AM213A Homework 3 Part 1 (Numerical Problems)

Owen Morehead

February 24, 2022

1 General Guidelines

Under the `part1` directory inside `hw3/code`, there are `prob1_Cholesky` and `prob2_QR` directories that contain all the code and files for these numerical problems. Both subdirectories share the following same files:

- `LinAl.f90` program which contains all the subroutines.
- `utility.f90` which holds the variable, $dp = kind(0.e0)$.
- the `least_squares_dat.dat` file that contains $i = 21$ rows and 2 columns describing the data to be fitted as (x_i, y_i).

Within both `prob1_Cholesky` and `prob2_QR` there are unique Makefiles and unique Driver programs, titled `Driver_Cholesky.f90` and `Driver_QR.f90` respectively.

For both problems, the makefile is setup to compile the program such that the command `make LinAl` will make the executable that can then be run with the command, `./ LinAl`. This will print all the desired matrices and information as asked in the assignment, for both the 3rd degree and 5th degree polynomials in that order. In addition, for both problems, running the code will produce two `.dat` files containing column vector data: one that contains the solution polynomial coefficients for the 3rd degree polynomial fit, and one that contains the coefficients for the 5th degree polynomial fit.

A note is that all code executions are done using single precision floating point arithmetic. If one wants to change this do double precision, the step to take is in the `utility.f90` file, which is to simply change $dp = kind(0.e0)$ to $dp = kind(0.d0)$. Enforcing double precision floating point arithmetic can also be done in the makefile by adding the two flags, `fdefault-real-8` and `fdefault-double-8`.

There is also an additional matlab file, in both subdirectories `prob1_Cholesky` and `prob2_QR`, that is used to extract the data files and make the plot of the 3rd and 5th degree polynomials alongside the provided data.

2 Cholesky Solution of the Least-Squares Problem

2.1 Algorithm Outline

Here we look at numerically solving overdetermined systems, those with a larger number of equations (or data points, m) compared to the number of unknown parameters, n. There are generally no exact

solutions to overdetermined problems, rather we now find approximate solutions which minimize the residual error using some norm (Euclidean norm shown below),

$$E^2 = ||\mathbf{r}||_2^2 = ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2^2 = \sum_{i=1}^{m} (y_i - f(x_i, \mathbf{a}))^2.$$

Here we consider only **linear data fitting** problems in which the fitting function, $f(x_i, \mathbf{a})$, is linear in $\mathbf{a} = [a_1, \cdots, a_n]^T$, although f can be nonlinear in x.
One example of a linear data fitting problem explored here is a polynomial fitting in which we try to fit a polynomial

$$f(x_i, \mathbf{a}) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \tag{1}$$

where $n-1$ is the degree of the polynomial, to a set of m points (x_i, y_i). In matrix form, finding the best fit involves solving the overconstrained problem:

$$\mathbf{A}\mathbf{x} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & & & & \\ 1 & x_m & x_m^2 & \cdots & x_m^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \cong \begin{pmatrix} y_1 \\ y_1 \\ \vdots \\ y_m \end{pmatrix} = \mathbf{b} \tag{2}$$

where the matrix, A, formed in this process is called a **Vandermonde matrix**.
If there are many data points in this fitting problem, it is numerically advantageous to consider a slight modification. This uses the fact that minimizing the norm or its square is the same thing. Therefore we can write the square of the Euclidean norm of $\mathbf{r}$ as an inner product:

$$E^2 = ||\mathbf{r}||^2 = \mathbf{r}^T\mathbf{r} = (\mathbf{b} - \mathbf{A}\mathbf{x})^T(\mathbf{b} - \mathbf{A}\mathbf{x}).$$

The next steps are to expand E^2, and then minimize the residuals, i.e., find the solution x that satisfies $\frac{\partial E^2}{\partial x_i} = 0$ for $i = 1, \ldots, n$, which is straightforward to show that this requires,

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}.$$

These equations are referred to as the **normal equations** associated with the overdetermined problem $\mathbf{A}\mathbf{x} = \mathbf{b}$.

We see that since $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{A}^T\mathbf{A}$ is a square $n \times n$ matrix which is usually much smaller than the original matrix $\mathbf{A}$. Similarly, $\mathbf{A}^T\mathbf{b}$ is now a $n-$ long vector which is usually much smaller than the original $m-$long vector $\mathbf{b}$. Also it is true that as long as $\mathbf{A}$ is full rank (i.e. rank n), then $\mathbf{A}^T\mathbf{A}$ is positive definite, allowing us to use the **Cholesky decomposition** method to solve this overdetermined problem. Otherwise said, Cholesky decomposition is only applicable to positive definite Hermitian matrices such that the property $\mathbf{x}^*\mathbf{A}\mathbf{x} > 0$ is satisfied for any nonzero vector $\mathbf{x}$.

