Question 1.1
Explain in reasonable detail how the Cholesky decomposition and backsubstitution work.

Solution: Cholesky factorization as presented in this course requires an input of a Hermitian positive definite matrix. This process transforms such a matrix A into $U^*U = LL^*$, where U is upper triangular and L lower, such that $u_{jj} = l_{jj} > 0$. That is, we have positives along the diagonal. We begin by choosing $l_{11} = \sqrt{a_{11}}$, and move to the next equation, giving

$$l_{21} = \frac{a_{21}}{l_{11}}$$

$$l_{22} = l_{22}^* = \sqrt{a_{22} - l_{21}l_{21}^*}$$

So the next l depends on the previous one, and we can sequentially compute these values with no glaring issues. With our matrix being symmetric positive definite at the start, we won't arrive at square roots of negative numbers so all should be well defined. In short, we work column by column computing the diagonal element in each column with the process shown above. Then we work on the column below that element, continuing the process until we're out of columns. Once all is said and done, the elements of L are stored by over-writing the lower triangular elements of A. A nice artifact of this method is that in computing the square roots of these numbers, we can actually check if a matrix is symmetric positive definite *if* the algorithm works all the way through. Meaning instead of checking if a matrix is SPD before passing it into this process, just pass it in and see if it works!

With our matrix Cholesky decomposed, it is time for back substitution. This process involves solving two sub problems in order to find the solution to $LL^*x = b$. The first, a forward substitution step solving

$$Ly = b$$

giving us the vector y. With this, we perform the back substitution step

$$L^*x = y$$

which gives the vector x, our desired solution.

Question 1.2
Fit the data with a 3rd-degree polynomial, using single-precision floating point arithmetics. Report what the polynomial coefficients you find are, what the 2-norm error on the fit is, and provide a figure comparing the fitted curve with the data.

Solution: Requiring a 3rd degree polynomial and using single precision floating point arithmetics, I find the coefficients to be (full decimal representation kept since output.txt has been filled with higher degree solutions)

$$1.8319077733858611, \quad -5.1704640498899117, \quad 11.204369949902796, \quad -7.2851782508501746$$

or the line of best fit is (approximately) given by the function

$$f(t) = 1.831 - 5.170t + 11.204t^2 - 7.285t^3$$

The 2-norm of $Ax - b$ is

$$7.7429599506130024E - 015$$

and the 2-norm of the error between the fitted curve and our given data is

$$0.24457513137092385$$

As you can tell by the above errors, this is a pretty good fit. Not bad for single precision! Alarmingly, I initially implemented the whole process in double precision. After changing to single precision, the errors really didn't change much. I guess single precision was good enough for this algorithm? Regardless, consider the figure showcasing both the fitted curve and given data.

