Math 477/577 Project Report

Xi Sun
Komal Sandhu

November 28, 2018

Contents

1 Introduction

1.1 Problem Statement

Compare and contrast the stability and accuracy of SVD with various QR decomposition methods. How does the forward and backward error grow with the number of columns? What about the number of rows?

1.2 Motivation

Singular Value Decomposition (SVD) and QR decomposition methods such as Classical Gram-Schmidt (CGS), Modified Gram-Schmidt (MGS), and Householder Triangularization (HH) are useful approaches when solving linear systems, least square problems, and eigenvalue problems. Such algorithms are able to simplify the problem through transformations while still preserving the solution. In doing so, however, it is noted that the forward and backward error can cause the stability and accuracy of the solution to change from method to method.

In order for an algorithm to be numerically stable, any small perturbations caused by instances such as truncation errors would ideally reduce or decay over time or steps. This would eventually lead to having little effect on the final result and would thus the algorithm will be more stable. Otherwise, as these small perturbations grow over time, the algorithm will be deemed unstable. Furthermore, an algorithm may be able to provide a solution, but it is important to observe whether such solution is accurate enough. Although such methods may not present an output with the exact solution, some methods may vary in how accurate or close it is to the exact solution. When discussing stability and accuracy, it is also important to mention that forward errors (error in the output) and backward errors (errors in the input) can play major roles in explaining whether a method is stable or unstable and accurate or inaccurate. Any minor error could grow over time to cause instability in the system and end in inaccurate results. Therefore, it is significant to consider errors in order to compare stability and accuracy of algorithms to see which algorithm(s) would be ideal in real world applications.

1.3 Goal

For this project, the goal is to find the stability, accuracy, and forward and backward error for various algorithms such as Classical Gram-Schmidt, Modified Gram-Schmidt, Householder Triangularization, and Singular Value Decomposition. These algorithms are important topics in areas such as data analysis, machine learning, and artificial intelligence. Thus, the purpose and goal for this project is to see which algorithm performs the best in regard to stability and accuracy as well as any forward and backward error that may arise.

2 Method Analysis

In order to test the stability, accuracy, and forward and backward error for the various algorithms, random 50×50 matrices are built and used. Although there is a different 50×50 matrix being used each time (all of which will produce different solutions), the output error will not change drastically if all the elements in the random matrices are in some small range. Therefore, for the rest

of the project, it is assumed that the values for each random matrix is restricted in the interval $[0, 1]$.

2.1 Stability

For the stability test for CGS, MGS, and HH, Q and R are constructed with Q being a random unitary matrix and R being a random upper triangular matrix. Then, the forward and backward errors are checked and verified. Also, a small perturbation is added to the matrices, and if the forward and backward error also being very small (close to machine epsilon $= 10^{-16}$), then we can say that this algorithm(CGS, MGS, HH) is stable. Same thing for the SVD algorithm, the random unitary matrices for U and V will be constructed, as well as a random diagonal matrix Σ containing the singular values being arranged from the largest to the smallest. And the method about how to check the stability is pretty much the same as what needs to be done for QR decomposition.

2.2 Accuracy

In order to verify accuracy of each algorithm, linear system $Ax = b$ is solved. The vector $x \in \mathbb{R}^{50}$ is constructed where all entries are 1:

$$x = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Therefore, the entry b_i in the vector b will be the sum of the ith row in matrix A. By constructing the vector x in the above form, it is easy to see how the output differs from the true value of x. In order to show this in a better way, square distances from the output error to the original error are compared.

2.3 Forward and Backward Error

In order to observe how the forward and backward errors grow with the number of columns, the rows are fixed to 200 and the columns are set from 1 to 200. Then, the forward error for Q and R for the respective algorithm and backward error for the respective algorithm are calculated and plotted.