In summary, the Hermitian matrix A (we consider the SPD matrix associated with the normal equation $\mathbf{A}^T\mathbf{A}$) has an LU decomposition such that $\mathbf{A} = \mathbf{L}\mathbf{U}$. We also see that this implies that,

$$\mathbf{A}^* = (\mathbf{L}\mathbf{U})^* = \mathbf{U}^*\mathbf{L}^* = \mathbf{A} = \mathbf{L}\mathbf{U}$$

which shows that the LU decomposition of a Hermitian matrix satisfies

$$\mathbf{U}^*\mathbf{L}^* = \mathbf{L}\mathbf{U}.$$

Since the transpose of an upper-triangular matrix is a lower triangular one, and vice versa, this shows that in principle we can find a decomposition such that $\mathbf{L} = \mathbf{U}^*$, or, $\mathbf{U} = \mathbf{L}^*$, such that for any Hermitian matrix $\mathbf{A}$,

$$\mathbf{A} = \mathbf{LU} = \mathbf{U}^*\mathbf{U} = \mathbf{LL}^*.$$

It is also important to note that this factorization only exists for positive definite matrices. This can be seen from the following:

$$\mathbf{x}^*\mathbf{Ax} = \mathbf{x}^*\mathbf{U}^*\mathbf{Ux} = (\mathbf{Ux})^*(\mathbf{Ux}) > 0$$

where the existence of a decomposition $\mathbf{A} = \mathbf{U}^*\mathbf{U}$ (or $\mathbf{A} = \mathbf{LL}^*$) depends on the positive definiteness of $\mathbf{A}$.

So we have the definition that a Cholesky factorization of a Hermitian positive definite matrix $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{U}^*\mathbf{U} = \mathbf{LL}^*$ where $\mathbf{U}$ is upper-triangular and $\mathbf{L}$ is lower-triangular such that $u_{jj} = l_{jj} > 0$.

If we write out the decomposition $\mathbf{A} = \mathbf{LL}^*$ in component form, we have

$$a_{ij} = \sum_{k=1}^{m} (\mathbf{L})_{ik}(\mathbf{L})^*_{kj} = \sum_{k=1}^{m} l_{ik}l^*_{jk} = \sum_{k=1}^{\min(i,j)} l_{ik}l^*_{jk} \tag{3}$$

where we remember that since $\mathbf{L}$ is lower triangular, its coefficients are zero if the column number is larger than the row number. Since the matrix is Hermitian, we have that $a_{ii} = a^*_{ii}$, and so a_{ii} must be real, implying that, for example $i = 1$, we have $a_{11} = l_{11}l^*_{11} = ||l_{11}||^2$.

There are many similar versions of the Cholesky factorization algorithm which simply reorder some of the operations in different ways. Most versions are roughly equivalent in terms of time and stability. The algorithm presented here is straightforward and corresponds to working column by column, first calculating the diagonal element in each column, that is, calculating

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}l^*_{jk}} \tag{4}$$

and then working on the column below that element, in which

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}j^*_{jk}}{l_{jj}}. \tag{5}$$

The elements of $\mathbf{L}$ are stored by over-writing the lower triangular elements of $\mathbf{A}$.
An important note is that the asterisks on the matrix elements such as l^*_{jk} in Equation 4 represent the complex conjugate and not conjugate transpose since these are individual matrix entries and thus scalars. Since the matrices we are dealing with are purely real, we can simply write that $l^*_{jk} = l_{jk}$.

This algorithm is popular for symmetric positive definite (SPD) matrices because, for example, there is no pivoting required for numerical stability, only about $n^3/6$ multiplications and additions are required, making it around twice as fast as standard Gaussian elimination.
Also, if $\mathbf{A}$ is Hermitian positive definite, then the diagonal elements are all positive so the square root is well defined. If one of the diagonal elements is not positive, this means the original matrix was not positive definite. Therefore this algorithm is in addition, a test to deterimine the positive-definiteness

of a matrix.

To determine solutions to the problem $\mathbf{Ax} = \mathbf{b}$, we now need to solve the problem $\mathbf{LL}^*\mathbf{x} = \mathbf{b}$, which can be decomposed into the two steps that define this backsubstition algorithm:

- a forward substitution step of the form $\mathbf{Ly} = \mathbf{b}$
- a backsubstitution step of the form $\mathbf{L}^*\mathbf{x} = \mathbf{y}$, which gives the solution vector $\mathbf{x}$.

2.2 Numerical Results

Continuing with the numerical results of this algorithm, we first fit the data with a 3rd-degree polynomial using single-precision floating point arithmetics. Reported here is the polynomials coefficients (corresponding to those in Equation 1), the 2-norm error on the fit, and a figure comparing the fitted curve to the data.
The data is also fit with a 5th-degree polynomial, and the same quantities are reported as for the 3rd degree polynomial.

2.2.1 3rd Degree Polynomial Fit

Solution vector. The obtained polynomial coefficients, $\mathbf{a} = [a_0, a_1, a_2, a_3]^T$ corresponding to the polynomial defined by Equation 1 (note I switch to use of $\mathbf{x}$ to follow the algorithm notation:

$$\mathbf{x} = \begin{bmatrix} 1.83189678 \\ -5.17035389 \\ 11.2041264 \\ -7.28502893 \end{bmatrix}$$

To verify the solution is correct we can calculate its 2-norm error, defined as:

$$||\mathbf{A}^T\mathbf{Ax} - \mathbf{A}^T\mathbf{b}||_2 = 3.471 \times 10^{-6}.$$

This close-to-zero value of the 2-norm error tells us that the solution vector $\mathbf{x}$ is indeed the correct solution to the normal equation, $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$.

