5M: Advanced Numerical Methods
Assessed Coursework 1
Least Squares Fitting

Yana Staneva – 2019862s

Semester 1, 2016–2017

'Least squares fitting' is a standard method, used to approximate solutions of *overdetermined* linear systems. By overdetermined we mean systems of equations with more equations than unknowns. The general overview of the method is to minimize the *residual* $\mathbf{r}$, defined by the difference between $\mathbf{b}$ and $A\mathbf{x}$, namely, $\mathbf{r} = \mathbf{b} - A\mathbf{x}$.

To be more precise, consider a matrix $A \in \mathbb{C}^{m\times n}$, where $m \geq n$ and a vector $\mathbf{b} \in \mathbb{C}^m$. We wish to minimize the 2-norm of the residual $\mathbf{r}$, that is, $\|\mathbf{r}\| = \|\mathbf{b} - A\mathbf{x}\|_2$, by finding an appropriate $\mathbf{x} \in \mathbb{C}^n$. Let us formalize this statement in the following theorem.

Theorem 1. Let $A \in \mathbb{C}^{m\times n}$, $m \geq n$, and let $\mathbf{b} \in \mathbb{C}^m$. If there exists a vector $\mathbf{x} \in \mathbb{C}^n$, which minimizes $\|\mathbf{r}\|_2$, then the following are equivalent:

- The vector $\mathbf{x}$ solves the Least Squares Problem.
- The residual $\mathbf{r}$ is perpendicular to the range of A, i.e. $\mathbf{r} \perp \text{range}(A)$ or equivalently, $A^*r = 0$.
- If A is of full rank, then $\exists!\mathbf{x}$, which solves the non-singular *Normal Equations* (1) given by

$$A^*A\mathbf{x} = A^*\mathbf{b}. \tag{1}$$

- The vector $\mathbf{x}$ is a solution of the system $P\mathbf{b} = A\mathbf{x}$, where $P \in \mathbb{C}^{m\times m}$ is the orthogonal projector, projecting onto range(A).

Consider the Normal Equations (1). We wish to derive an expression for $\mathbf{x}$. Left multiplication by a factor of $A^{-1}A^*$ gives

$$A^{-1}\underbrace{(A^*)^{-1}A^*}_{\mathbb{I}}A\mathbf{x} = A^{-1}(A^*)^{-1}A^*\mathbf{b} \quad \Longrightarrow \quad \underbrace{A^{-1}A}_{\mathbb{I}}\mathbf{x} = (A^*A)^{-1}A^*\mathbf{b} \quad \Longrightarrow \quad \mathbf{x} = \underbrace{(A^*A)^{-1}A^*}_{A^+}\mathbf{b}, \tag{2}$$

where we call the matrix $A^+ = (A^*A)^{-1}A^*$ the *pseudoinverse* of A. Thus, we have now simplified the least squares fitting problem to computing one of the following two vectors, namely,

$$\mathbf{x} = A^+\mathbf{b} \quad \text{or} \quad \mathbf{y} = P\mathbf{b},$$

where A^+ is the pseudoinverse of A and P – the orthogonal projector onto range A, as defined above.

Let $A \in \mathbb{C}^{m\times n}$ be of full rank. Then the system given by the normal equations $A^*A\mathbf{x} = A^*\mathbf{b}$ is $n \times n$ square, hermitian, and positive definite. Then the Least Squares Problem can be solved via the Normal Equations (1), using the Cholesky factorization of the matrix AA^*. The precise steps are described below in Algorithm (1).

Algorithm 1 (Least Squares via Normal Equations)**.** Given $A \in \mathbb{C}^{m\times n}$ and $\mathbf{b} \in \mathbb{C}^m$, then

1. Construct the matrix $T = A^*A$ and the vector $\mathbf{t} = A^*\mathbf{b}$.
2. Compute the Cholesky factorization of $T = R^*R$, where $R \in \mathbb{C}^{n\times n}$ is upper-triangular.
3. Compute the vector $\mathbf{w}$, which solves the lower-triangular system $R^*\mathbf{w} = A^*\mathbf{b}$, by $\mathbf{w} = (R^*)^{-1}A^*\mathbf{b}$.
4. Obtain the required vector $\mathbf{x}$, which solves the upper-triangular system $R\mathbf{x} = \mathbf{w}$, by $\mathbf{x} = R^{-1}\mathbf{w}$.

Thus, the Least Squares Problem is now reduced to solving the equations

$$RR^*\mathbf{x} = A^*\mathbf{b}, \tag{3}$$

where instead of working with A and A^*, we are working with the Cholesky decomposition RR^*, which is much more useful and efficient for numerical computations, as it is always stable and does not require pivoting, as shown in Lecture (6).