In order to observe the forward and backward errors for the different number of rows, the columns are fixed to 50 and the rows are set from 50 to 200. Then the forward error for Q and R for the respective algorithm and backward error for the respective algorithm are calculated and plotted.

All in all, the conclusion that needs to be derived from here is the following: If the backward error is small enough(ie, close to machine epsilon), then the algorithm is backward stable. And we want to check in detail how do those conclusions being compared in section 2.1(in this project, not in the textbook).

3 Results and Analysis

3.1 Stability

We start this part by constructing random matrix, like mentioned in the previous section. We constructed random unitary matrix Q and random upper triangular matrix R, and we multiply together to get matrix A, then we use all three factorizations to check the results. And here is our results:

```
The forward error for Q in CGS is 3.502910e-02
The forward error for R in CGS is 7.627150e-03.

The forward error for Q in MGS is 2.653670e-05
The forward error for R in MGS is 2.033376e-06.

The forward error for Q in HH is 5.863187e-06
The forward error for R in HH is 6.370900e-06.

The forward error for U in SVD is 2.000000e+00
The forward error for Sigma in SVD is 1.110223e-15
The forward error for V in SVD is 2.000000e+00.
```

According to the above result, we can see that all of those errors are not close to machine epsilon(10^{-16}) except the Σ matrix in SVD. But actually this cannot represent anything, since the singular value is the 'stretching' matrix in the geometric interpretation, so the error is small here means doesn't really means the algorithm is stable, since both U and V are unstable, but we can make a guess that SVD is actually a stable algorithm. So next, we need to check the backward stability, and we get the following results:

```
The backward error for CGS is 1.136787e-16.

The backward error for MGS is 1.294685e-16.

The backward error for HH is 8.832684e-16.

The backward error for SVD is 8.228905e-15.
```

That is out of our expectations, according to the results above, we have all the algorithms being backward stable. According to the theory, this should not be the case, so in order to verify that in detail, we decided to perform a least square problem and checking for the accuracy, and here is the result:

```
LSQ solution with degree 14 polynomial.
The problem is normalized so that the coefficient
computed below should be 1.

via classical Gram-Schmidt: c(15) = 0.00054494359517
                    error: 9.994551e-01
Unstable algorithm

via modified Gram-Schmidt: c(15) = 1.03237410404371
                    error: 3.237410e-02
Unstable algorithm
```

```
via Householder QR: c(15) = 1.00000011000427
                    error: 1.100043e-07
Stable algorithm

via SVD: c(15) = 1.00000011000643
                 error: 1.100064e-07
Stable algorithm.
```

Here the result is more clear, we have the CGS and MGS being unstable algorithm, HH and SVD being stable algorithm. But the question becomes: why we get the different result for both method(random matrix generation and least square problem)? The answer we get is: because it is very hard to say about stability for a random matrix, and the least square problem is (one of) the special case for verifying the stability, which means in general, the random matrix cannot represent all kinds of situations since there might be some special case that could possibly contains some results in some particular situations, and least square problem is one of the special case as well as one of the most popular case in real world applications.

3.2 Accuracy

In order to check for the accuracy of those algorithms, we can construct random matrices again. This time, since we are only caring about the solution since we are solving the equation $Ax = b$, so the random matrix being constructed could (possibly) represent all kinds of situations. As we discussed in the section 2.2, we are going to construct a matrix in $\mathbb{R}^{50}$ which all the entries are 1. By performing the various of algorithms, here is the result from matlab:

```
The norm between the actual x and CGS is 1.782275e-10.

The norm between the actual x and MGS is 3.292541e-10.

The norm between the actual x and HH is 2.815232e-12.

The norm between the actual x and SVD is 1.429226e-13.
```