We can also calculate the 2-norm error between the fitted curve and the data. The 2-norm of the error on the fit refers to the quantity,

$$||\mathbf{r}||_2 = ||\mathbf{b} - \mathbf{Ax}||_2 = \sqrt{\sum_{i=1}^{m}(y_i - (a_0 + a_1x_i + a_2x_i^2 + a_3x_i^3 + \ldots))^2}$$

which is a minimization of the residual vector $\mathbf{r}$ in the broader context of overdetermined systems. Again, sorry for the double-labeling of the coefficients $\mathbf{a}$ as also the solution vector $\mathbf{x}$. Nevertheless, we can plug in the solution coefficients as well as the data points from the data file, and find that,

$$||\mathbf{r}||_2 = 0.244575.$$

2.2.2 5th Degree Polynomial Fit

Solution vector. The obtained polynomial coefficients, $\mathbf{a} = [a_0, a_1, a_2, a_3, a_4, a_5]^T$:

$$\mathbf{x} = \begin{bmatrix} 1.86812568 \\ -7.22018242 \\ 28.5390663 \\ -58.0934753 \\ 60.3702278 \\ -24.9616680 \end{bmatrix}$$

We can verify the solution is correct by calculating its 2-norm error, defined as:

$$||\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}||_2 = 5.779 \times 10^{-5}.$$

Similar to the 3rd degree polynomial, this close-to-zero value of the 2-norm error tells us that the solution vector $\mathbf{x}$ is indeed the correct solution to the normal equation, $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$. We expect this value to be slightly larger than that for the 3rd degree polynomial since we are dealing with larger matrices in our numerical calculations, leaving more room for potential errors arising from floating point arithmetic.

We also find, for this 5th degree polynomial case, that the 2-norm error on the fit is,

$$||\mathbf{r}||_2 = 0.172762.$$

This 2-norm error on the fit is reasonably small and tells us that both the 3rd and 5th degree polynomials fit the data fairly well, with the 5th degree polynomial returning slightly smaller residuals (better fit).

A plot of the 3rd and 5th degree fitted curves against the data is shown below.

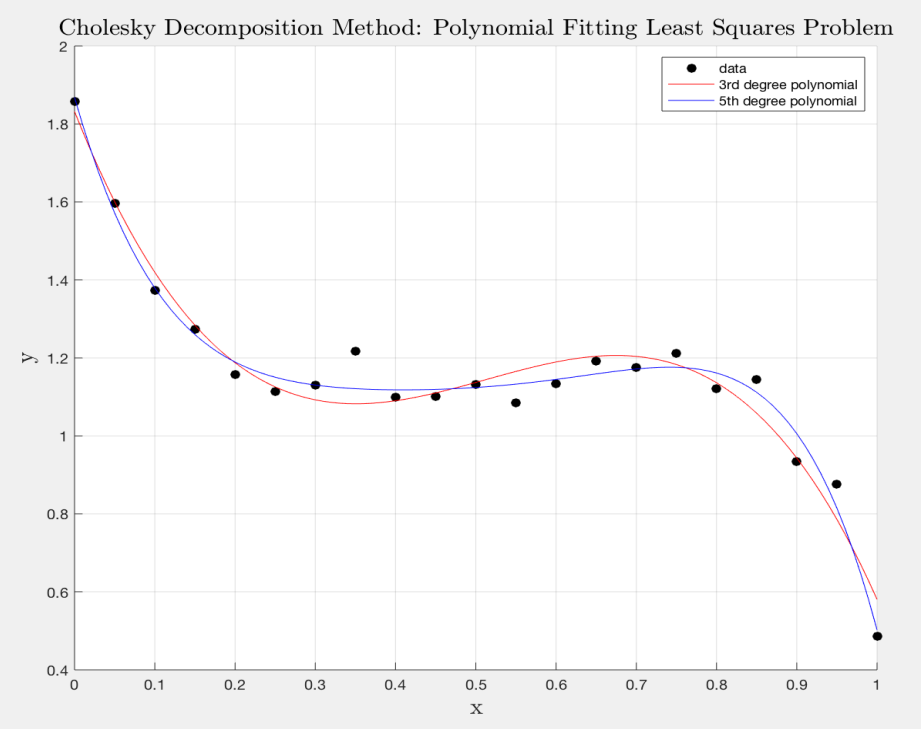

Figure 1: A plot comparing the 3rd degree polynomial and 5th degree polynomial fitted curves against the data, for the Cholesky decomposition method of solving the least squares problem.

2.3 Discussion

The maximum degree of polynomials for the given data should be $m-1$ where m is the amount of data points we are given. In this case $m = 21$, so the max degree of polynomials for this data should be 20. The important point here is that since the dimension of A as the Vandermonde matrix is $(m \times n)$, where $n-1$ is the degree of our polynomial fit, we have that A^TA is full rank and has dimension $n \times n$. This is an invertible matrix that can be used in Cholesky decomposition. But in the case that our polynomial is of degree n, the dimension of A^TA is $(n+1) \times (n+1)$ and the rank remains n, so in this case (and also the case that the degree of our polynomial $> n$), A^TA is not invertible. In other words, linear dependence arises as the result of fitting more (or an equal amount of) variables than data points.