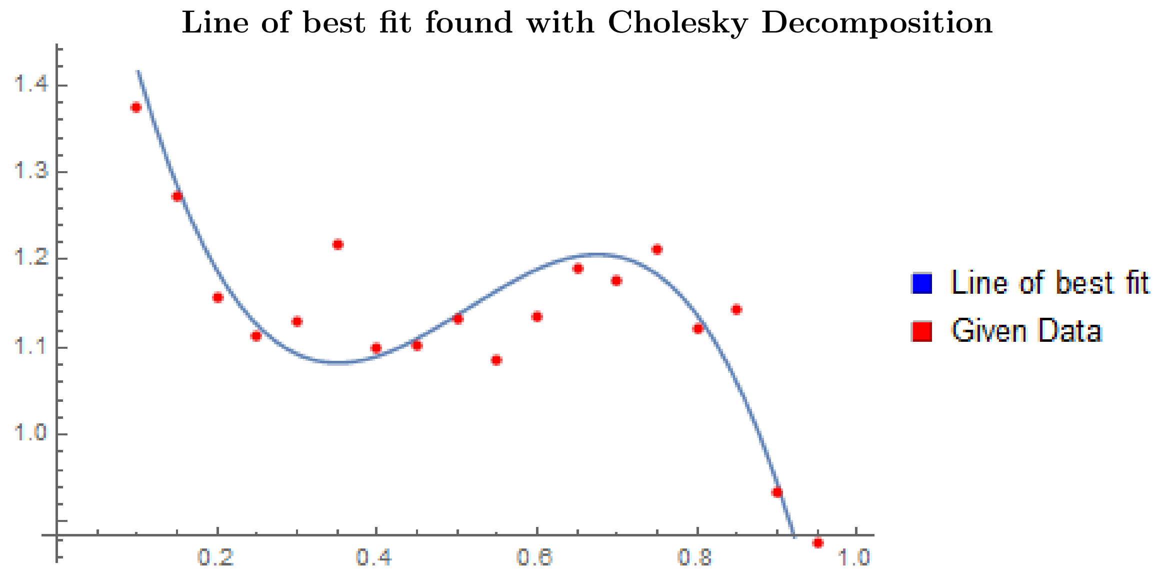

figure 1. Line of best fit vs. given data, single precision, 3rd degree polynomial

Question 1.3
Fit the data with a 5th-degree polynomial, using single-precision floating point arithmetics. Report what the polynomial coefficients you find are, what the 2-norm error on the fit is, and provide a figure comparing the fitted curve with the data.

Solution: Requiring a 5th degree polynomial and using single precision floating point arithmetics, I find the coefficients to be

$$1.869, \quad -7.264, \quad 28.817, \quad -58.761, \quad 61.053, \quad -25.212$$

or the line of best fit is (approximately) given by the function

$$f(t) = 1.869 - 7.264t + 28.817t^2 - 58.762t^3 + 61.053t^4 - 25.212t^5$$

The 2-norm of $Ax - b$ is

$$3.1025916901776484E - 014$$

and the 2-norm of the error between the fitted curve and our given data is

$$0.17274771750962778$$

This is very interesting, in comparison to our 3rd degree polynomial the 2-norm of $Ax - b$ is larger, meaning it solves the system with normalized equations worse. However, the 2-norm error between the fitted curve and our data is smaller, meaning we've found a better line of fit. This makes sense as a higher degree polynomial can have more degrees of freedom, but I'm surprised at the decrease in accuracy for $Ax - b$.

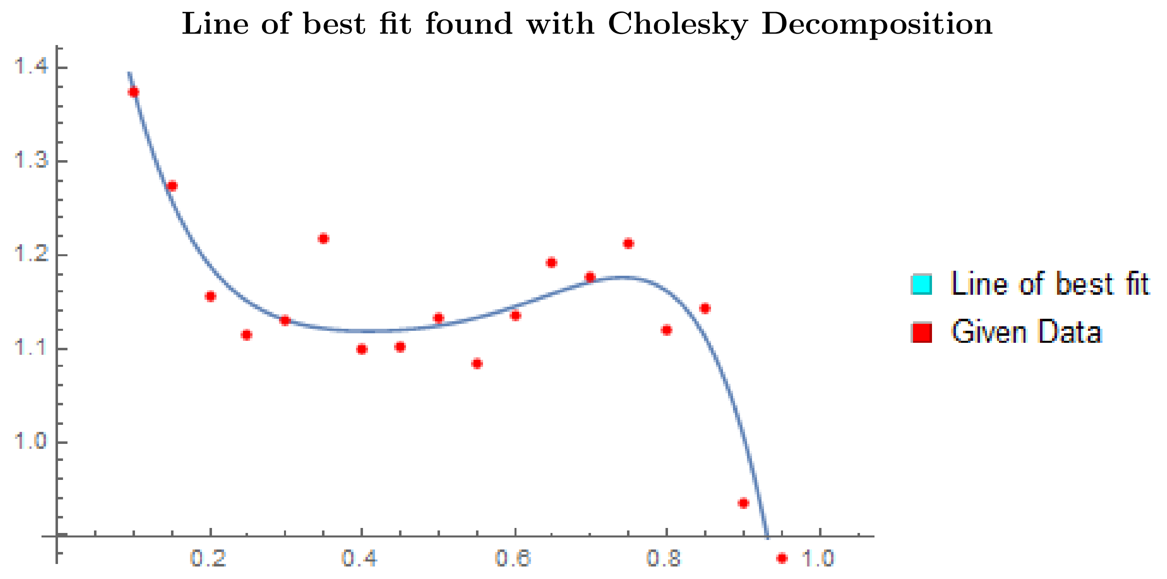

figure 2. Line of best fit vs. given data, single precision, 5th degree polynomial

Question 1.4
Discuss what should be the maximum degree of polynomials for the given data? Explain briefly what happens when you try to fit the data with a higher-degree polynomial, and why.