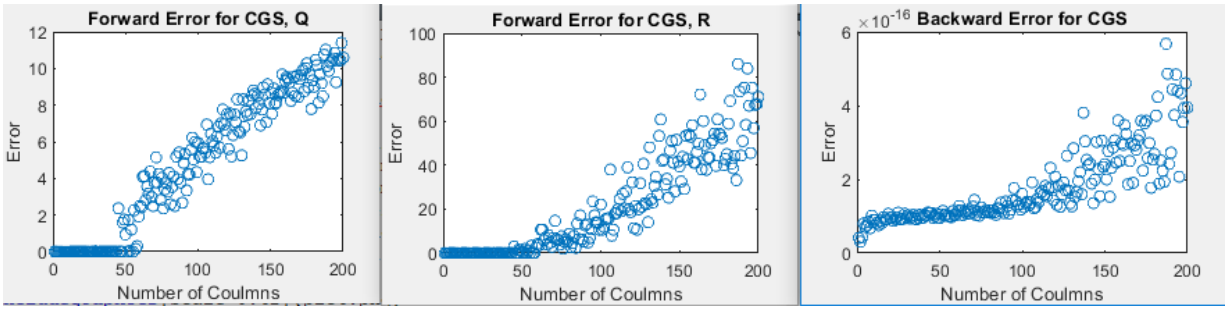
According to the above result, it is very easy to see that the HH and SVD are more accurate than CGS and MGS. For HH, we know that it has the least flop counts in all of three QR algorithm, so the fewer flops we did, the more accuracy we will get. However, for the SVD algorithm, it is not the case, according to google, we have the flop counts being $O(11n^3)$. However, consider about the psudoinverse (in real space), we have $A^+ = (A^TA)^{-1}A^T$, now, if we are doing the SVD for A, we will get the psudoinverse as the following:

$$\begin{aligned} A^+ &= (A^TA)^{-1}A^T \\ &= ((U\Sigma V^T)^TU\Sigma V^T)^{-1}(U\Sigma V^T)^T \\ &= (V\Sigma U^TU\Sigma V^T)^{-1}V\Sigma U^T \\ &= (V\Sigma\Sigma V^T)^{-1}V\Sigma U^T \\ &= V\Sigma^{-1}\Sigma^{-1}V^TV\Sigma U^T \\ &= V\Sigma^{-1}\Sigma^{-1}\Sigma U^T \\ &= V\Sigma^{-1}U^T \end{aligned}$$

notice that A^+ has the similar form for SVD decomposition as A(ie, V and U being unitary matrix and Σ^{-1} being singular value matrix), since in general, the psudoinverse provides the best(most accurate) solution to the linear system, then we have in this case, the SVD algorithm performs a more accurate solution to the linear system $Ax = b$. Therefore, our conclusion in this section will be the HH and SVD performs a more accurate solution to the linear system $Ax = b$ than CGS and MGS.

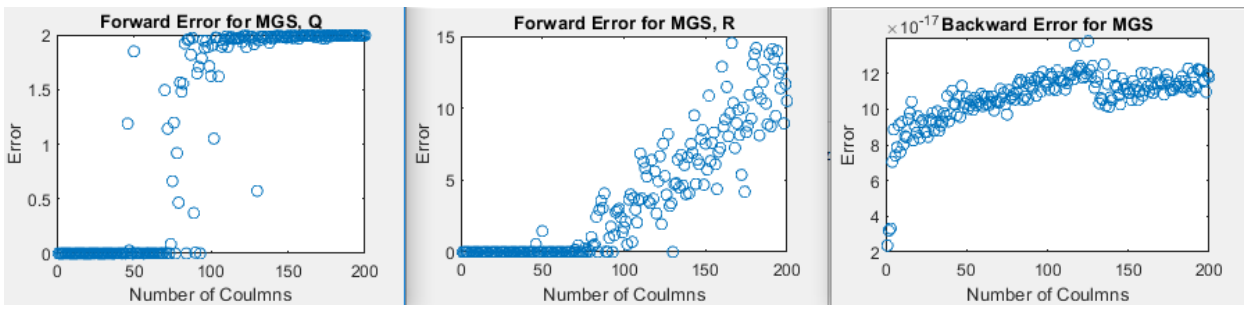
3.3 Forward and Backward Error

First of all, we will talk about how forward error and backward error grows when we variate the number of column. In order to perform this, we need to fix the number of rows first, and in this project, we will fix the number of rows being 200 and we variate the number of columns from 1 to 200, and here is the graph:

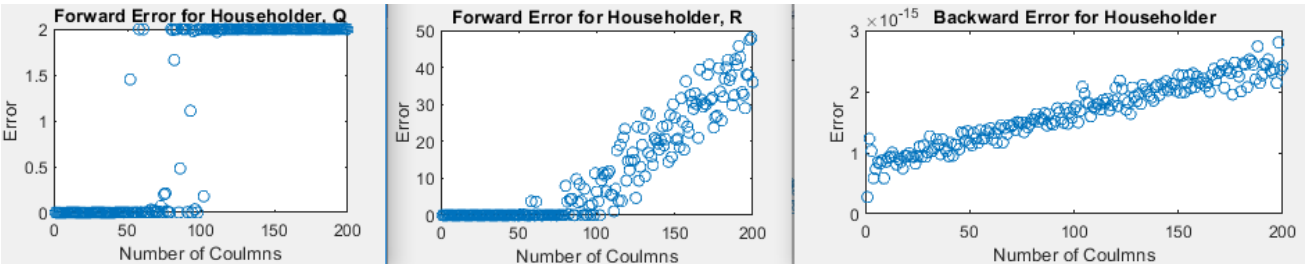


As we can see from the graph, the forward error keep increasing when we increase the number of columns. And the reason for causing that is since we are constructing the random matrix, and the forward error is bounded by $O(\kappa_A \epsilon_{machine})$, where κ_A is the condition number of the matrix A. In other wards, this phenomena happens because we are construction some random matrices that has a really large condition number which turns out that the forward error being so large. Actually this corresponding to our initial statement for stability, the error could be very small like machine epsilon(as we can see around 0), but it does not have to be. And according to this, we can conclude that the forward error will increase when we increase the number of columns.

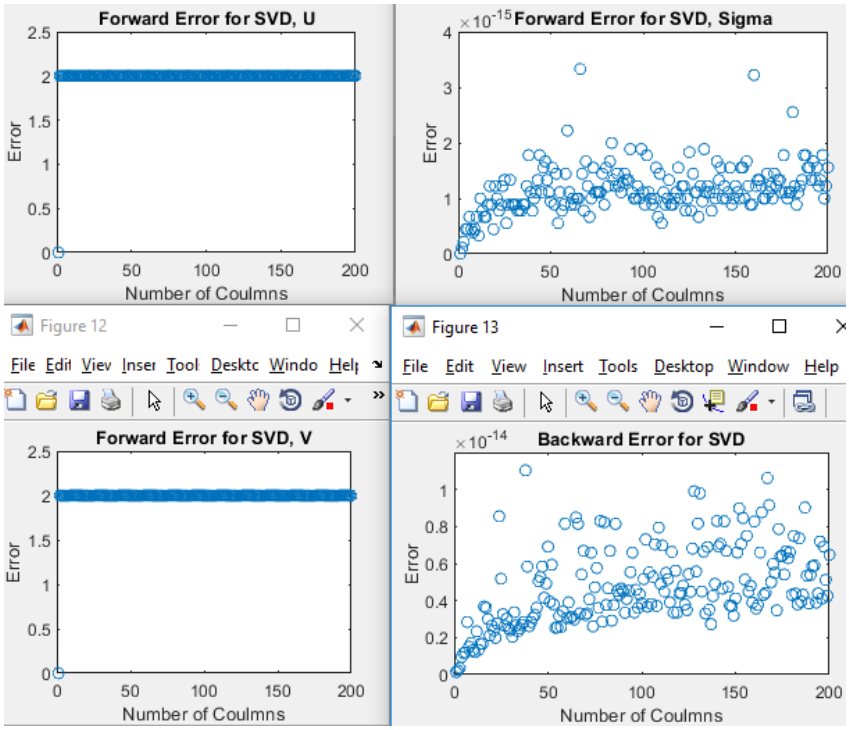
From the last graph above for backward error, we can see that the error is actually increasing again, but it turns out that all the errors are below 1.4×10^{-16}. However, this cannot say that CGS is stable since we can see that the increasing rate is not 0, which means that for some matrix being very large, the backward error will be increasing a lot (maybe greater than 1 at some point). Therefore, we can conclude that the CGS algorithm is unstable when we increasing the number of columns, which also corresponding to our result in stability analysis when we perform a least square problem.