Essentially what happens when we try to fit the data with a higher-degree polynomial is we begin to overfit the data. A problem that arises is also that polynomials such as $x^{n-1} = x^{20}$ grow very rapidly. However, since all of our data points are between 0 and 1, this is not an issue. The issue is instead that the values of x^{20} and other large degree polynomial terms will be very small (since $0 \geq x_i \leq 1$). For example if $x = 0.05 \rightarrow (0.05)^{20} = 9.5 \times 10^{-27}$. This means that the coefficients on these terms will have to be very large in order to best fit a large number of noisy measurements. So either way, whether due to huge values of the x^{n-1} terms or huge values of the coefficients on those terms, the fit will become less smooth. The fit polynomial might go through the data points better, but inbetween the data points the function fluctuates widely and is an unreasonable fit (see Figure 2). A high degree polynomial will generally do very poorly at predicting values that are out-of-sample. And in general, the goal of numerical fitting routines is to find the best fit that is

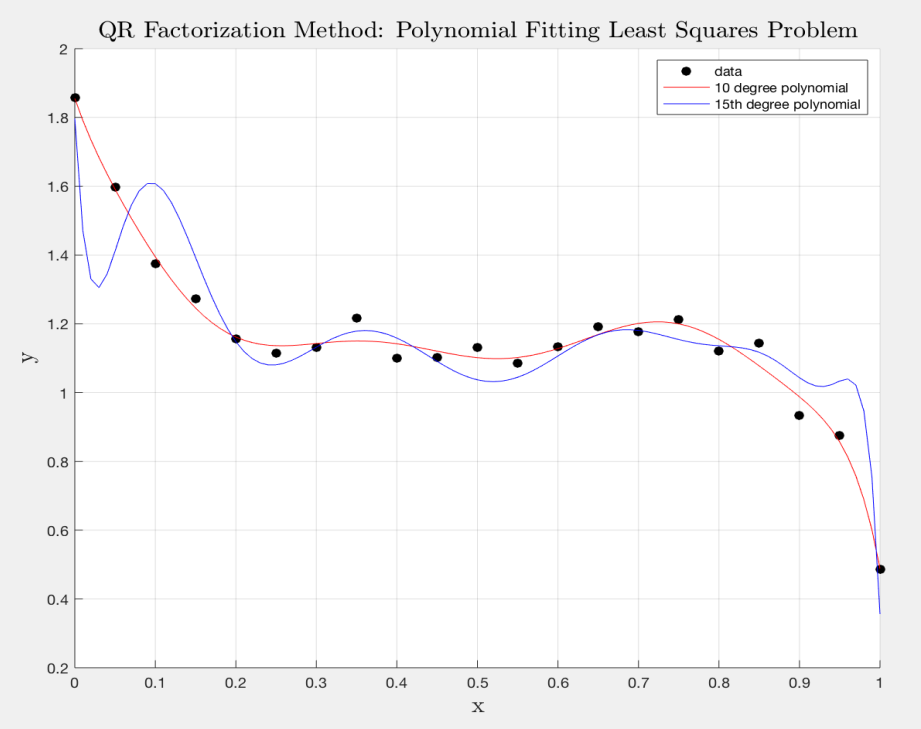

Figure 2: A plot comparing the 10th degree polynomial and 15th degree polynomial fitted curves against the data, for the QR factorization method of solving the least squares problem.

also the most simple or described by the least amount of terms. I.e., it is not advantageous to fit the data with a polynomial of say 15 or more terms when there is a fit described by a much smaller degree polynomial that can fit the data equally as well or better.

In Figure 2 we use the QR factorization method to compute these high degree polynomial fits (10th and 15th degree polynomials) because the Cholesky decomposition method fails in single precision at calculating polynomial coefficients at these high degrees (discussed below). But despite which method is used, we still see that the higher degree the polynomial fit is, the more variation we get in the smoothness of the fit. In general, a large degree polynomial fit as such will not fit the data any better than a mid or smaller degree polynomial fit.

As previously stated, Cholesky's method serves as a test of positive definiteness. Therefore, in single precision floating point arithmetic, the algorithm fails if $\mathbf{A}$ is not a Hermitian positive definite matrix. As we see in Equation 4, this algorithm requires computing the square root of some numbers located on the diagonal of the matrix we are working on. If the matrix is not positive definite, it can be proved that one of these diagonal elements will be negative, and thus the algorithm will fail when trying to take the square root of the negative real number.

In addition, if we continue to fit polynomials with larger degrees, there will be a point at which this algorithm fails even though the matrix is SPD. For single precision floating point arithmetic, this happens for a 6th degree polynomial and above. For double precision we expect the algorithm to be able to calculate higher degree polynomials, and indeed we see it starts to fail when trying to compute an 11 degree polynomial fit. We also note that the 2-norm error of the solution vector is 3.394×10^{-10}, which can be compared to say this quantity for a 5th degree polynomial fit, 5.357×10^{-14}. We see for a smaller degree polynomial the norm of the solution error is on the order of 10^4 times closer to machine accuracy, which is quite a difference.

This has to do with the fact that Cholesky decomposition is backward stable and hence can accurately solve problems with condition numbers that are not extremely large. Although this method is known to have the least operation count, it is also the least accurate of all methods because of the fact that it utilizes the normal equations and the unfortunate consequence that is,

$$\text{cond}(\mathbf{A}^T\mathbf{A}) = \text{cond}(\mathbf{A})^2.$$

So if $\mathbf{A}$ is relatively poorly conditioned (large condition number), then $\hat{\mathbf{A}} = \mathbf{A}^T\mathbf{A}$ is very poorly conditioned. A numerical consequence is that this quantity can be singular even if the original problem is not.

