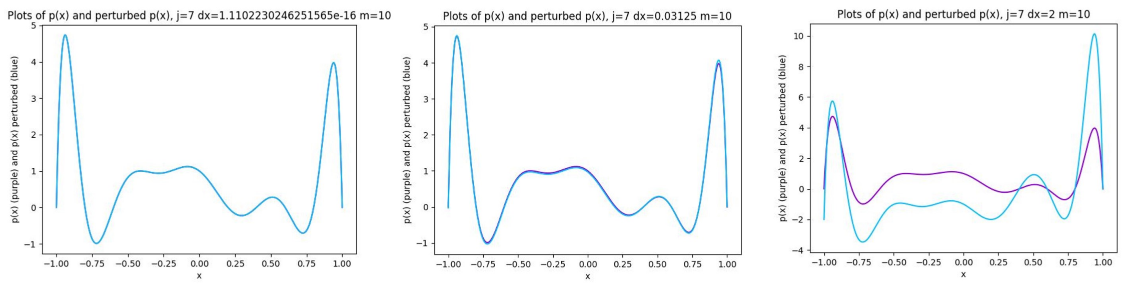


Figure 10: Comparison of plots with perturbations, j=11, m=10

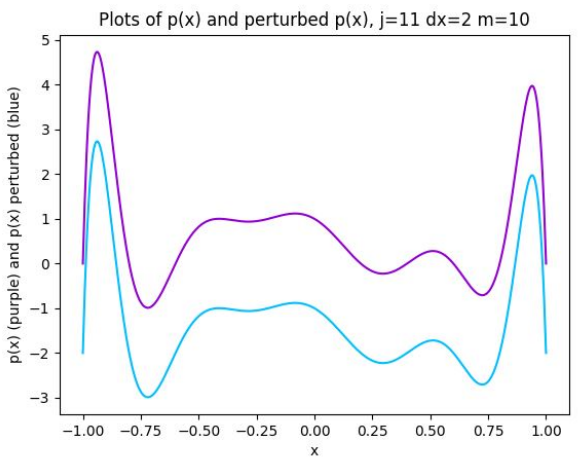


Figure 11: Comparison of plots with m=7, m=10

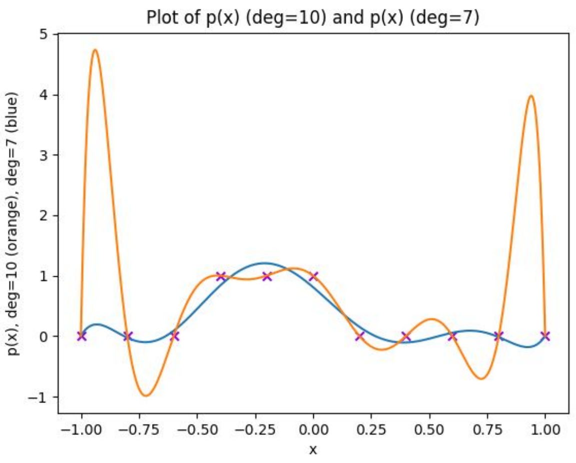


Figure 12: Comparison of plots with perturbations, j=1, m=7

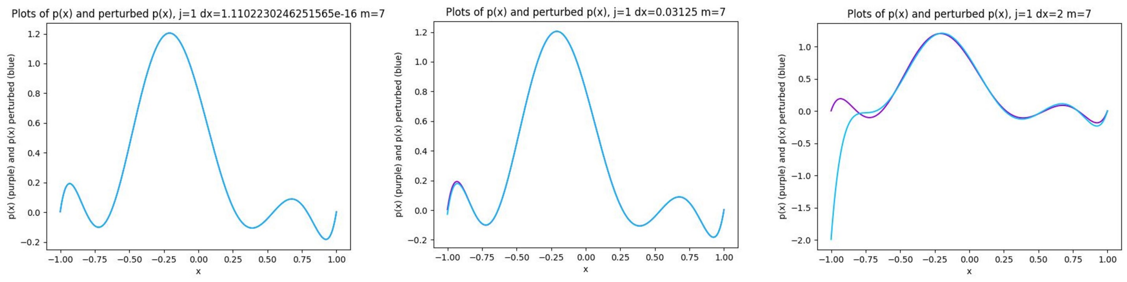


Figure 13: Comparison of plots with perturbations, j=3, m=7

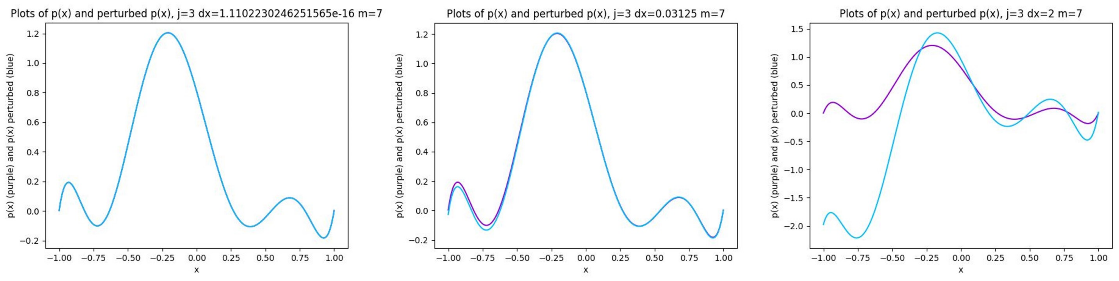


Figure 14: Comparison of plots with perturbations, j=5, m=7

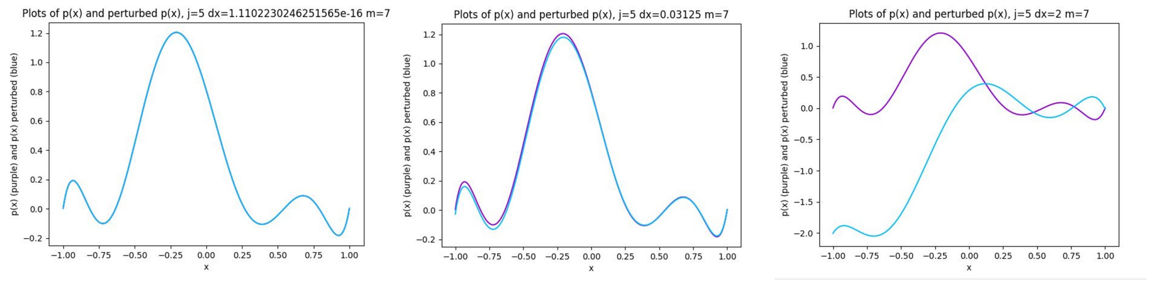


Figure 15: Comparison of plots with perturbations, j=7, m=7

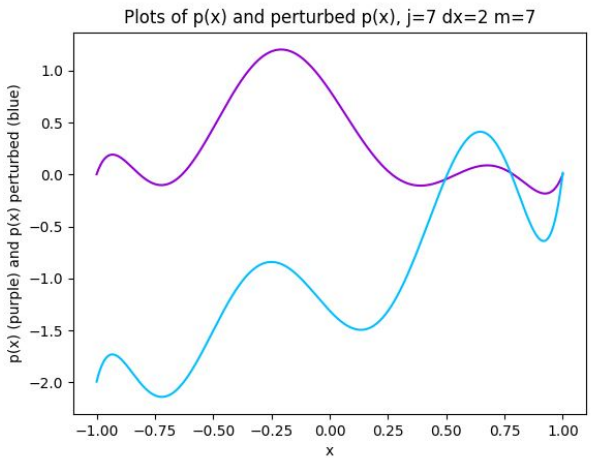


Figure 16: Householder plots for p=5 and varying M

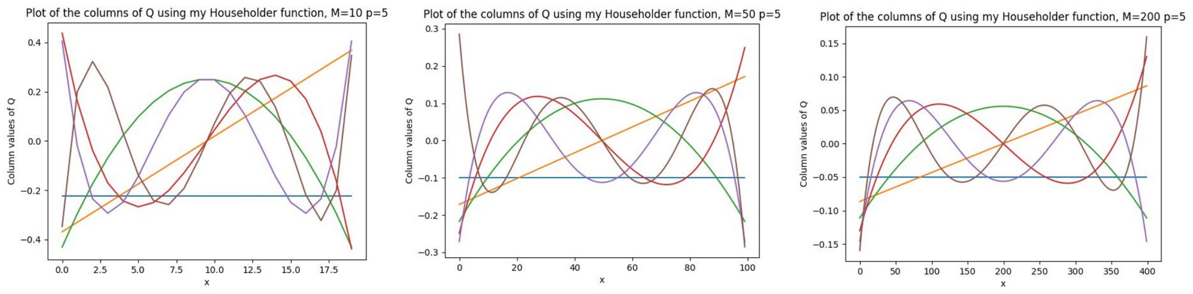


Figure 17: Comparison of Q plots, p=5