So here we get the pattern for forward error for MGS. For the forward error of matrix Q, the pattern is more like a generalized linear model from 0 to 2 when we increasing the number of columns, and the forward error for R is like stable before approximately 100 and increasing linearly after that. Also, according to the last graph, we can see that the backward error is increasing but in a very slow increasing rate(looks like $\log(x)$), therefore, we can conclude that MGS could be used when performing the calculation with small dimensions, but might not performs well when we have a very large matrix, especially for matrices with large number of rows(ie, long flat matrix).

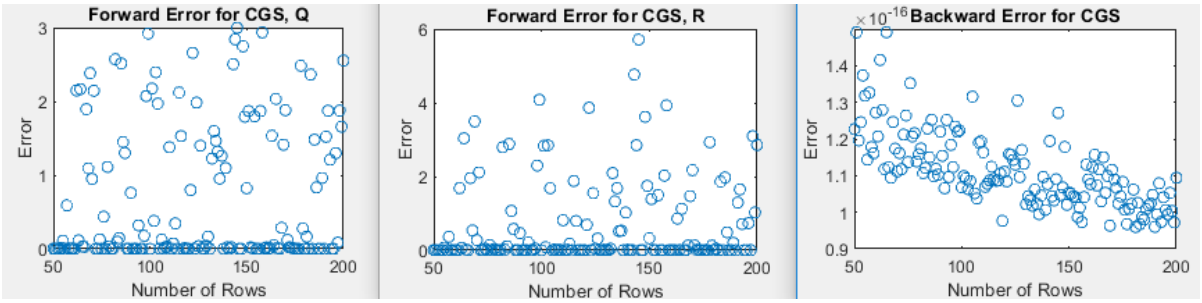


According to the above graph, we can see that for Q and R, the forward error looks the same as MGS, and for the backward error, we can see that it is still increasing, but in a very small increasing rate(almost flat, the ratio is like $\frac{1}{100}$), therefore, we can conclude that the error for HH algorithm is more backward stable than other QR methods.

