CENG 371 - Scientific Computing 2023-1 Homework 4

Anıl Eren Göçer
e2448397@ceng.metu.edu.tr

January 22, 2024

Question 1

All the codes are located in their corresponding .m files.

Question 2

a) By referring to the **Lemma 3** in Chapter 2.4 of Mahoney's notes, the purpose of scalings, $\frac{1}{\sqrt{cp_{i_t}}}$, is to make CR an unbiased estimator (element-wise) for AB.

That is, these scalings are used to satisfy $\mathbf{E}[(CR)_{ij}] = \mathbf{E}[(AB)_{ij}]$ and $\mathbf{Var}[(CR)_{ij}] = \mathbf{Var}[(AB)_{ij}]$ approximately. By making expected values (nearly) equal, we achieved to make CR an unbiased estimator for AB. Also, having the (nearly) equal variance enable us to make sure that CR is an estimator that is not only centered correctly on the average but also has a similar distribution to the actual values, AB.

After explaining the purpose of these scalings, I want to mention about the underlying idea of these scalings. Remember that we are dealing with randomized matrix multiplication, so the distribution from where probabilities plays a key role in the degree of closeness of CR to AB. The index i_t can be chosed with the probability of p_{i_t} and note that we are sampling with replacement (i.e. a column or a row can be chosen more than once). Because of this reason, contribution of a row or a column to expected value would be proportional to p_{i_t}. In order remove this effect, we scale the entries with $\frac{1}{\sqrt{cp_{i_t}}}$. We used square root because, it will be applied twice (once for a row and once for the corresponding column). Therefore, it yields $\frac{1}{cp_{i_t}}$.

b)