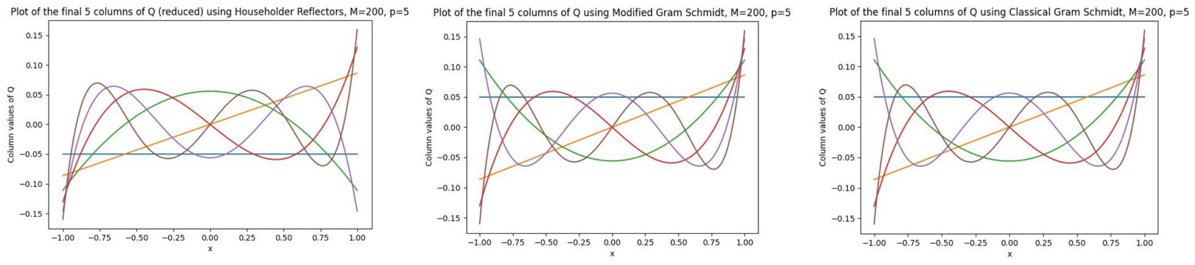


Figure 18: Comparison of Q plots, p=15

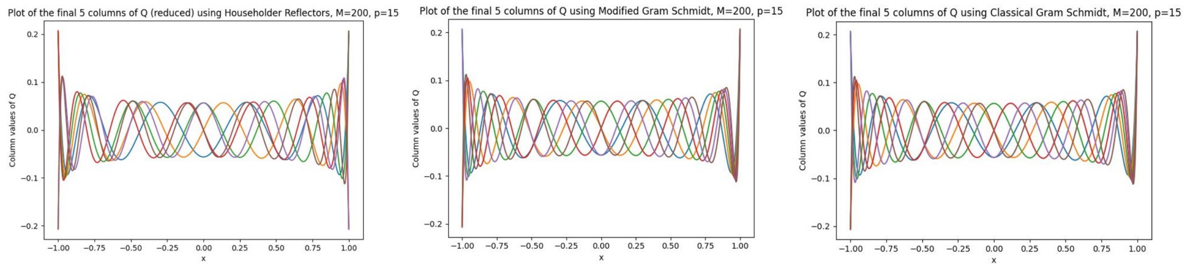


Figure 19: Comparison of Q plots, p=25

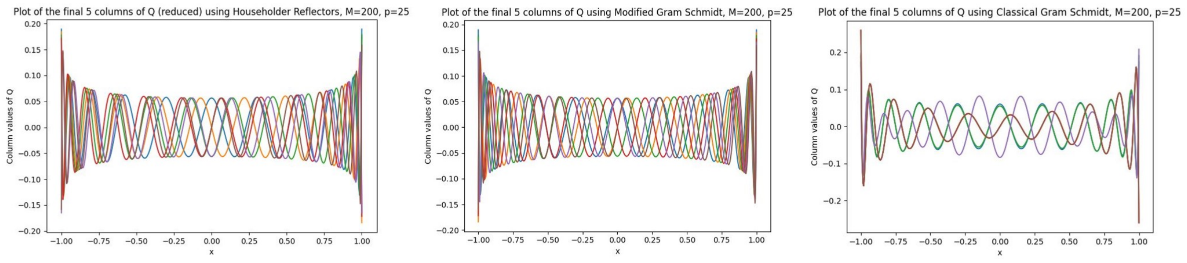


Figure 20: Comparison of Q plots, p=50

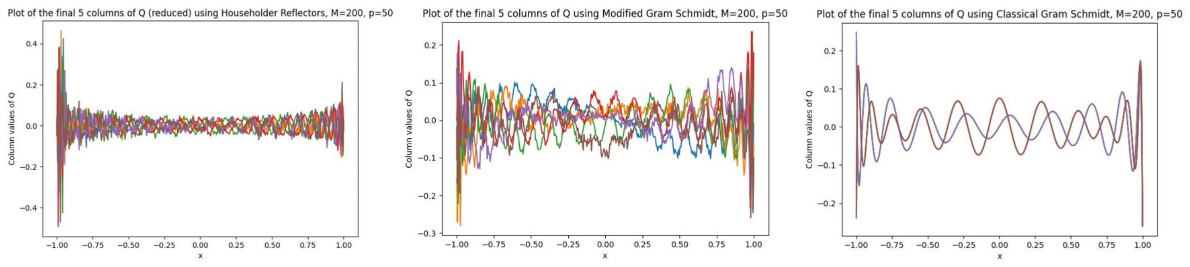


Figure 21: Comparison of Q plots, p=150

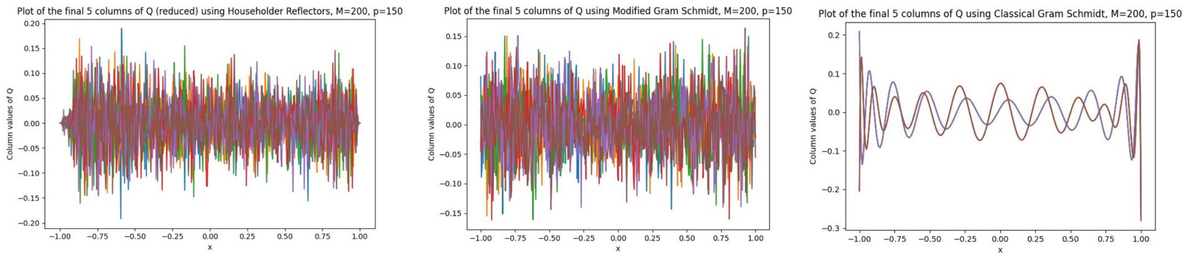


Figure 22: Polynomial plots with p=5 and p=8

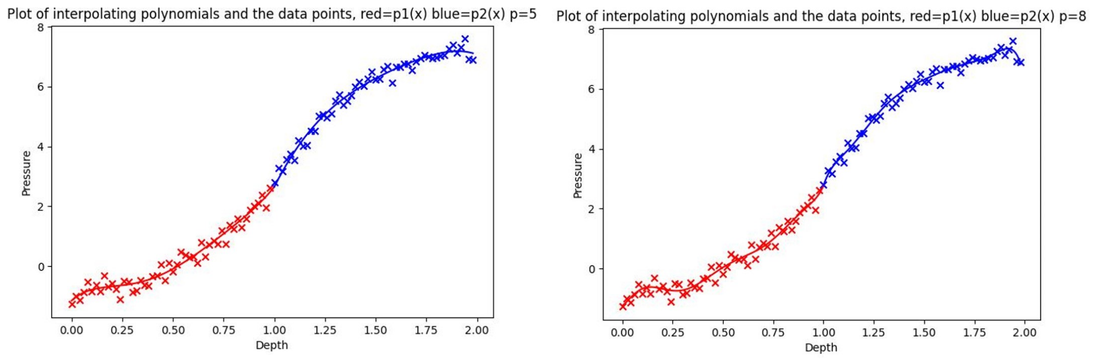


Figure 23: Polynomial plots with p=10 and p=13

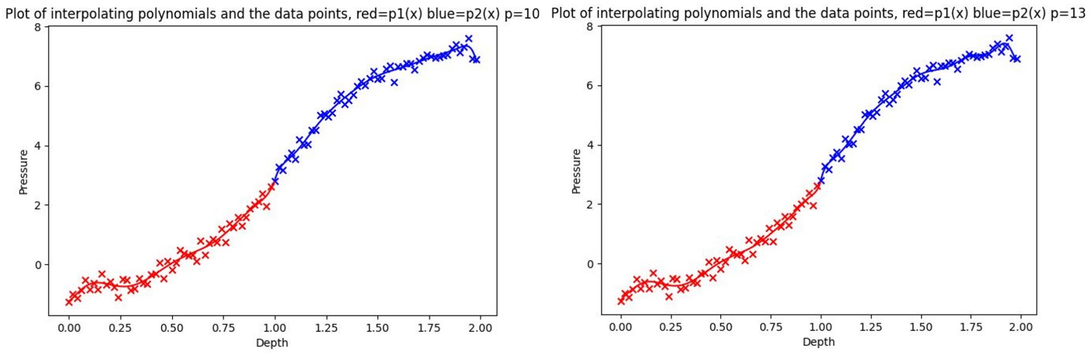


Figure 24: Polynomial plots with p=15

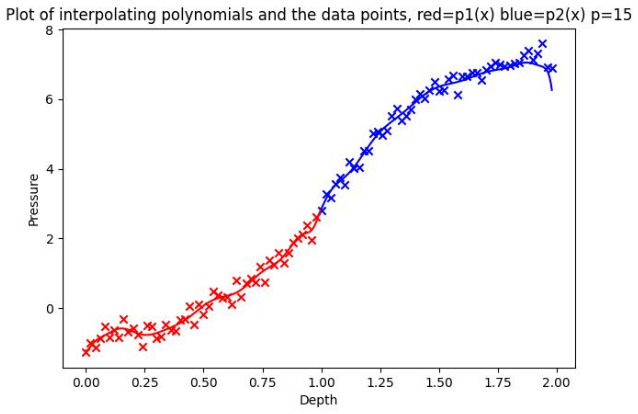


Figure 25: Polynomial plots with p=15

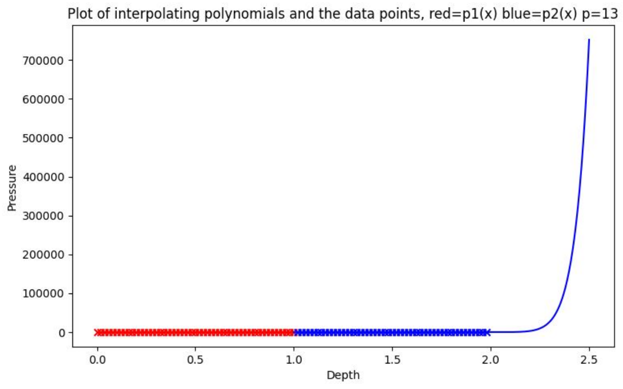
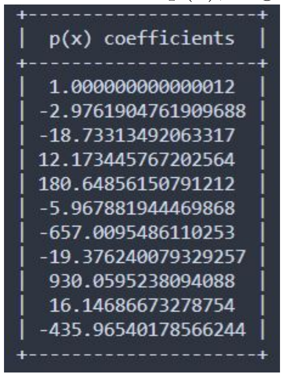
