Computational Linear Algebra Project 1

Jonathan Cheung
CID: 01063446
Lecturer: Professor Colin Cotter
The contexts of this report and associated
codes are my own work unless otherwise stated and
references to sources must be provided if they are used.

5th November 2019

1 QR factorisation by Householder reflections

The script 'CLA_P1_Q1.py' contains a function 'householderQR' (line 14) that takes as its input an m x n matrix A and returns Q and R (labelled b and A) such that A is equal to the product of Q and R. The algorithm is an in place algorithm, so there is also a third output (A_copy), which is a copy of the original input matrix A. This is useful for when we later check that the Q and R output multiply back to form the original input

Running script 'CLA_P1_Q1_tests.py' will execute a series of tests to check that the function is behaving properly. The script contains a function 'QRtester' (line 14) which will make 3 checks. It checks that QQ* = I = Q*Q (I_test). It also checks that the the matrix R is upper triangular (uptritest). Lastly, it checks that Q multiplied by R is equal to A. In all these tests I have used np.allclose since working with floating point numbers means that the difference between QR and the original input A will be some number approximately 10^{-17}. np.allclose will return true if the difference between A and QR is less than 10^{-5}.

In line 11 the rectangular matrix 'test1' is defined and running 'QRtester(test1)' (line 48) will output True since it has passed all three of the previously mentioned checks. The matrix 'test2' is defined in line 50. Unfortunately, the householderQR function does not work correctly on test2. Running QRtester on test2 (line 51) will return False. In fact householderQR does not work on matrices with complex numbers. It will generate a Q such that QQ*= I = Q*Q. However, it will not create an upper triangular R. QR is also marginally different to A. Given more time, this would be something I would like to fix.

Nevertheless, we can test the function for a number of randomly generated

matrices with random dimensions (lines 53-59). The array 'bulktests' contains the test results for ten randomly generated matrices, with 1s representing True and 0s representing False. Line 59 prints true if every matrix passed the tests.

2 Polynomial fitting by least squares

Given a number of data points which contain light intensity in one column and temperature change in the other, we can fit a polynomial of arbitrary degree to this data using a least squares approach. Using the light intensity data, we can generate a Vandermonde matrix A. To find the coefficients of the polynomial that fits this data, we need to solve Ax = b for x, where b is the temperature change data. A can be factorised into QR. So, $\hat{Q}\hat{R}x = b$ and $\hat{Q}^*\hat{Q}\hat{R}x = \hat{Q}^*b$, where $\hat{Q}$ and $\hat{R}$ are the reduced QR decomposition and $\hat{Q}^*$ is the hermitian conjugate of $\hat{Q}$. So, if we can take the reduced QR decomposition of A, we can solve Ax=b by solving $\hat{R}x=Q^*b$. Thus, we will find a polynomial that interpolates the data.

The script 'CLA_P1_Q2.py' contains a function 'LSE' (line 35) which will fit a polynomial for the readings.csv data. It returns x, r, r_2norm, and poly. These are, respectively, the coefficients that solve $\hat{R}=\hat{Q}^*b$, a vector of residues, the 2-norm of this residue vector, and the polynomial function that fits the data. This function LSE makes use of the 'householderQR' function from Q1.

The code in lines 103-110 creates the following graph.

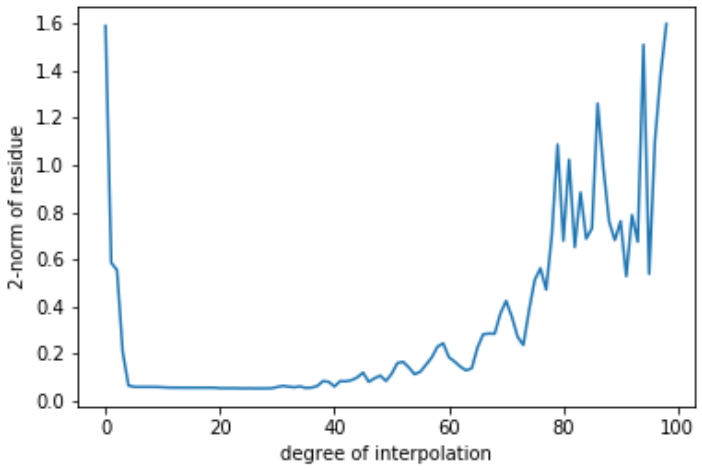


Figure 1: The accuracy of the interpolating function for different degrees

This graph shows how the 2-norm of the residue changes as the degree varies from 0 to 99. We can in fact take a closer look at the behaviour for degrees 0 to 8.