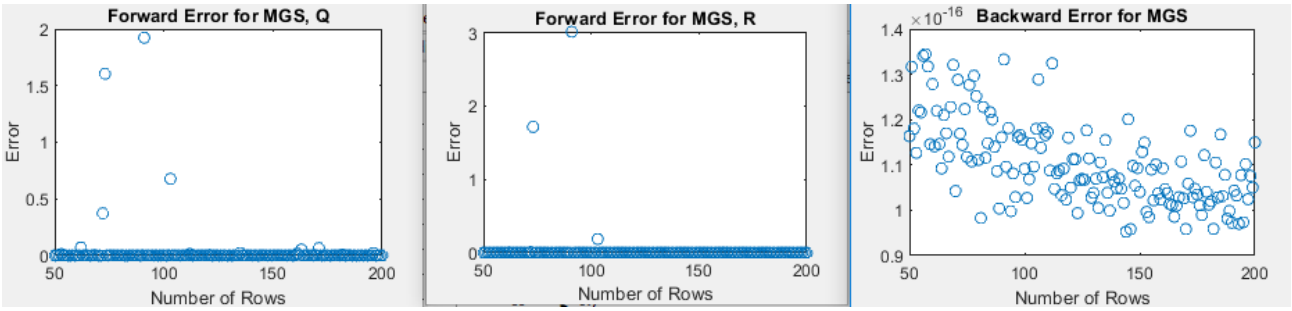


The last algorithm we need to mention is the SVD, we can actually observe a very interesting pattern for matrix U and V, the error is accumulated around 2, and for the matrix Σ, we can see that the error is around 10^{-15} which is close to machine epsilon. Also, for the backward error, we can actually see that the backward error is not changing too much between 0.2×10^{-14} to 0.8×10^{-14}. Therefore, we can conclude that since it is close to machine epsilon, and the error is not performing in a huge difference.

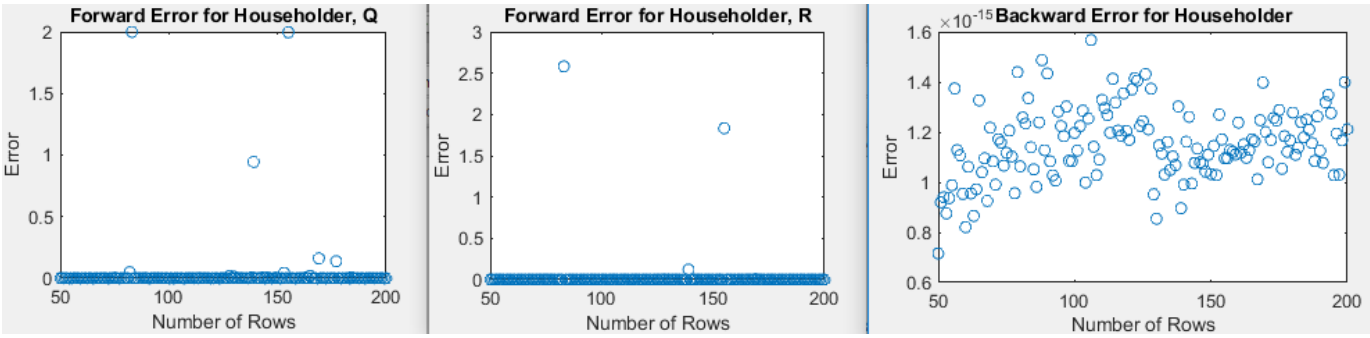
Next, we will talk about how forward error and backward error grows when we variate the number of rows. In order to perform this, we need to fix the number of columns first, we will fix the number of columns being 50 and we variate the number of columns from 50 to 200, and here is the graph for CGS:



As we can see, the error performs really different from the previous case, and there is actually no pattern for the forward errors. And for the backward error, we can see that the error is decreasing, and the possible reason might causing that is because the flop counts, according to what we learned from the lecture, the QR decompositions are being well performed when we have tall skinny matrix, and this is the case, the more differences we have between number of rows and columns, the less error we will get, and this is the same as MGS as shown below.