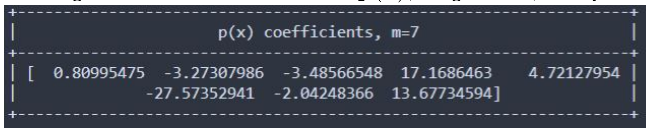


Figure 26: Coefficients for p(x), degree=10, in Q2



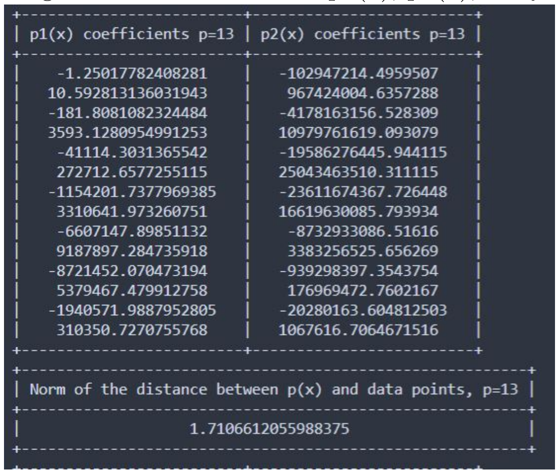
```
+---------------------+
|  p(x) coefficients  |
+---------------------+
|  1.000000000000012  |
| -2.9761904761909688 |
| -18.73313492063317  |
| 12.173445767202564  |
| 180.64856150791212  |
| -5.967881944469868  |
| -657.0095486110253  |
| -19.376240079329257 |
|  930.0595238094088  |
|  16.14686673278754  |
| -435.96540178566244 |
+---------------------+
```

Figure 27: Coefficients for p(x), degree=7, in Q2



```
+-----------------------------------------------------------------------+
|                        p(x) coefficients, m=7                         |
+-----------------------------------------------------------------------+