Alternatively, we can solve the Least Squares Problem by computing the *reduced* $\hat{Q}\hat{R}$ factorization of A, i.e. $A = \hat{Q}\hat{R}$. The matrix A is usually decomposed in reduced $\hat{Q}\hat{R}$ form by either Gram–Schmidt orthogonalization, covered in Lecture (3), or by *Householder triangularization* – a method, which achieves the desired upper-triangular matrix $\hat{R}$ by sequentially multiplying A on the left by $\hat{Q}_k$ matrices.

Let us now discuss the Householder triangularization algorithm in greater detail. Note that we will go over the *full* QR factorization, but in the end will provide explanation on how to obtain the reduced. We are given a matrix $A \in \mathbb{C}^{m\times n}$ and we want to form the QR factorization of A. The general aim of the method is to make R upper-triangular by multiplying A on the left by a sequence of Q_k matrices, where $k = 1, \dots, n$, such that zeros are introduced in the entries below the diagonal in the k-th column. That is, the matrix Q_k operates on rows $k, \dots, m$ in the k-th column. Thus, at step k there is a block of zeros in the first $k-1$ columns. At the end of the n-th step, all entries below the main diagonal are set to 0 and the product $Q_n \cdots Q_2Q_1A = R$ yields the required upper-triangular matrix R. Note that the multiplication by Q_k *does not* affect the already introduced zeros over the first $k-1$ steps.

In order to construct the matrices Q_k in the desired way, we require Q_k to be of the form

$$Q = \begin{bmatrix} \mathbb{I} & 0 \\ 0 & F \end{bmatrix},$$

where $\mathbb{I}$ is the $(k-1)\times(k-1)$ identity, and F is the $(m-k+1)\times(m-k+1)$ *Householder reflector*, defined as follows. Let $\mathbf{a} \in \mathbb{C}^{m-k+1}$ be the entries in rows $k, \dots, m$ of the k-th column of A, i.e.

$$\mathbf{a} = \begin{bmatrix} a_{k,k} \\ a_{k+1,k} \\ \vdots \\ a_{m,k} \end{bmatrix}.$$

Then the Householder reflector F acts on $\mathbf{a}$ by the rule

$$F\mathbf{a} = \begin{bmatrix} \|\mathbf{a}\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|\mathbf{a}\|\mathbf{e}_1,$$

where $\mathbf{e}_1$ is the standard basis vector. Geometrically, the Householder reflector F reflects $\mathbf{a}$ with respect to the hyperplane H orthogonal to $\mathbf{v} = \|\mathbf{a}\|e_1 - \mathbf{a}$, as illustrated below in Figure 1.

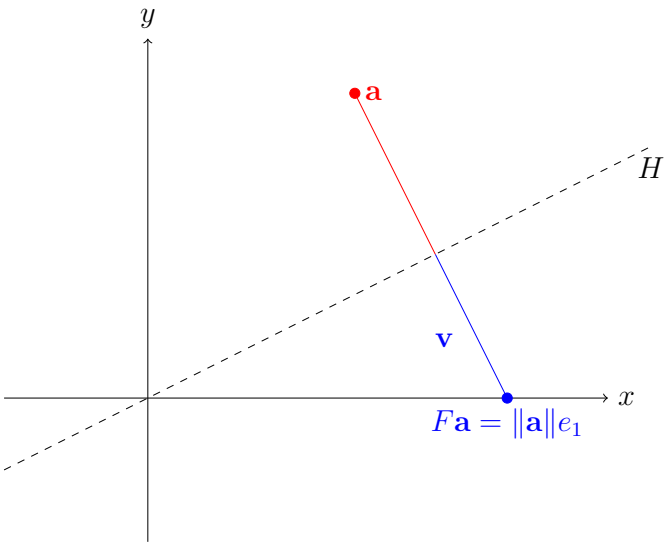


Figure 1: The Householder Reflector.

Analytically, the Householder reflector F maps any vector $\mathbf{x} \in \mathbb{C}^m$ by the rule

$$F\mathbf{x} = \left(\mathbb{I} - 2\frac{\mathbf{v}\mathbf{v}^*}{\mathbf{v}^*\mathbf{v}}\right)\mathbf{x} = \mathbf{x} - 2\mathbf{v}\left(\frac{\mathbf{v}^*\mathbf{x}}{\mathbf{v}^*\mathbf{v}}\right). \tag{4}$$

Hence, we obtain an expression for F, namely

$$F = \mathbb{I} - 2\frac{\mathbf{v}\mathbf{v}^*}{\mathbf{v}^*\mathbf{v}},$$

and thus,

$$Q = \begin{bmatrix} \mathbb{I} & 0 \\ 0 & \mathbb{I} - 2\dfrac{\mathbf{v}\mathbf{v}^*}{\mathbf{v}^*\mathbf{v}}, \end{bmatrix}.$$

We need to note that in general there exist two possible reflections – one with respect to the hyperplane H^+, and one with respect to H^- as illustrated below in Figure 2.