Solution: In general, the degree of your polynomial need not be higher than the number of points you have. Each increase in degree results in an increase of possible local minima and maxima of our function - a known result about polynomials. That is, a degree n polynomial has at most $n-1$ local extrema. Thus in the worse case scenario, such as the derivative of our polynomial changing in sign between every time step, we would require a degree $21-1=20$ polynomial. However, this is not the case for our particular data set. Instead the points plotted exhibit a cubic nature, e.g., one local minimum and one local maximum.

It is worth noting the operative words *at most* in the statement "a degree n polynomial has *at most* $n-1$ local extrema." For instance, in question 1.3 a 5th degree polynomial exhibited two local maxima. Thus from an analytic perspective, any degree works as long as the degree is at least one higher than the number of local maxima witnessed. So why the decrease in accuracy between the 3rd and 5th degree polynomials? I can only assume this is because of computer rounding. As we let $t \to 0$, t^5 is much smaller than t^3 - and thus more likely to be smaller than ϵ_{mach}. Being smaller than ϵ_{mach} means a computer will treat said value like 0 - I argue this is why higher degree polynomials aren't always better error wise, at least for our normal equations $Ax = b$.

Question 1.5
Discuss at what point does this algorithm fail (in single-precision floating point arithmetic)? Or, does the algorithm works always? Discuss.

Solution: This algorithm fails if there is a zero along the main diagonal, i.e., the matrix is singular. Similarly, if an entry along the main diagonal is less than ϵ_{mach} it will be considered zero by a computer, and the algorithm will fail. This is because the algorithm incorporates division by the main diagonal of our input matrix in a handful of places. As is well known, division by zero should be avoided.

In addition to the above, Cholesky requires the matrix A be Hermitian positive definite, as otherwise the square root will not be well defined. The way we created the matrix to be passed into our decomposition algorithm handles this, so we don't need to worry as it will always be SPD. This is because we have many a square root in our algorithm, (m of them to be precise), and as such a SPD matrix won't introduce imaginary numbers.

Question 2.1
Explain in reasonable detail how Householder QR works. Calculate explicitly the product of H_j with A at step j to show it casts A in the correct form. Show the new diagonal element of A is $-s_j$

Solution:

Householder QR works by zeroing out the elements below the diagonal in their respective columns. This is done by constructing the appropriate vector, v_j, creating the Householder H_j transformation from this vector, then applying the Householder reflector to A, our starting matrix. The cleverness of this approach is that the H_j are constructed such that they only zero out the $j-th$ column, leaving the previous changed columns alone. So, if we wish to compute explicitly H_j applied to A, we expect zeros under the main diagonal in column j. The columns whose index is less than j should also have zeros under the main diagonal. In the usual language, we're creating R with this process. Q can also be constructed from repeated applications of H_i, where i ranges from $1,\dots,n$. We know from that in class theory that such an H is of the form

$$H_j = I - 2v_jv_j^T$$

for $v_j = [0, 0, \dots, 0, a_{jj} + s_j, a_{j+1,j}, a_{j+2,j}, \dots, a_{m,j}]^T/||v_j||$.

Let's take

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \ddots & \dots & \vdots \\ a_{m1} & \dots & \dots & a_{mn} \end{bmatrix}$$

Then

$$H_jH_{j-1}\dots H_1A = \begin{bmatrix} \alpha_1 = a_1 - 2\frac{v_1^Ta_1}{v_1^Tv_1} & \alpha_2 & \dots & \alpha_n \end{bmatrix}$$

for columns a_i of A. Then on the main diagonal in position j, j we have

$$\hat{a}_{jj} = 2\frac{2v_j^T - a_j}{||v_j||}a_{jj} - 2\frac{2v_j^T - a_j}{||v_j||}s_j = -s_j$$

Question 2.2
Fit the data with a 3rd-degree polynomial, using single-precision floating point arithmetics. Report what the polynomial coefficients you find are, what the 2-norm error on the fit is, and provide a figure comparing the fitted curve with the data.