| [  0.80995475  -3.27307986  -3.48566548  17.1686463    4.72127954 |
|            -27.57352941  -2.04248366  13.67734594]                    |
+-----------------------------------------------------------------------+
```

Figure 28: Coefficients for p1(x), p2(x), in Q4



```
+-------------------------+-------------------------+
| p1(x) coefficients p=13 | p2(x) coefficients p=13 |
+-------------------------+-------------------------+
|    -1.25017782408281    |    -102947214.4959507   |
|   10.592813136031943    |     967424004.6357288   |
|   -181.8081082324484    |    -4178163156.528309   |
|   3593.1280954991253    |    10979761619.093079   |
|    -41114.3031365542    |   -19586276445.944115   |
|    272712.6577255115    |    25043463510.311115   |
|   -1154201.7377969385   |   -23611674367.726448   |
|    3310641.973260751    |    16619630085.793934   |
|    -6607147.89851132    |     -8732933086.51616   |
|    9187897.284735918    |     3383256525.656269   |
|   -8721452.070473194    |    -939298397.3543754   |
|    5379467.479912758    |     176969472.7602167   |
|   -1940571.9887952805   |    -20280163.604812503  |
|    310350.7270755768    |    1067616.7064671516   |
+-------------------------+-------------------------+
+-------------------------------------------------------+
| Norm of the distance between p(x) and data points, p=13 |
+-------------------------------------------------------+
|                   1.7106612055988375                  |
+-------------------------------------------------------+
```