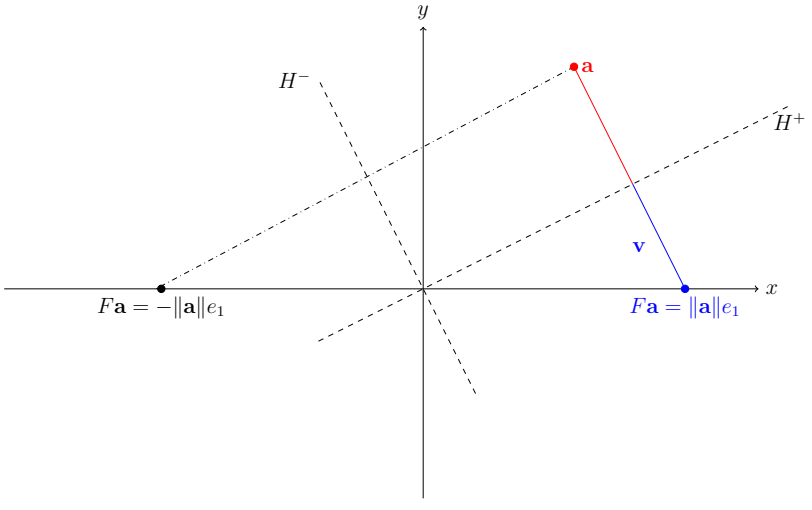


Figure 2: The two possible reflectors.

In order to ensure stability of the numerical scheme involved, we shall reflect $\mathbf{a}$ with respect to the hyperplane which is of greatest distance from the vector $\mathbf{a}$ itself. This aids to preserve the scalability of all quantities, that is, we avoid situations like $\mathbf{v} \ll \mathbf{a}$ when the angle between H^+ and e_1 is quite small. Hence, we use $\mathbf{v} = \text{sign}(a_1)\|\mathbf{a}\|e_1 + \mathbf{a}$ to avoid any ambiguities and we set $\text{sign}(a_1) = 1$ if $a_1 = 0$.

Now, what we have discussed so far employs the *full* QR factorization as we stated before. To ensure that we get the *reduced* form, we only consider the non-zero submatrix $\hat{Q}$ of the Q from the full matrix factorization and the upper-triangular submatrix $\hat{R}$ of R, which does not contain any zero rows.

The Householder reduced $\hat{Q}\hat{R}$ factorization is implemented in the MATLAB function `reducedQRHouseholder.m` as per Algorithm 10.1 in Trefethen & Bau. The reader is kindly invited to open the file and a short walk-through will be provided just now.

- The function takes as an input an $m \times n$ matrix A, which is to be entered in the console. Thus, we store the values of m and n for future use.
- Initiate the matrix $\hat{R}$ as A and the matrix $\hat{Q}$ as an $(m \times m)$ identity matrix with extra $(n - m)$ zero rows.
- Initialize an $(m \times n)$ matrix V to store the Householder reflector vectors, which are to be computed.
- Iterate over the n columns using the index j. We start the loop by extracting the first column of A, call it $\mathbf{x}$. At every next iteration we extract the sub-column of the continuously transformed $\hat{R}$, where we wish to set the entries below the main diagonal equal to zero.
- Construct the reflection vector $\mathbf{v}$, using the formula $\mathbf{v} = \mathbf{x} + \text{sign}(x_1)\|\mathbf{x}\|e_1$. Here we take care of the case when $x_1 = 0$ by setting $\text{sign}(x_1) = 1$ as errors occurred when the input matrix had a zero in the very first diagonal entry of A.
- Store the reflector vectors $\mathbf{x}$ as new vectors $\mathbf{v}$, which construct the matrix V to be later used. Then normalize the vector $\mathbf{v}$.
- Use the formula as in Equation (4) to clear the entries in the j-th column including and below the $(j+1)$-th entry of $\hat{R}$.
- Implicitly form the reduced $\hat{Q}$ matrix using Algorithm 10.3 as in Trefethen & Bau, while iterating in two nested loops. We extract the reflector vectors $\mathbf{v}$ from the matrix V and construct the i-th column of $\hat{Q}$ by the rule $\hat{Q}_{k:m} = \hat{Q}_{k:m} - 2\mathbf{v}_k(\mathbf{v}_k^*\hat{Q}_{k:m})$, where $k = n, n-1, \ldots, 1$. Note that in the outer loop the iterate is increasing, i.e. $i = 1, \ldots, n$ and in the inner – decreasing, $k = n, n-1, \ldots, 1$. Thus, at the first iteration, we get the first column of $\hat{Q}$. However, at the second iteration, we first populate entries including and below position $(2, 2)$ in the second column. The whole second column of $\hat{Q}$ is fully obtained only at the next iteration, and so on.
- Finally, make sure to extract the non-zero sub-matrix of the resulting R to get the *reduced* $\hat{R}$.