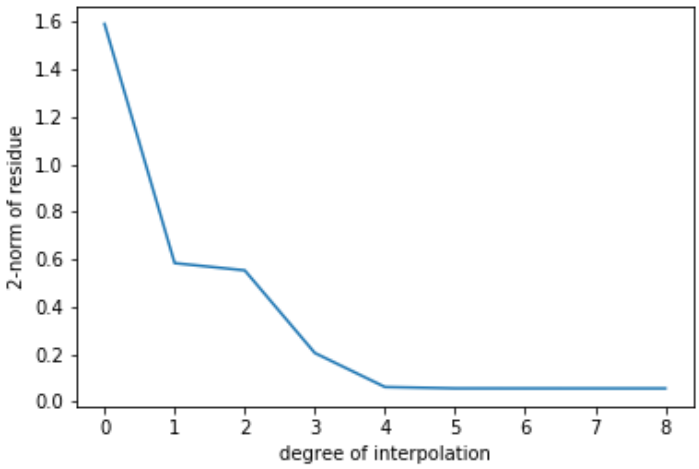
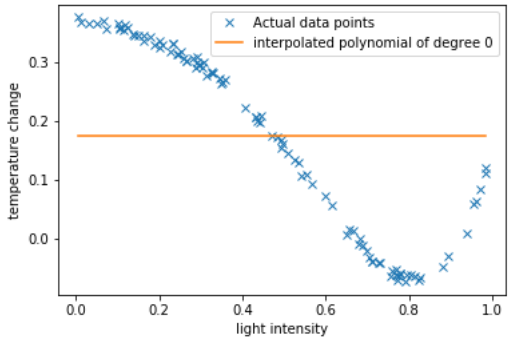


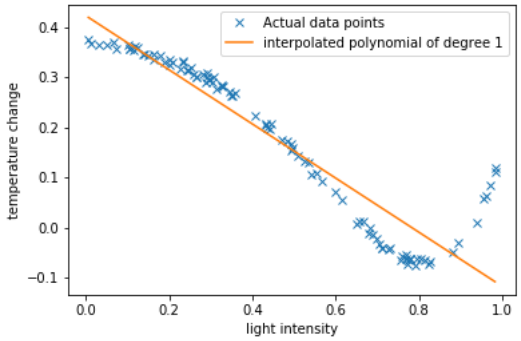
Figure 2: The accuracy of the interpolating function for small degrees

The residue is largest for degree 0. This makes sense since we do not have fixed temperature change values across all the data points. As degree increases to 1, the residue decreases significantly since a degree 1 polynomial can at attempt to fit the data to some non constant line, unlike a degree 0 polynomial. There is not much decrease from degree 1 to 2, but a large decrease from 2 to 3. The last significant decrease is from degree 3 to 4. After this point, the residue does decrease, but by very small amounts. We can check this by running line 112 of this script (CLA_P1_Q2.py), which will display the first 9 residues, which we can clearly see is decreasing monotonically. Lines 114-116 will help us find the degree that produces the smallest residue value. This value is 27. However, it is very unlikely that the polynomial that best fits the data is of degree 27. In fact, it is very likely that the degree 26 (27th entry in the array) polynomial is overfitting the data.

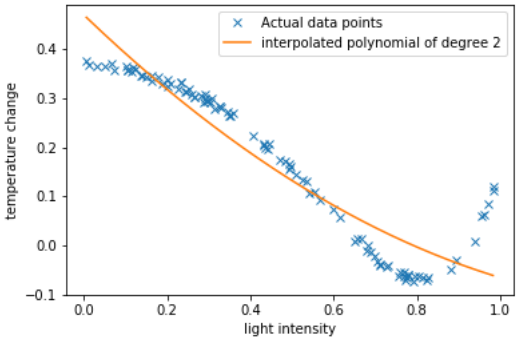
Here are some graphs to support some of these conclusions.



