Computational Linear Algebra CW1

JIAQI LI

5^{th} November 2021

1 Question2

1.1 question a

After QR-decomposition, R is actually only valid for the first three lines after we ignore accuracy higher than 8 decimal places, and every other line is 0, that means the value is too small to ignore. So it's really only the first three columns of Q that are valid when you multiply them, the other columns are just multiplying 0, and just get 0.
By $C = Q * R$, we can also find the characteristics of C. By this the number after three lines of R is too small to ignore, so rank of C is equal to three.

1.2 question b

By $C = Q * R$, we found in 1 (a) that R only has numbers in the first three rows after ignoring accuracy higher than eight decimal places, and all the other rows are zeros, so Q is actually only valid when you multiply the first three columns, all the other columns are multiplied by zeros to get zero. So we can store the first three columns of Q as a basis, and the first three rows of R as coefficients, as a C for the new, smaller memory.
For example, try the last column of C. Firstly, look at the first three rows of Q, and each row is regarded as a vector as a basis. The first three rows of the last column of R are regarded as a vector as the corresponding coefficient of these three basis. Then the last column of C can be obtained after multiplication.
The code can be found in CW1.py in cw1, the function is compress(C), and the test function is test_compress() in test_cw1 folder.

1.3 question c

We can do row operation on C, and then look at the remaining rows that their entries are not 0 which is the rank of C, and that makes it more obvious to show that $rank(C) = 3$.

2 Question3

2.1 question a

Once Q and R are worked out, let's say that the coeffients I want is x, then $QRx = f$, $Rx = Q^*f$ and a solve_triangular can do the work.
The code is stored in folder cw1 in CW1.py.
The Classical gram-Schmidt Method uses function GS_classicalnew(A), the Modified gram-Schmidt Method uses function GS_modifiednew(A), The Householder method uses the function householder_ls(A) of exercise3 directly.

2.2 question b

Modified G-S's x_2 is the same as Householder's x_3, but the Classical G-S's x_1 is different.
Reason: The classical Gram-Schmidt algorithm of QR decomposition is not stable compared with the modified Gram-Schmidt and Householder algorithm. For example, if α_1 and α_2 are nearly parallel, then $\alpha_2 - \gamma_{12}q_i$ is approximately to be zero, then the roundoff error becomes obvious. But the other like Modified G-S Method is insensitive to rounding error and has a small error.

3 Question4

3.1 question a

This method store the total V, but the reduced matrix only needs A.
In the original householder, we sought all v_k, but m was much larger than n, so we have to store a lot of v_k, and we just need the reduced QR, and we don't need the V for computing the reduced QR, so there's a lot of waste.

3.2 question b

For an $m \times n$ matrix, after QR-factorization, R is an upper triangular matrix and entries below the diagonal are 0.
While the length of $v_k = m - k$ from $k = 0$ to $n - 1$, then we can store v_k column by column in the lower triangle of R.
Then there is a problem: the diagonal(not exactly the diagonal just means the line of i=j) has a value in the original R, and should represent the first number of v_k in each column, where v_k and R overlap.

To solve this problem, we can divide each column v_k by the first entry of each column, so that the newly constructed v_{knew} are all new vectors with the first entries are 1, such that $v_{knew}[i] = \frac{v_k[i]}{v_k[0]}$. Therefore, we do not need to store the first row of each v_{knew}, because the first row of all v_{knew} is now 1, so it will not overlap with the upper triangle part of the original R that has a value.

There is another solution, we can directly add the combination of v_k from below to the original R, and resulting in one more line than the original R, which means that put each row in the previous column.

3.3 question c

Compared with householder's Q^*b, we can find that there is almost no difference between them, so there is no problem with this method. This approach is less complex than storage compared to forming Q explicitly.

The code can be found in CW1.py in cw1, the function is Rv_array(A), and the test function is test_Rv_array(m, n) in test_cw1 folder.