Solution: Requiring a 3rd degree polynomial and using single precision floating point arithmetics, I find the coefficients to be (full decimal representation kept since output.txt has been filled with higher degree solutions)

$$1.8319077733860329, \quad -5.1704640498919616, \quad 11.204369949907697, \quad -7.2851782508533018$$

Which is remarkably close to what was found using Cholesky. In fact, they only differ in the 12th or so decimal place. Then the line of best fit is (approximately) given by the function

$$f(t) = 1.831 - 5.170t + 11.204t^2 - 7.285t^3$$

Which when rounded is identical to the function found earlier.
The 2-norm of $Ax - b$ is

$$2.6149597066775007E - 015$$

As you can tell by the above error, this is a pretty good fit. This is further evidenced in the following figure:

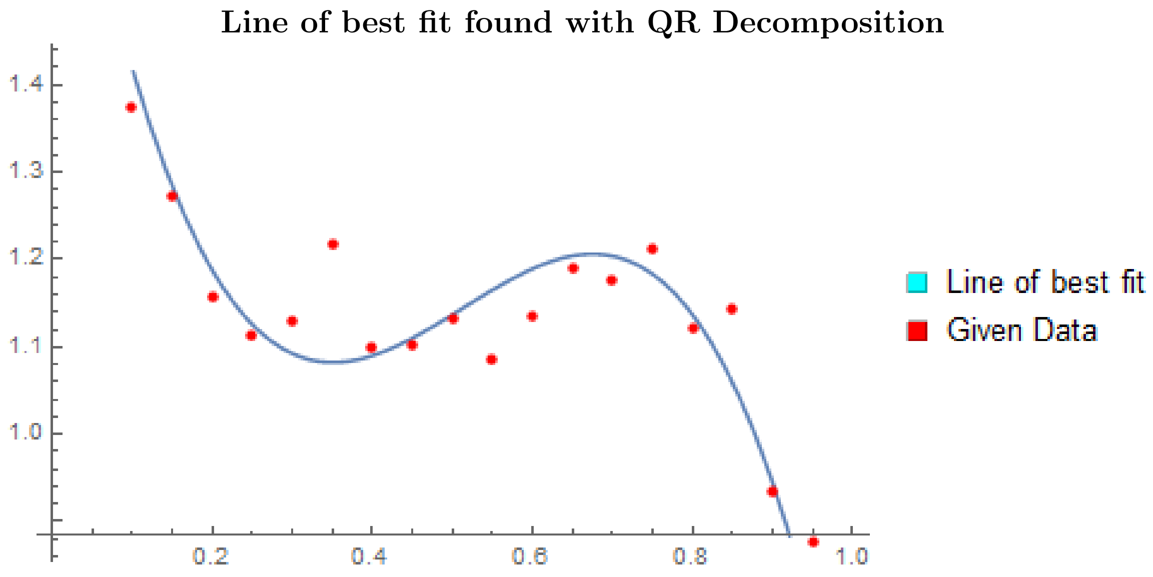

figure 3. Line of best fit vs. given data, single precision, 3rd degree polynomial

Question 2.3
Fit the data with a 5th-degree polynomial, using single-precision floating point arithmetics. Report what the polynomial coefficients you find are, what the 2-norm error on the fit is, and provide a figure comparing the fitted curve with the data.

Solution:
Requiring a 5th degree polynomial and using single precision floating point arithmetics, I find the coefficients to be

$$1.869 \quad -7.264 \quad 28.817 \quad -58.761 \quad 61.053 \quad -25.212$$

or the line of best fit is (approximately) given by the function

$$f(t) = 1.869 - 7.264t + 28.817t^2 - 58.761t^3 + 61.053t^4 - 25.212t^5$$

This may look identical to that received from Cholesky when truncated, but in reality they differ in about the 10th decimal place. This can be seen in output.txt, but in an effort to avoid death by significant figures they have been omitted here.

The 2-norm of $Ax - b$ is

$$1.2202500181627309E - 014$$

In comparison to our 3rd degree polynomial the 2-norm of $Ax - b$ is yet again larger. We witnessed this exact phenomenon in Cholesky, a higher degree polynomial resulted in a larger error for $Ax-b$. I assume this has to do with round off error as higher degrees of t get brought into play. However, I am still impressed the errors are so close as we've transformed a linear least square problem into a triangular least square problem. They indeed have the same solution (up to the 10th decimal place or so).