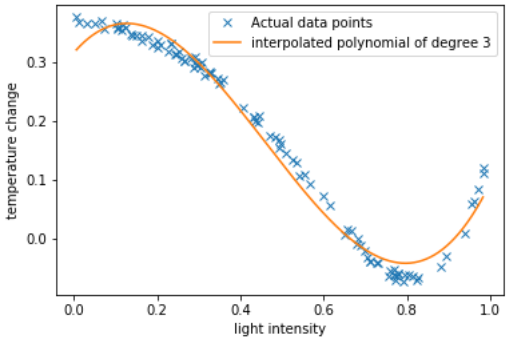
(a) degree 0



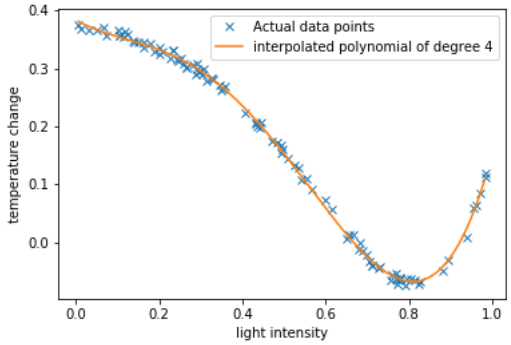
(b) degree 1



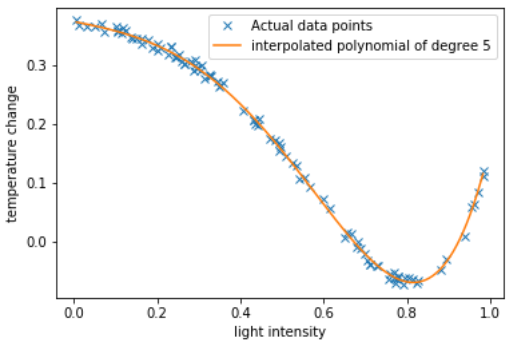
(c) degree 2



(d) degree 3



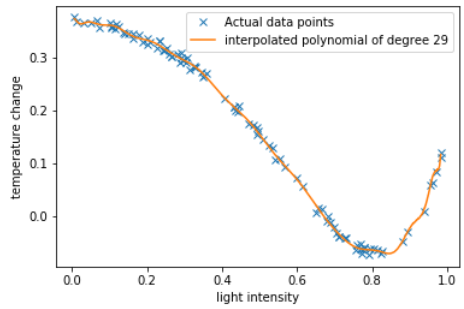
(e) degree 4



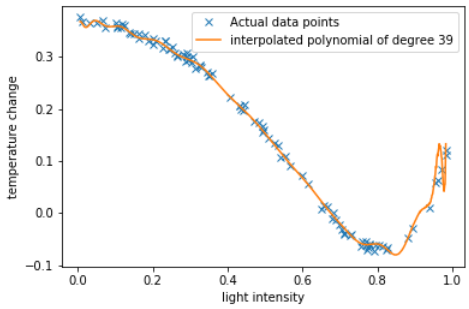
(f) degree 5

Figure 3: small degree interpolating polynomials

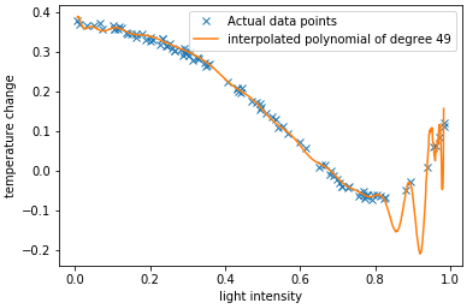
Looking back at Figure 1, we see that the residues increase at around degree 40. We can plot some more graphs that show the interpolating polynomials for larger degrees.



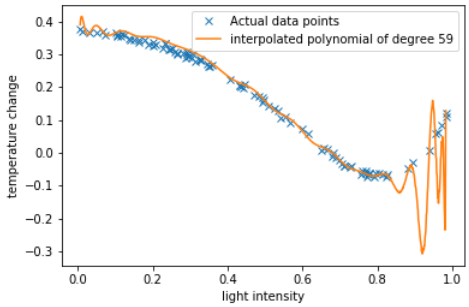
(a) degree 29



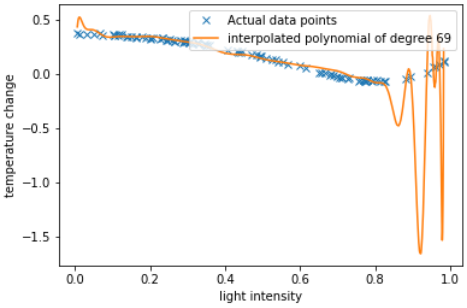
(b) degree 39



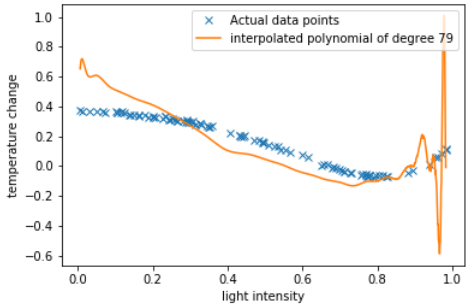
(c) degree 49



(d) degree 59



(e) degree 69



(f) degree 79

Figure 4: large degree interpolating polynomials

At degree 39, you can see the erratic behaviour that is causing the residues to increase in figure 1. One would expect that increasing the degree should decrease the residue function, since the interpolating polynomial is more able to pass through all the data points. However, this is clearly not the case here. Possible reasons for this behaviour is that the data is noisy and not smooth. In addition to this, the function 'systemsolve2' uses back substitution to find the coefficients. So, as the degree increases, there are more floating point rounding errors.

The most appropriate choice of polynomial is of degree four. This degree has a much smaller residue than the lower degrees and the residue is not significantly larger than the higher degree residues. This degree 4 polynomial is also much less likely to be overfitting the data like the higher degrees. From the previous graphs (figure 3e) we can also see that the degree 4 polynomial is a good fit visually.

3 QR analysis of dataset

Running the script 'CLA_P1_Q3.py' will output the R matrix from the QR decomposition of the 'readings2.csv' file. This R matrix has 0 entries for all rows except the first two. This can be checked using np.allclose again (line 16). R represents the change of basis from Q to A. The fact that R only has two non zero rows means that only two columns of Q are needed to generate A. So A only has two linearly independent vectors. Thus, we can conclude that the dataset can be interpolated with a degree 2 polynomial.

References