The Householder triangularization takes advantage of matrix-vector multiplication, instead of performing full matrix multiplication in each iteration. The most computationally expensive work is performed in the loop, which builds up the reduced upper-triangular matrix $\hat{R}$, that is,

$$\hat{R}_{j:m,j:n} = \hat{R}_{j:m,j:n} - 2\mathbf{v}(\mathbf{v}^*\hat{R}_{j:m,j:n}), \qquad j = 1, \ldots, n. \tag{5}$$

Let us now count the total number of flops. Given the initial $m \times n$ matrix $A \in \mathbb{C}^{m \times n}$, the very first iteration involves computations with every single entry of A, that is, $A_{1:m,1:n} = \hat{R}_{1:m,1:m}$. In the second

iteration, however, we work with $A_{2:m,2:n} = \hat{R}_{2:m,2:n}$, that is, with the sub-matrix formed by ignoring the first row and column of the original A. In the third iteration, we consider $A_{3:m,3:n} = \hat{R}_{3:m,3:n}$, and so on. Hence, at every iteration step j, we manipulate the matrix $A_{j:m,j:n} = \hat{R}_{j:m,j:n}$.

Now, at each iteration step j we perform

- $(m-j)(n-j)$ subtractions, that is, $\cdots = \hat{R}_{j:m,j:n} - \ldots$;
- $(m-j)(n-j)$ outer products, i.e. $\cdots - 2\mathbf{v} \times (\ldots$;
- $2(m-j)(n-j)$ vector-matrix products, namely, $\ldots (\mathbf{v} \times \hat{R}_{j:m,j:n}) \ldots$.

Adding all countable towards the total number of flops operations, we get $4(m-k)(n-k)$ operations performed at every iteration. Hence, summing over the total number of iterations, we get

$$\begin{aligned}\sum_{j=1}^{n} 4(m-j)(n-j) &= 4\sum_{j=1}^{n} \left(mn - j(m+n) + j^2\right) \\ &\approx 4\left(mn^2 - (m+n)\frac{n^2}{2} + \frac{n^3}{3}\right) \approx 4mn^2 - 2mn^2 - 2n^3 + \frac{4n^3}{3} \\ &\approx 2mn^2 - \frac{2n^3}{3}. \end{aligned} \tag{6}$$

Let us now compare the obtained result with a well-studied example. As we proved in Lecture (4), the Modified Gram–Schmidt orthogonalization requires $2mn^2$ flops. In order to get a better idea of how the two methods compare, suppose for a second that $m \approx n$, that is, our matrix A is (nearly) square. Then a crude approximation for the flops is given by

$$\begin{aligned}\text{Modified Gram–Schmidt:} &\quad \approx 2n^3 \\ \text{Householder Triangularization:} &\quad \approx 2n^3 - \frac{2n^3}{3} \approx \frac{4n^3}{3}.\end{aligned}$$

Thus, for approximately square matrices, the Householder transformation appears to be more computationally efficient. Nevertheless, given reasonable values of m and n, the Householder factorization is more stable in contrast with the modified Gram–Schmidt process.

Now we can proceed by analyzing the various methods which can be used to solve the Least Squares Problem. The reader is invited to open the MATLAB script `mainFile.m`. The script is sub-divided into sections, which are intended to be ran separately. The first one can be used to test the `reducedQRHouseholder.m` function, but it requires a matrix A to be input in the console. The second one solves the Least Squares Problem, based upon the following three methods:

- Computing the reduced $\hat{Q}\hat{R}$ factorization of A using Householder triangularization,
- Factorizing A in its reduced $\hat{Q}\hat{R}$ form by the modified Gram–Schmidt algorithm, and
- Solving the Normal Equarions (1) using Cholesky factorization.