Poor conditioning is unfortunately common in data sets that involve points with largely varying values of x, especially for polynomial fitting and their associated Vandermonde matrices. As initially explained in the beginning of this Discussion seciton, the data set we are fitting has a domain $0 \geq x_i \leq 1$, thus larger degree polynomialy will end up with coefficients which vary vastly, some being small and some being very large values. For example, in double precision the largest degree polynomial that the Cholesky algorithm succeeds with is 11 and the solution coefficients vary from 1.85 to 516, 589. In double precision it seems our Cholesky algorithm cannot handle solution values that range any larger than that! It is for this reason that method such as QR decomposition were developed that do not involve forming the normal equations (as done for Cholesky decomposition). Instead, we get to work directly with the original matrix $\mathbf{A}$.

Nevertheless, for problems with low to moderate condition numbers, the normal equation method with Cholesky decomposition can be preferable over other methods. However is the condition number is not moderately small, we will want to move towards a better algorithm for Least Squares problems.

3 QR Solution of the Least-Squares Problem

3.1 Algorithm Outline

In general, QR decomposition (factorization) is a decomposition of a matrix $\mathbf{A}$ into a product of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$ such that $\mathbf{A} = \mathbf{QR}$.
There are two main methods of QR factorization:

- A reduced QR factorization of a full rank $m \times n$ $(m > n)$ matrix $\mathbf{A}$ expressed as $\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$ where $\hat{\mathbf{Q}}$ is an $m \times n$ matrix whos column vectors are orthonormal, and $\hat{\mathbf{R}}$ is an $n \times n$ upper triangular matrix.

- A full QR factorization of a full rank $m \times n$ $(m > n)$ matrix $\mathbf{A}$ expressed as $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is an $m \times m$ orthogonal matrix (unitary if $\mathbf{A}$ is complex) and $\mathbf{R}$ is an $m \times n$ upper triangular matrix (i.e. a matrix whose entries r_{ij} are 0 if $i > j$).

We see that these two definitions are related in the sense that $\hat{\mathbf{Q}}$ forms the first n column vectors of $\mathbf{Q}$, and $\hat{\mathbf{R}}$ forms the first n rows of $\mathbf{R}$. The last $m - n$ columns of $\mathbf{Q}$ are filled with vectors that are orthogonal to eachother and to the span of $\hat{\mathbf{Q}}$, while the last $m - n$ rows of $\mathbf{R}$ are just filled with zeros.

A QR factorization based on Householder transformations is sometimes known as the orthogonal triangularization of$\mathbf{A}$ and is a full QR factorization, by contrast with the Gram-Schmidt QR factorizaiton algorithm which performs a triangular orthogonalization and is a reduced QR method. In the Householder method, $\mathbf{A}$ is gradually transformed into an upper triangular matrix $\mathbf{R}$ by successive application of orthogonal transformations of the kind

$$\mathbf{Q}_n\mathbf{Q}_{n-1}\dots\mathbf{Q}_1\mathbf{A} = \mathbf{R}.$$

The product of orthogonal matrices is also orthogonal, so if we call it $\mathbf{Q}^T$, we then have,

$$\mathbf{A} = \mathbf{QR} \text{ where } \mathbf{Q}^T = \mathbf{Q}_n\mathbf{Q}_{n-1}\dots\mathbf{Q}_1$$

These orthogonal matrices used in the transformation process are known as *Householder matrices* and geometrically describe a reflection about a plane or hyperplane containing the origin. The Householder matrix is defined by,

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} \tag{6}$$

which tells us that $\mathbf{H}$ is both symmetric ($\mathbf{H}^T = \mathbf{H}$), and orthogonal ($\mathbf{H}^T\mathbf{H} = I \rightarrow \mathbf{H}^T = \mathbf{H}^{-1}$). The vector $\mathbf{v}$ is a column unit vector satisfying $||\mathbf{v}|| = 1$. Geometrically this Householder matrix is a reflection across the plane perpendicular to $\mathbf{v}$. Defining properties of this orthogonal reflection about a plane are that any vector parallel to the plane is exactly reflected, while vectors perpendicular to the plane remain invariant.

The general idea is that these Householder matrices are used in QR decomposition to gradually zero out the sub-diagonal elements of $\mathbf{A}$. As a reminder, A is the Vandermonde style matrix used for the application of best fitting a certain degree polynomial to an array of data.

Let us be reminded that orthogonal operations on vectors are norm-preserving. By the fact that $\mathbf{Q}^T\mathbf{Q}$, we have that $||\mathbf{Qx}|| = \sqrt{(\mathbf{Qx})^T\mathbf{Qx}} = \sqrt{\mathbf{x}^T\mathbf{x}} = ||\mathbf{x}||$.
In terms of zeroing out all the entries below the diagonal in A, we first create an orthogonal transformation that takes the first column vector $\mathbf{a}_1$ and returns a vector with the same norm but pointing in the $\mathbf{e}_1$ direction, $\mathbf{Q}_1\mathbf{a}_1 = \pm||\mathbf{a}_1||\mathbf{e}_1$. Geometrically, constructing $\mathbf{Q}_1$ thus means to construct the relevant plane that reflects $\mathbf{a}_1$ onto $\pm||\mathbf{a}_1||\mathbf{e}_1$.