3.4 question d

Describe this algorithm: We can restore Rv to distinct V_k because starting from row $K + 1$ to the end row the Column k of Rv is V_k (the first row and column are all represented as row 0 and column 0), and then we apply the same operations to Q^*b that are applied to the column of A, i.

The difference between the values of Q^*b obtained by these two methods is too small to be ignored, so this method is correct. The n^{th} column of householder(A_hat, n) is directly Q^*b, which is also very convenient, but requires more storage space and need to use another function householder() to compute R, it means that it is necessary to use A and b to find out A_hat first, but the current method only needs to store one Rv to contain all the required information.

The code can be found in CW1.py in cw1, the function is Q_starb(R_v, b), and the test function is test_Q_starb(m, n) in test_cw1 folder.

3.5 question e

Describe this algorithm: Rv is an $m \times n$ matrix. First, take out the upper triangle of Rv, which is the original R, and the first n rows of R are R_hat. In the least square Problems, we learned that $\hat{R}x = \hat{Q}^*b$. We can use the function Q_starb(R_v, b) in part d to find the Q^*b, and $\hat{Q}^*b$ is the first n row of Q^*b. Finally, we can use the function solve_triangular to solve the x in $\hat{R}x = \hat{Q}^*b$.

The difference between the values of x obtained by these two methods is too small to be ignored, so this method is correct. This method now needs less storage and is more convenient. We needed to make householder() with parameter A, b, get R, get $\hat{R}$ and $\hat{b}$, which requires a lot of storage and a little bit more complex, but this method just needs an Rv array to retrieve R and find $\hat{R}$, here we do not need to use any other function in it to calculate R, we can just output the the upper triangle of the Rv array which is R exactly, then

we can find x directly by writing $\hat{R}x = \hat{Q}^* b$.

The code can be found in CW1.py in cw1, the function is $R\_v\_ls(R\_v, b)$, and the test function is $test\_R\_v\_ls(m, n)$ in test_cw1 folder.

4 Question5

4.1 question a

We want to find out the minimum of $f(x) = \|Ax - b\|^2$ 2 under the requirement that $\|x\| = 1$.
By introducing a Lagrange multiplier $\lambda \in \mathcal{R}$ we can get that

$$\Phi(x, \lambda) = \|Ax - b\|^2 + \lambda(x^T x - 1) = \|Ax - b\|^2 + \lambda(\|x\|^2 - 1) \tag{1}$$

Because if we have sufficient regularity for $f(x), g(x)$, the Lagrange formula

$$\Phi(x, \lambda) = f(x) + \lambda g(x) \tag{2}$$

is used to determine the stationary point (for the min/max) of $f(x)$ such that $g(x) = 0$.

4.2 question b

By question a we can find that

$$min\,\|Ax - b\|^2 = min(Ax - b)^T(Ax - b) = min\,\|Ax - b\|^2 + \lambda(\|x\|^2 - 1) \tag{3}$$

and $x^T x = \|x\|$.
We can find that these points are the solutions of $\nabla\Phi = 0$
Then

$$\frac{\partial}{\partial x}\Phi(x, \lambda) = (2A^T A + 2\lambda I)x - 2A^T b = 0 \tag{4}$$

and

$$\frac{\partial}{\partial \lambda}\Phi(x, \lambda) = x^T x - 1 = 0 \tag{5}$$

So, $2A^T Ax + 2\lambda Ix - 2A^T b = 0$ and $x^T x - 1 = 0$.
For $\|x\| = 1$ is the required condition, the solution of $\frac{\partial\Phi}{\partial x}$ is $2A^T Ax + 2\lambda Ix - 2A^T b = 0$ which is equivalent to $A^T Ax + \lambda Ix - A^T b = 0$.