Let us quickly discuss the operation count for solving the Least Squares Problem using Householder triangularization to form the reduced $\hat{Q}\hat{R}$ factorization of A. As shown above the work is dominated by the construction of the reduced upper-triangular matrix $\hat{R}$, given in (5). Further, the number of flops required to perform the Householder triangularization is $\approx 2mn^2 - \frac{2n^3}{3}$ as derived above in (6). The implementation of Algorithm 11.2, however, also requires the computation of the vector $\mathbf{y} = \hat{Q}^*\mathbf{b}$, which requires $\approx n(2m-1)$ flops. Finally, solving the upper-triangular system $\hat{R}\mathbf{x} = \mathbf{y}$ using backward substitution costs $\approx n^2$ flops. Adding these up yields

$$\approx 2mn^2 - \frac{2n^3}{3} + 2mn - n + n^2 \approx 2mn^2 - \frac{2n^3}{3} + \text{L.O.T.}$$

where by L.O.T. we mean lower order terms. Thus, indeed, we confirm the statement in Trefethen & Bau that solving the Least Squares problem is dominated by the $\hat{Q}\hat{R}$ factorization via Householder triangularization.

Next we demonstrate results obtained by applying the MATLAB script `mainFile.m` to a specific problem. Consider the interval $\mathbf{t} = [0, \ldots, 1]$ partitioned into equally spaced grid points. Let t_i be the i-th grid point of $\mathbf{t}$, where $i = 1, \ldots, 50$. Then let A be the Vandermonde matrix, associated with the elements of $\mathbf{t}$, and let $\mathbf{b}$ be the column vector with entries $b_i = \sin(5t_i)$, that is,

$$A = \begin{bmatrix} t_1^0 & t_1^2 & \cdots & \cdots & t_1^{49} \\ t_2^0 & t_2^2 & \cdots & \cdots & t_2^{49} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & t_{50}^2 & \cdots & \cdots & t_{50}^{49} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 1 & t_2^2 & \cdots & \cdots & t_2^{49} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & \cdots & \cdots & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} \sin(5t_1) \\ \sin(5t_2) \\ \sin(5t_3) \\ \vdots \\ \sin(5t_{50}) \end{bmatrix} = \begin{bmatrix} 0 \\ \sin(5t_2) \\ \sin(5t_3) \\ \vdots \\ \sin(5) \end{bmatrix}.$$

Now the reader is invited to run the subsection %% Least Squares Question 5 within the MATLAB script `mainFile.m`, which outputs the fitted coefficients $\mathbf{x}$, while solving the system $A\mathbf{x} = \mathbf{b}$, using the three methods discussed above. The results are listed below in Table 1.

$\mathbf{x}$	Householder	Gram–Schmidt	Cholesky
x_1	-0.00000	-0.00000	-0.00000
x_2	5.00001	5.00001	5.00002
x_3	-0.00058	-0.00057	-0.00077
x_4	-20.82265	-20.82285	-20.81975
x_5	-0.10278	-0.10123	-0.12532
x_6	26.63066	26.62369	26.73383
x_7	-2.15866	-2.13898	-2.45471
x_8	-10.25414	-10.29007	-9.70638
x_9	-8.55154	-8.50920	-9.20453
x_{10}	14.58125	14.55016	15.06566
x_{11}	-6.14271	-6.12977	-6.34613
x_{12}	0.86222	0.85989	0.89915

Table 1: Output from running `mainFile.m` with 5-digit floating point.

The figures in red indicate when the three methods start deviating from each other in calculating the entries of $\mathbf{x}$. Clearly, the method which solves the normal equations using Cholesky factorization differs the most from the other two and we can assume that it is the most susceptible to rounding errors. For example, consider x_3. Then Householder (-0.00058) and Gram–Schmidt (-0.00057) differ only in the very last digit, while the output from Cholesky (-0.00077) differs by two, etc. Thus, at this point we can conclude that Cholesky is more susceptible to rounding errors.