Both positive and negative choices of reflection equally work in principle, but the first can be shown to be more prone to numerical errors due to finite-precision arithmetic. Therefore, we prefer to choose the sign that is equal to **minus** the sign of $\mathbf{a}_1^T\mathbf{e}_1$ (i.e. $-\text{sign}(a_{11})$).
Geometrically, this means that we need a Householder transformation $\mathbf{H}_1$, such that

$$\mathbf{H}_1\mathbf{a}_1 = -s_1\mathbf{e}_1 \tag{7}$$

where $s_1 = \text{sign}(a_{11})||\mathbf{a}_1||$ is the signed norm of $\mathbf{a}_1$. The correct normalized vector $\mathbf{v}_1$ that can perform this transformation is found by solving

$$(\mathbf{I} - 2\mathbf{v}_1\mathbf{v}_1^T)\mathbf{a}_1 = -s_1\mathbf{e}_1. \tag{8}$$

This implies that

$$\mathbf{v}_1 = \frac{\mathbf{a}_1 + s_1\mathbf{e}_1}{2\mathbf{v}_1^T\mathbf{a}_1}. \tag{9}$$

Since the term in the denominator is a constant, and since $\mathbf{v}_1$ is normalized, we then have:

$$\mathbf{v}_1 = \frac{\mathbf{a}_1 + s_1\mathbf{e}_1}{||\mathbf{a}_1 + s_1\mathbf{e}_1||}. \tag{10}$$

Thus, once $\mathbf{a}_1$ is known, $\mathbf{v}_1$ can be easily constructed. Once we construct this vector, we then apply the corresponding Householder reflection $\mathbf{H}_1$ to $\mathbf{A}$, which, as a note, doesn't meaningfully affect the other columns of $\mathbf{A}$.
Looking at the next column, we effectively want to transform $\mathbf{a}_2$ into a vector that lies in the plane formed by $\mathbf{e}_1$ and $\mathbf{e}_2$, without moving the transformed vector $\mathbf{a}_1$ away from $\mathbf{e}_1$. We know that for

$\mathbf{e}_1$ to be invariant by transformation $\mathbf{H}_2$, $\mathbf{e}_1$ must lie in the plane of reflection (i.e. be perpendicular to the vector $\mathbf{v}_2$). This means $\mathbf{v}_2$ must have a null first entry.
All in all, for this second transformation, we will have a Householder matrix $\mathbf{H}_2$ that zeroes out the subdiagonal elements of the second column of $\mathbf{A}$ without modifying the first column. This is defined by, $\mathbf{H}_2 = \mathbf{I} - 2\mathbf{v}_2\mathbf{v}_2^T$, where

$$s_2 = \text{sign}(a_{22})(\sum_{k=2}^{m} a_{k2}^2)^{1/2}. \tag{11}$$

In general, we then have that at the j-th step,

$$\mathbf{H}_j = \mathbf{I} - 2\mathbf{v}_j\mathbf{v}_j^T \tag{12}$$

where the vector $\mathbf{v}_j$ is defined as:

$$\mathbf{v}_j = (0, \ldots, 0, a_{jj} + s_j, a_{j+1,j}, \ldots, a_{mj})^T / ||\mathbf{v}_j|| \tag{13}$$

and

$$s_j = \text{sign}(a_{jj})(\sum_{k=j}^{m} a_{kj}^2)^{1/2}. \tag{14}$$

Let us now explicitly calculate the product of $\mathbf{H}_j$ with $\mathbf{A}$ at step j to prove that it indeed casts $\mathbf{A}$ in the right form and forms the new diagonal element of $\mathbf{A}$ as $-s_j$. Consider the matrix,

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}$$

The first Householder step is to construct the Householder vector,$\mathbf{v}_1$, which will zero out the subdiagonal entries of the first column $\mathbf{a}_1$ of $\mathbf{A}$. In this simple case we have that, $s_1 = ||\mathbf{a}_1||_2 = \sqrt{3}$. Thus we have that,

$$\mathbf{v}_1 = \mathbf{a}_1 + s_1\mathbf{e}_1$$

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ -1 \\ 0 \end{bmatrix} + \begin{bmatrix} \sqrt{3} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1+\sqrt{3} \\ 0 \\ 0 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

Applying the resulting $\mathbf{H}_1$ to the first column returns,

$$\mathbf{H}_1\mathbf{a}_1 = \mathbf{a}_1 - 2\mathbf{v}_1\frac{\mathbf{v}_1^T\mathbf{a}_1}{\mathbf{v}_1^T\mathbf{v}_1} = \begin{bmatrix} -\sqrt{3} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We see that indeed the diagonal element, $A_{11} = -s_1$ and the elements below this first column are now zeroed out.

Continuing the general algorithm, after n applications of the algorithm, we have

$$\mathbf{H}_n \dots \mathbf{H}_1 \mathbf{A} = \mathbf{R} \tag{15}$$

where each Householder matrix $\mathbf{H}_j$ is orthogonal by construction, and as defined above $\mathbf{Q} = \mathbf{H}_1 \dots \mathbf{H}_n$ or similarly, $\mathbf{Q}^T = \mathbf{H}_n \dots \mathbf{H}_1$.
This is the entirety of the QR householder factorization algorithm. It gradually transforms $\mathbf{A}$ into $\mathbf{R}$ through successive application of $\mathbf{H}_j$ to $\mathbf{A}$, defined by $\mathbf{A} = \mathbf{A} - 2\mathbf{v}_j\mathbf{v}_j^T\mathbf{A}$. At the same time we can also be computing and storing $\mathbf{Q}$, providing us with the two matrices we need to then obtain the solution vector.