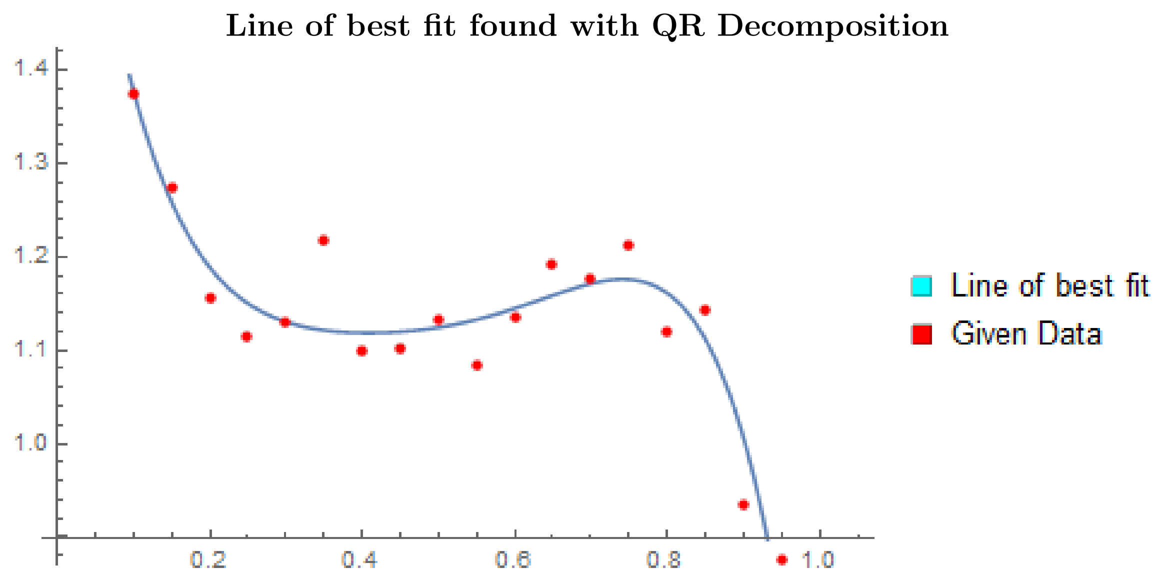

figure 2. Line of best fit vs. given data, single precision, 5th degree polynomial

Question 2.4
Discuss at what point does the algorithm fail (in single-precision floating point arithmetics)? Or, does the algorithm always work? Discuss.

Solution: This algorithm fails if there is a zero along the main diagonal, i.e., the matrix is singular. Similarly, if an entry along the main diagonal is less than ϵ_{mach} it will be considered zero by a computer, and the algorithm will fail. This is because the algorithm incorporates division by the main diagonal of our input matrix in a handful of places. As is well known, division by zero should be avoided.

In addition, we require symmetric positive definite matrices. This is because we take the square root of the diagonal entry in a handful of steps, so for this process to be well defined in the reals we do not want square roots of negative numbers. This is an artifact of this algorithm, and I do not believe it is a necessary and sufficient condition for a QR factorization to exist (in the sense that a matrix with negative diagonal entries could still have a representation as QR, just not in the sense of Householder). There probably exists another method of QR factorization, perhaps not as generalizable, that does not rely on roots of the diagonal entries. However, in the context of solving partiuclar problems certain pieces of this algorithm do seem necessary, such as Q being orthogonal so that it preserves vector norms.

Question 2.5
For the 5th order polynomial case, discuss your findings about the 2-norms of $A - QR$ and $Q^TQ - I$

Solution: Rather than write out 22 different column norms, which would be a bit overwhelming, I will present the largest of the column norms. For $A - QR$, the largest norm of a column was

$$2.918E - 015$$

which is pretty tiny. The largest column norm of $Q^TQ - I$ was

$$9.991E - 016$$

which again is very small. With upper bounds of error in a respective column so low, it is safe to assume we've found the desired factorization. In particular, Q is orthogonal to almost machine accuracy, and A has been factored into QR with upper triangular R to almost machine accuracy.