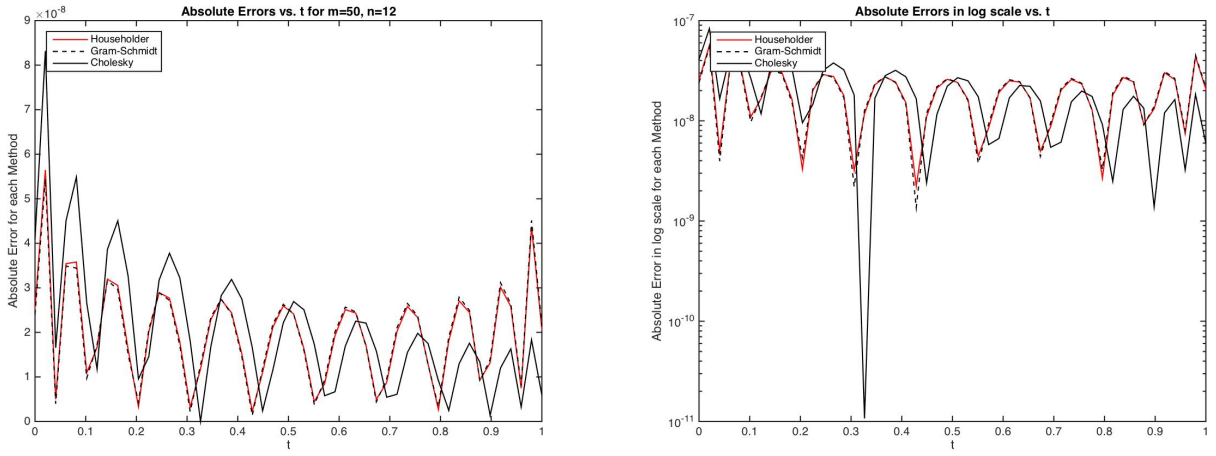
When discussing the accuracy of a numerical method, we usually rely on measuring the *absolute* and *relative* errors of the scheme. Below we shall present plots, which illustrate how both errors vary with the number of iterations executed, but beforehand we proceed by presenting the formulas, used

to calculate both errors.

$$\text{Absolute Error: } a_e(t) \coloneqq |\tilde{f}(t) - f(t)| = |A\mathbf{x} - \mathbf{b}|,$$

$$\text{Relative Error: } r_e(t) \coloneqq \frac{\|\tilde{f}(t) - f(t)\|}{\|f(t)\|} = \frac{\|A\mathbf{x} - \mathbf{b}\|}{\|\mathbf{b}\|}.$$

It is important to note that the code has been run on three different machines – MacBook, Lenovo B70, and Dell Optiplex 7010 (pc in Labs). But the plots presented below have been compiled on the MacBook. In all plots the red graph is the error of the $\hat{Q}\hat{R}$ via Householder method, the dashed line is the error of the $\hat{Q}\hat{R}$ via Gram–Schmidt, and the solid black line is the Normal Equations via Cholesky approach.



(a) Absolute Error plotted vs. $\mathbf{t} = [0, 1]$.

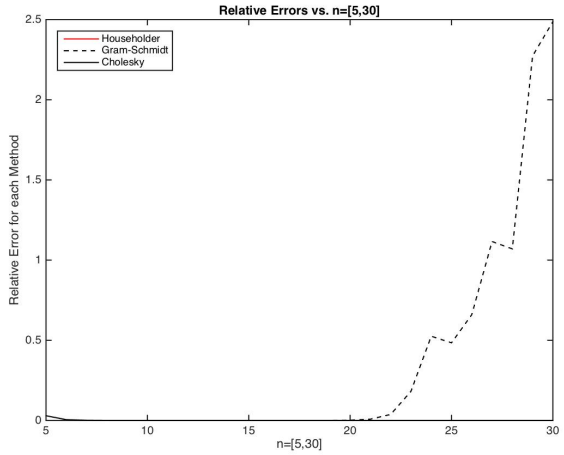
(b) Log scale of Absolute Error vs. $\mathbf{t} = [0, 1]$.

Figure 3: Plots of the absolute error in usual and log scale.

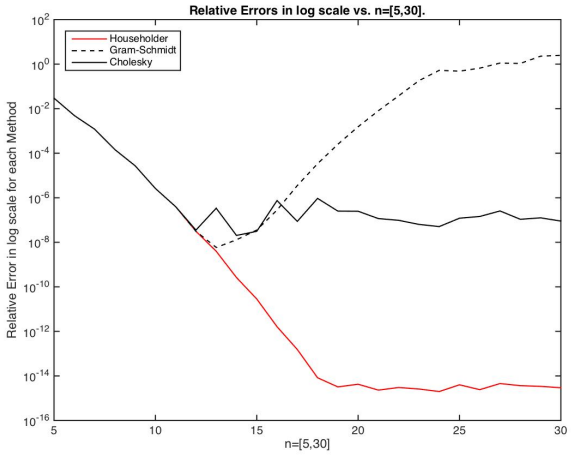
According to Figure 3a the Cholesky implementation yields a larger error in the first several iterations, but tends to decrease towards the final ones, while the Householder and Gram–Schmdit methods perform quite similarly. The log-scaled graph of the error in Figure 3b shows that there is a huge increase in the scale of the precision for values of $t \approx 0.32$. The inconsistency occurs at a scale of $\approx 10^{-11}$, which is quite lower in comparison to the errors produced by the Householder and Gram–Schmidt implementations. Note that this huge increase in the accuracy was only observed on the MacBook machine and thus, we can suppose that it occurs due to a larger machine precision. On the other hand, we can clearly see from both sub-figures in Figure 3 that the $\hat{Q}\hat{R}$ factorizations using Householder and Gram–Schmidt perform quite similarly.