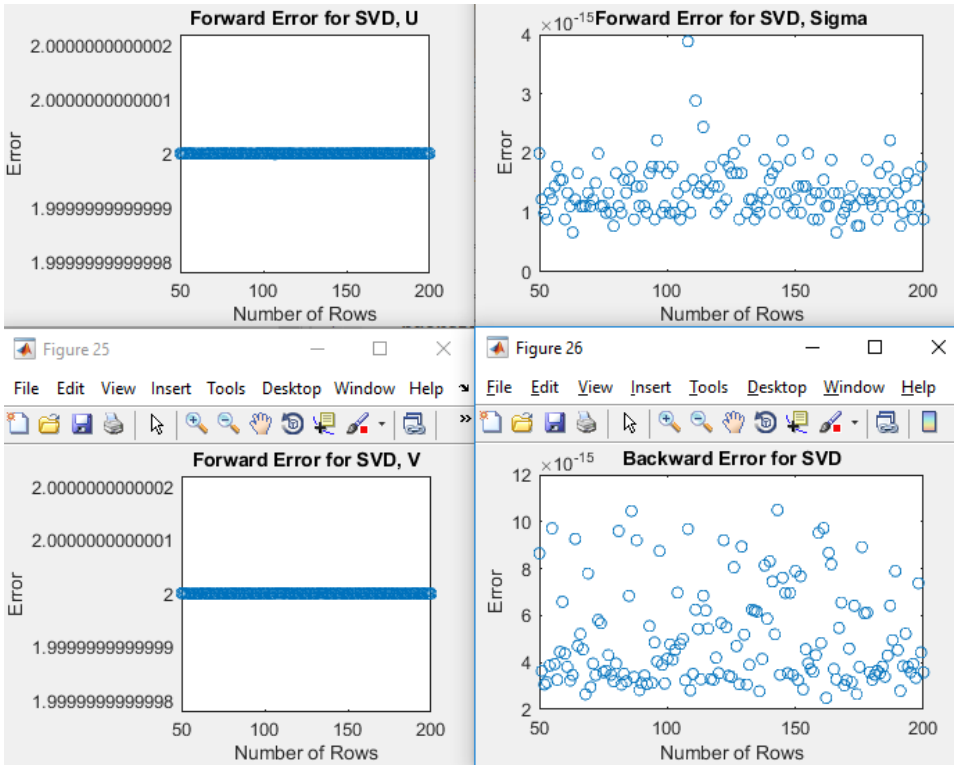


As we can see, the backward error performs in the same pattern as the CGS with the same reason mentioned above. And the forward errors are around 0 for both Q and R, although there are some error performs very large. But what we can say is the MGS method performs nicely for tall skinny matrix for both forward and backward error compare to CGS, therefore, MGS is a more stable algorithm than CGS.



According to the graph above for HH, actually we can still get the same pattern for forward errors for Q and R. But this time, the backward error looks better. We can see that all the errors are

mostly lie between the interval $[0.8 \times 10^{-15}, 1.4 \times 10^{-15}]$ and even for matrix with large number of rows, so at this point, since we finished looking at patterns for varies of QR decompositions, we can conclude that the QR decompositions are performing well when we have tall skinny matrix, which also corresponding to the theory since the flop count for QR decompositions are $O(2mn^2)$.



Finally, for the SVD-error shown above, we can see that the forward error for U and V is still around 2, and the forward error for Σ is around 10^{-15}, so we can claim that svd is a stable algorithm, and actually it is true if we see that the backward error for SVD is actually very small. And since this happens in all previous part, then we can get the conclusion that for SVD algorithms, the forward error of Σ does have something to do with the backward stability for the whole algorithm, which means that if the forward error for Σ is small, then the backward error for the algorithm is also small. But this happens only if we have (almost) the fixed forward error for U and V which are two unitary matrix, since the unitary matrix is invariant under the norms.

4 Summary

According to all the results we get above, there are several things that we need to notice and can be concluded:

- The random matrix generation cannot represent all cases when performing the stability.
- QR decompositions works well and spend shorter time for running on computer for tall skinny matrices.
- SVD algorithms is the 'best' algorithm in those four for stability and accuracy(never fails), but the disadvantage for SVD is the flop count$(2mn^2 + 11n^3)$. However, the flop counts also indicates that SVD performs even better for tall skinny matrices.

- More flop counts does not really means inaccurate.
- Stability and accuracy are not the same thing.

5 Reference

Trefethen, L. N., & Bau, D. I. (2000). *Numerical linear algebra*. Philadelphia: SIAM Society for Industrial and Applied Mathematics.

6 Appendix

GitHub Repository: `https://github.com/xxiissuunn/Math-577-Final-Project`