To obtain the solution vector, we can simply perform a back-substitution method to solve the upper triangular system $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$, where the transformed right hand side $\mathbf{Q}^T\mathbf{b} = \binom{\mathbf{c}_1}{\mathbf{c}_2}$ with n-vector $\mathbf{c}_1$ and $(m-n)$-vector $\mathbf{c}_2$. The result is that $\mathbf{Rx} = \mathbf{c}_1$, which is a linear system of equations transformed into row echelon form. This is now solvable by a standard backsubstitution routine. The process of such a routine utilizes the fact that the bottom most row has the most zeros in it. In essence, backsubstitution entails solving the last equation first, and then working our way up row by row until we have our full solution vector. As noted in more detail below, for this Householder QR factorization method, only the first n rows are needed to obtain the solution vector. Therefore this leaves the minimum residual as defined by $||\mathbf{r}||_2^2 = ||\mathbf{c}_2||_2^2$.

Let us remember that although the equality $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$ is well-defined in the sense that it equates two m-long vectors, there are only $n < m$ unknowns. However, this mismatch does not matter since $\mathbf{R}$ simply contains $\hat{\mathbf{R}}$ in its upper triangular elements (first n rows), and zeroes below. Also, the matrix $\mathbf{Q}$ contains $\hat{\mathbf{Q}}$ in its first columns and other orthogonal vectors after that. This tells us that we can recover the original reduced QR factorization of a full rank matrix, $\hat{\mathbf{R}}\mathbf{x} = \hat{\mathbf{Q}}^T\mathbf{b}$, by looking only at the first n lines of the problem, $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$, and ignoring the other lines, thus enhancing the numerical efficiency.

As a concluding note, the QR method using Householder transformation is preferred over Gram-Schmidt orthogonalization when used in the context of Least-Square problem because it guarantees the orthogonality of $\mathbf{Q}$ to within machine accuracy. However, the QR method lacks in its parallelizability since each column j must be zeroed out one at a time. This is not true of modified Gram-Schmidt algorithm where the entire matrix $\mathbf{A}$ is orthogonanlized at the same time by the iterative process. Thus, many parallel QR algorithms work with this Gram-Schmidt method as the loss of accuracy is an acceptable price to pay for the enhanced computation time.

3.2 Numerical Results

Continuing with the numerical results of this algorithm, we first fit the data with a 3rd-degree polynomial using single-precision floating point arithmetics. Reported here is the polynomials coefficients (corresponding to those in Equation 1), the 2-norm error on the fit, and a figure comparing the fitted curve to the data.
The data is also fit with a 5th-degree polynomial, and the same quantities are reported as for the 3rd degree polynomial.

3.2.1 3rd Degree Polynomial Fit

Solution vector. The obtained polynomial coefficients, $\mathbf{a} = [a_0, a_1, a_2, a_3]^T$ corresponding to the polynomial defined by Equation 1 (note I switch to use of $\mathbf{x}$ to follow the algorithm notation:

$$\mathbf{x} = \begin{bmatrix} 1.83190632 \\ -5.17046642 \\ 11.2043791 \\ -7.28518391 \end{bmatrix}$$

To verify the solution is correct we can calculate its 2-norm error, defined as:

$$||\mathbf{Rx} - \mathbf{Q}^T\mathbf{b}||_2 = 4.008 \times 10^{-6}.$$

This close-to-zero value of the 2-norm error tells us that the solution vector $\mathbf{x}$ is indeed the correct solution to the upper triangular equation where $\mathbf{A}$ has been transformed into the upper triangular matrix $\mathbf{R}$, $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$.

We can also calculate the 2-norm error between the fitted curve and the data. Again, sorry for the double-labeling of the coefficients $\mathbf{a}$ as also the solution vector $\mathbf{x}$. Nevertheless, we can plug in the solution coefficients as well as the data points from the data file, and find that,

$$||\mathbf{r}||_2 = 0.244575$$

which is the same value we found for the Cholesky factorization method.

3.2.2 5th Degree Polynomial Fit

Solution vector. The obtained polynomial coefficients, $\mathbf{a} = [a_0, a_1, a_2, a_3, a_4, a_5]^T$:

$$\mathbf{x} = \begin{bmatrix} 1.86953092 \\ -7.26421976 \\ 28.8174820 \\ -58.7613411 \\ 61.0526695 \\ -25.2121868 \end{bmatrix}$$

We can verify the solution is correct by calculating its 2-norm error, defined as:

$$||\mathbf{Rx} - \mathbf{Q}^T\mathbf{b}||_2 = 5.183 \times 10^{-6}.$$

Similar to the 3rd degree polynomial, this close-to-zero value of the 2-norm error tells us that the solution vector $\mathbf{x}$ is indeed the correct solution to the upper triangular equation where $\mathbf{A}$ has been transformed into the upper triangular matrix $\mathbf{R}$, $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$.
In comparing the 2-norm on the error of the solution vector to that found from the Cholesky decomposition method, we see this quantity is slightly smaller, i.e. there are less numerical errors arising from floating point arithmetic from this method.

We also find, for this 5th degree polynomial case, that the 2-norm error on the fit is close to the exact the same as that for Cholesky decomposition. These two values differ at the 5th decimal place. This tell us that the two methods are computing essentially equally as good fits for the data.

$$||\mathbf{r}||_2 = 0.172749.$$

A plot of the 3rd and 5th degree fitted curves against the data is shown below.