Now, of course, we have examined the accuracy of solving the Least Squares Problem via the Normal Equations *only* by constructing the Cholesky factorization of the matrix A. The computational time for this method appears to be a bit faster than the other two methods (has not been precisely measured, we are just relying on general observations by running the code). According to both plots in Figure 3 the Cholesky method yields a higher absolute error for values of $t_i < 0.6$, but then somewhat stabilizes, while the Householder and Gram–Schmidt start off with a high value, then decrease a bit, but end up with a high absolute error as well. Thus, we need to compare the three methods using other parameters, in order to gain a better understanding of the performance and susceptibility to rounding errors.

Let us now draw the attention of the reader to the plots of the relative error of each method. The following are presented below in Figure 4.



(a) Relative Error plotted vs. $5 \leq n \leq 30$.



(b) Log scale of Relative Error vs. $5 \leq n \leq 30$.

Figure 4: Plots of the relative error in usual and log scale.

The relative error is computed in a slightly different way, compared to the absolute error. Here we have considered polynomials of lower powers. This is achieved by forming the Vandermonde matrix using the following iterative process: let $n \in \mathbb{Z}$, $n = 5, \ldots, 30$, and consider the Vandermonde matrix V with dimensions $50 \times n$. Thus, we compute the relative error of each of the three methods for solving the least squares problem at the n-th iteration, using the matrix $V(50, n)$.

Now, Figure 4a is a plot of the relative errors of each method vs. the iterated integer n. Clearly, the Gram–Schmidt method produces the highest relative error for $n > 20$, but we do not get a clear understanding of what is going on with the Householder and Cholesky methods. Thus, we list the calculated relative errors in Table 2 below.

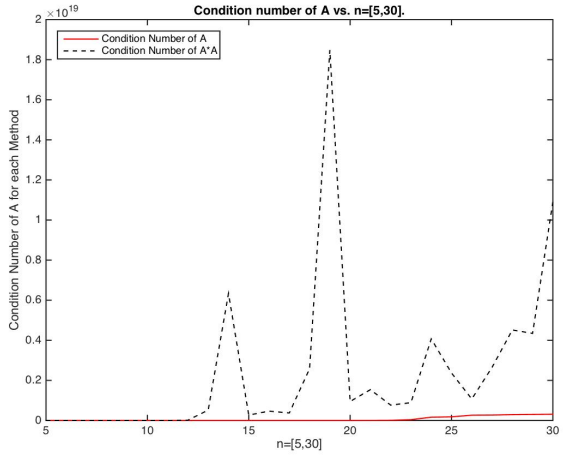
Upon inspection of the values listed in Table 2, we see that the Gram–Schmidt method indeed blows up for large values of n, the relative error of the Cholesky method is really small, and the Householder method produces almost negligible errors. This is also clearly indicated in Figure 4b, where we use a logarithmic scale. *Hence, when considering the relative error of each method, we find out that the Gram–Schmidt is the most susceptible to rounding errors, followed by the Cholesky method and finally the Householder approach is the least subjective to computational inconsistencies.*

Value of n	r_e Householder	r_e Gram–Schmidt	r_e Cholesky
$n = 5$	0.0296565561	0.0296565561	0.0296565561
$n = 6$	0.0049396175	0.0049396175	0.0049396175
$n = 7$	0.0012004091	0.0012004091	0.0012004091
$n = 8$	0.0001459159	0.0001459159	0.0001459159
$n = 9$	0.0000273699	0.0000273699	0.0000273699
$n = 10$	0.0000026128	0.0000026128	0.0000026128
$n = 11$	0.0000003973	0.0000003973	0.0000003973
$n = 12$	0.0000000311	0.0000000311	0.0000000354
$n = 13$	0.0000000040	0.0000000059	0.0000003391
$n = 14$	0.0000000003	0.0000000130	0.0000000203
$n = 15$	0.0000000000	0.0000000356	0.0000000312
$n = 16$	0.0000000000	0.0000002793	0.0000007470
$n = 17$	0.0000000000	0.0000035324	0.0000000869
$n = 18$	0.0000000000	0.0000339056	0.0000009352
$n = 19$	0.0000000000	0.0002521531	0.0000002519
$n = 20$	0.0000000000	0.0015482363	0.0000002478
$n = 21$	0.0000000000	0.0080813234	0.0000001161
$n = 22$	0.0000000000	0.0377605187	0.0000000973
$n = 23$	0.0000000000	0.1815876186	0.0000000636
$n = 24$	0.0000000000	0.5251835227	0.0000000508
$n = 25$	0.0000000000	0.4842414740	0.0000001208
$n = 26$	0.0000000000	0.6596997856	0.0000001445
$n = 27$	0.0000000000	1.1159322905	0.0000002554
$n = 28$	0.0000000000	1.0701229956	0.0000001079
$n = 29$	0.0000000000	2.2732454441	0.0000001253
$n = 30$	0.0000000000	2.4869800762	0.0000000899