4.3 question c

First we can assumde that A is an $n \times m$ matrix. By part(b), we can find that the solution is $A^T Ax + \lambda Ix - A^T b = 0$, and use the QR factorisation of A, we can write A as $A = QR$.
Then the equation $A^T Ax + \lambda Ix - A^T b = 0$ is equivalent to $A^T Ax + \lambda Ix = A^T b$,

so we can get that $(R^TQ^TQR + \lambda I)x = A^Tb$.
By this,

$$x = (R^TQ^TQR + \lambda I)^{-1}A^Tb \tag{6}$$

By applying the Woodbury matrix identity, we can write the $(R^TQ^TQR + \lambda I)^{-1}$ first.

$$(R^TQ^TQR + \lambda I)^{-1} = (\lambda I_m)^{-1} - (\lambda I_m)^{-1}R^T(I_n^{-1} + R(\lambda I_m)^{-1}R^T)^{-1}R(\lambda I_m)^{-1} \tag{7}$$

$$= \frac{1}{\lambda}I_m - \frac{1}{\lambda}I_mR^T(I_n + R\frac{1}{\lambda}I_mR^T)^{-1}R(\frac{1}{\lambda}I_m) \tag{8}$$

So come back to the x expression, we can find that

$$x = (\frac{1}{\lambda}I_m - \frac{1}{\lambda}I_mR^T(I_n + R\frac{1}{\lambda}I_mR^T)^{-1}R(\frac{1}{\lambda}I_m))R^TQ^Tb \tag{9}$$

$$= (\frac{1}{\lambda}I_m - \frac{1}{\lambda^2}I_mR^T(I_n + R\frac{1}{\lambda}I_mR^T)^{-1}RI_m)R^TQ^Tb \tag{10}$$

$$= (\frac{1}{\lambda}I_m - \frac{1}{\lambda^2}R^T(I_n + \frac{1}{\lambda}RR^T)^{-1}R)R^TQ^Tb \tag{11}$$

$$= \frac{1}{\lambda}R^TQ^Tb - \frac{1}{\lambda^2}R^T(I_n + \frac{1}{\lambda}RR^T)^{-1}RR^TQ^Tb \tag{12}$$

$$= R^T(\frac{1}{\lambda}I_m - \frac{1}{\lambda^2}(I_n + \frac{1}{\lambda}RR^T)^{-1}RR^T)Q^Tb \tag{13}$$

$$= R^T(\frac{1}{\lambda}I_m - \frac{1}{\lambda}(\lambda I_n + RR^T)^{-1}RR^T)Q^Tb \tag{14}$$

$$= R^T((\lambda I_n + RR^T)^{-1}(\frac{1}{\lambda}I_n(\lambda I_n + RR^T) - \frac{1}{\lambda}RR^T))Q^Tb \tag{15}$$

$$= R^T((\lambda I_n + RR^T)^{-1}(I_n + \frac{1}{\lambda}RR^T - \frac{1}{\lambda}RR^T))Q^Tb \tag{16}$$

$$= R^T(\lambda I_n + RR^T)^{-1}Q^Tb \tag{17}$$

In the case of $m >> n$ this formula is beneficial to the original minimization problem.
Reason: The first solution is R^TR, which is an $m \times m$ matrix, and the new solution is RR^T, which is an $n \times n$ matrix. The second solution, if we want to be simpler than the first solution, we have to make m smaller than n, so that there will be less store data.

Algorithm: Use the function solve_x(A, b, lam) to find out the x. Assuming that A is a matrix of $n \times m$ form, we use householder_qr(A) to work out Q and R. Then through the expression of X deduced previously, we can work out X.

4.4 question d

Here, we can first find x using the function calculated by part(c) based on the input A, B and λ. Because we need $\|x\| = 1$, we can work out x first. If the

absolute value of the difference between x and 1 is larger than the accuracy that we can accept, then we need to use the while loop to cycle through this step until the absolute value of the difference is less than the accuracy we can accept, and we can output λ.
The reason we can use $\lambda = \lambda \|x\|$ in the loop is that according to the part (c) we can find that when the λ becomes larger then x will become smaller, so in the process if $\|x\| < 1$, we need to make x larger, with $\lambda = \|x\| \lambda$ which will make λ smaller.

4.5 question e