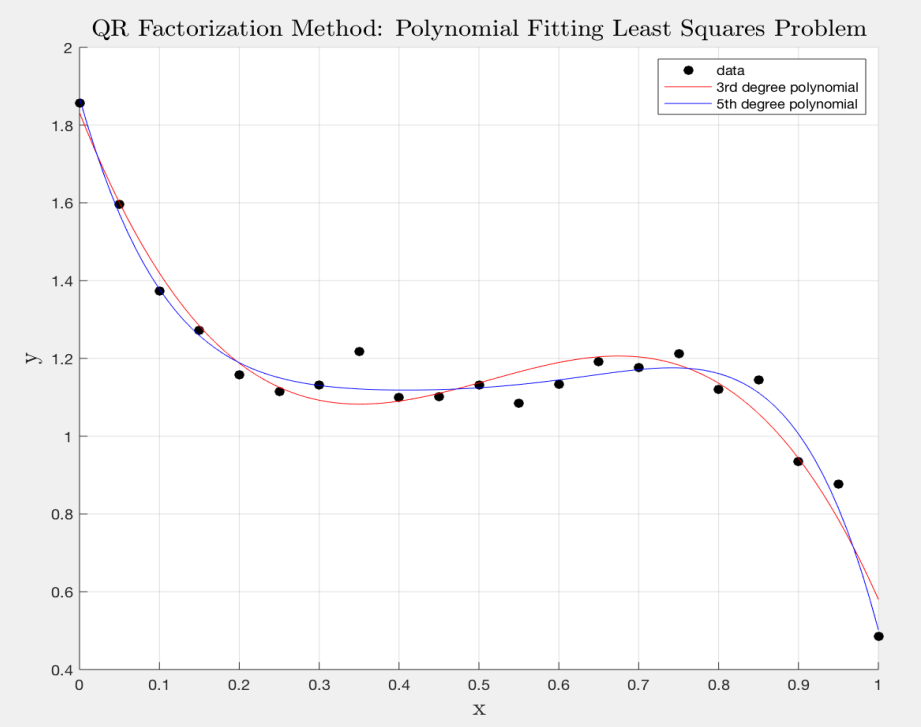

Figure 3: A plot comparing the 3rd degree polynomial and 5th degree polynomial fitted curves against the data, for the QR factorization method of solving the least squares problem.

3.3 Discussion

The point at which this algorithm fails for our data set (in single-precision floating point arithmetic) is never! Unless of course we try to fit a polynomial of an equal or higher degree than the amount of data points we have. In general, a QR decomposition can **always** be computed on any matrix, even if it does not have a full rank. Therefore the constructor will never fail. This method guarantees orthogonality of $\mathbf{Q}$ to within machine accuracy.

We can verify this by calculating higher degree polynomial fits using the QR factorization method. As shown in the figure in the Cholesky section, we were able to calculate 10th and 15th degree polynomial fits, although that is not to say they fit the data better than a lower degree polynomial. I was also able to calculate a 20th degree polynomial fit, where we find that the polynomial coefficients range from 1.9 to -4,914,195.5! This makes for a large condition number, but note we do not have to form the normal equations for this QR factorization rather we can work with the original Vandermonde matrix. So this method is able to calculate degree polynomial fits up to $m-1$ where again m is the total amount of data points we have. As a confirmation, we indeed get a runtime error when we try to calculate an $m = 21$ degree polynomial fit, as the index 22 is above the upper bound of 21 for the Vandermonde matrix.

Moving along, we want to analyze the the values obtained for the Frobenius-norms of the following quantities for the 5th order polynomial fit (in single precision):

$$||\mathbf{A} - \mathbf{QR}||_F = 8.3298 \times 10^{-6}$$

and

$$||\mathbf{Q}^T\mathbf{Q} - \mathbf{I}||_F = 2.3420 \times 10^{-6}.$$

From this we can conclude that indeed, this Householder QR factorization method accurately calculates the decomposition of $\mathbf{A}$, measured by $||\mathbf{A} = \mathbf{QR}||$, and also the orthogonality of the $\mathbf{Q}$ matrix, measured by $||\mathbf{Q}^T\mathbf{Q} - \mathbf{I}||$. One thing to note is that these quantities are slightly larger than machine accuracy in single precision ($\approx 1 \times 10^{-8}$ to 9×10^{-7} depending on the definition being used). In brief, machine accuracy measure of the best relative accuracy that we can hope to achieve. This QR factorization method entails a relatively large number of floating point arithmetic, and slowly over time the algorithm accumulates more roundoff error. Nevertheless these values obtained are satisfactory and show that Householder QR factorization is an accurate method of approximating a polynomial fit to the least squares problem.

In addition, in double precision we find that the Frobenius-norms of these same quantities are:

$$||\mathbf{A} - \mathbf{QR}||_F = 4.4859 \times 10^{-15}$$

and

$$||\mathbf{Q}^T\mathbf{Q} - \mathbf{I}||_F = 4.0493 \times 10^{-15}.$$

Again, these values are close to machine accuracy in double precision ($\approx 1 \times 10^{-16}$), confirming the Householder QR method guarantees the orthogonality of $\mathbf{Q}$ to within machine accuracy as well as the $\mathbf{QR}$ decomposition of $\mathbf{A}$ to within machine accuracy.