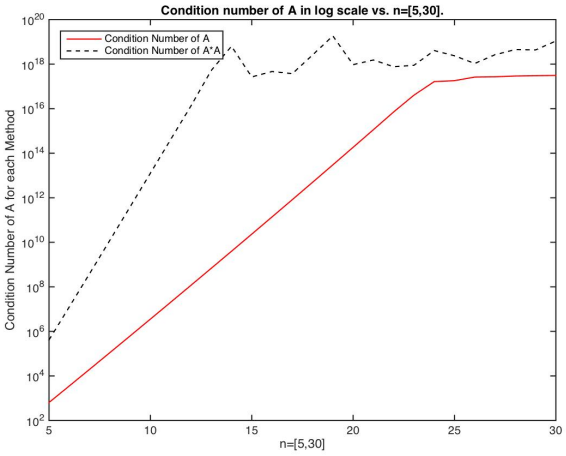
Table 2: Relative errors for each method with 10-digit floating point.

Finally, we discuss the *condition number* κ of the matrix A and the product A^*A. The condition number is defined as the ratio between the largest and the smallest singular values of the respective matrix. Once again, we are using the iterative approach described above to compute the n-th condition number of each Vandermone matrix V with dimensions $50 \times n$. The plots are displayed in Figure 5.

Figure 5a shows the graphs of the condition numbers of A and A^*A in the usual scale. We can see that the condition number of A, given by the red graph, does not vary much. On the contrary, the condition number of the product A^*A fluctuates quite a lot. When we observe Figure 5b we can infer that the condition number of A increases steadily up to $n \approx 24$, but then stabilizes. The condition number of A^*A increases at an even higher rate up to a much smaller value of n at $n \approx 14$, and does not even stabilize after that point. This leads to the conclusion that in general methods which rely on computing A^*A are more unstable as the matrix A^*A itself is more sensitive to perturbations. Thus, we can relate this to our previous analysis by pointing out that Cholesky has a higher relative error compared to Householder since it relies on the factorization of A^*A. Also, we can conclude that the instability of the Modified Gram–Schmidt is due to issues, caused by machine precision, i.e. rounding errors, overflow, etc.



(a) Condition numbers plotted vs. $5 \leq n \leq 30$.



(b) Log scale of Condition Numbers vs. $5 \leq n \leq 30$.

Figure 5: Plots of the relative error in usual and log scale.

Solving the Normal Equations via Cholesky factorization is the only method we have used, which relies on numerical computations involving A^*A. Both the Householder and the Gram–Schmidt method involve factorization of A solely. If we only consider the graphs in Figure 4 we might rule out the Gram–Schmidt method completely, since it proves to be very unstable upon analyzing the graph of the relative error it produces, which might be due to floating point overflows, occurring during the orthogonalization of the vectors. Inspecting Figure 5 on the other hand suggests that the Cholesky method is subject to a large number of numerical inconsistencies, as it relies on factorizing the product A^*A, which is much more sensitive to perturbations, compared to the Gram–Schmidt and Householder approaches, which factorize A instead. Thus, we can conclude that it is necessary to perform multiple checks and analyze as many as possible parameters, which provide information about each method's stability and sensitivity to rounding errors. In both cases, however, we can confirm that the Householder method proves to be the most efficient and least subjective to rounding errors. First, it relies only on factorizing A, which has a smaller and more stable condition number. Second, it requires a smaller number of flops, which decreases the probability of having more computational inaccuracies due to rounding errors. And finally, it is quite consistent while analyzing its absolute errors. In conclusion, solving the Least Squares Problem via reduced $\hat{Q}\hat{R}$ factorization by Householder triangularization is the most efficient and least susceptible to rounding errors method from the ones used.