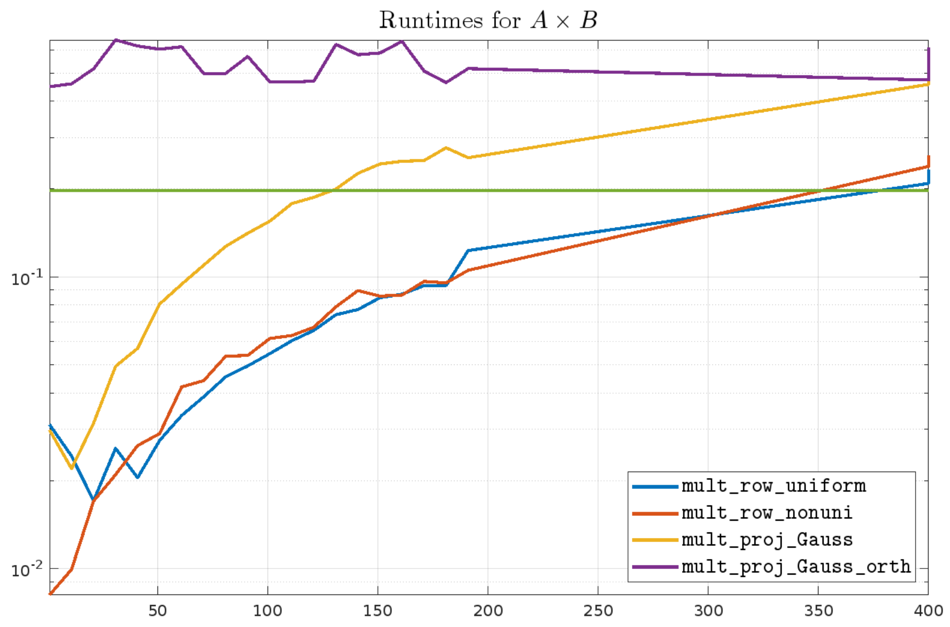

Figure 1: runtimes_AB.png

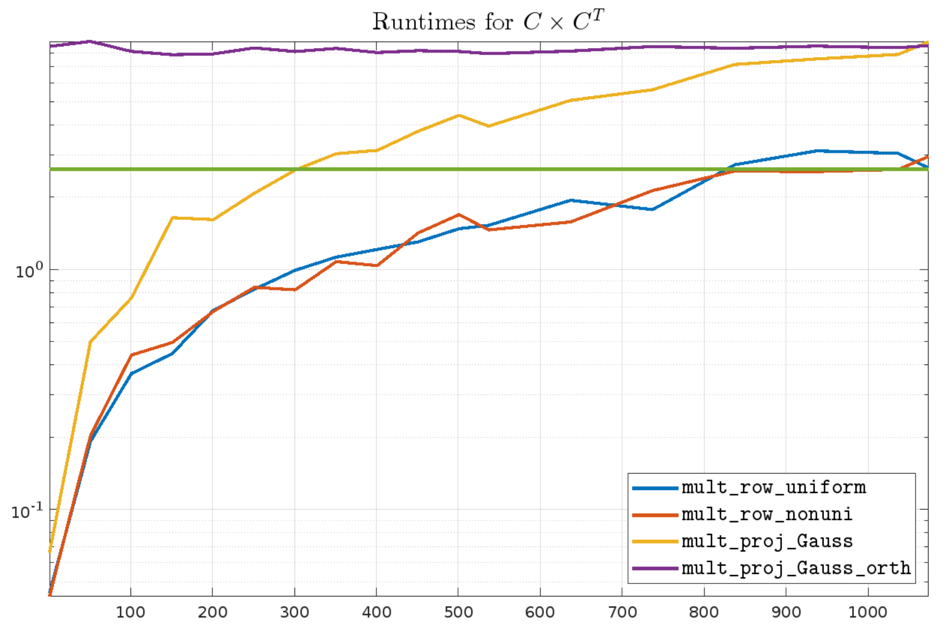

Figure 2: runtimes_C.png

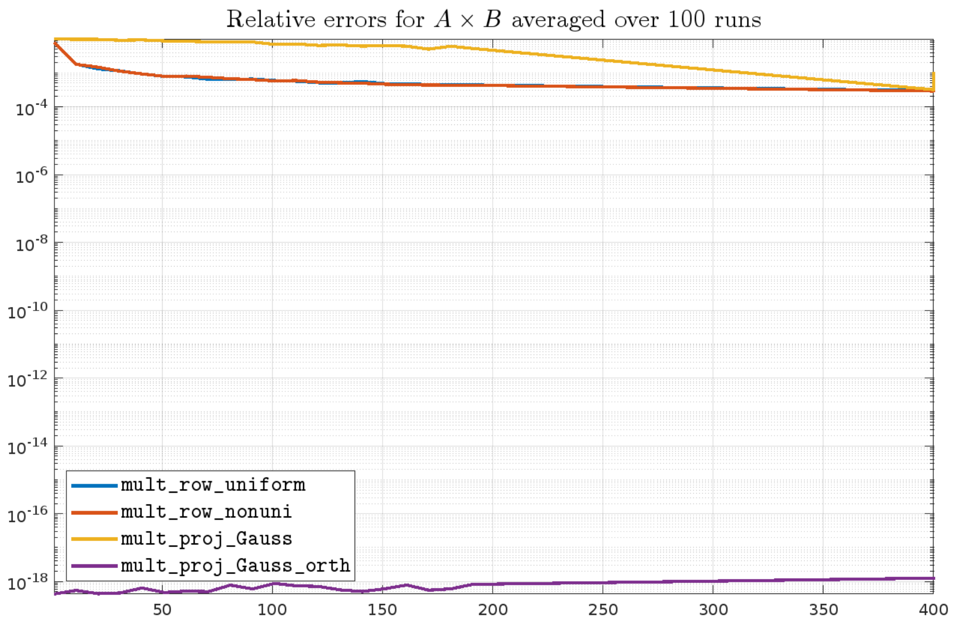

Figure 3: res_AB.png

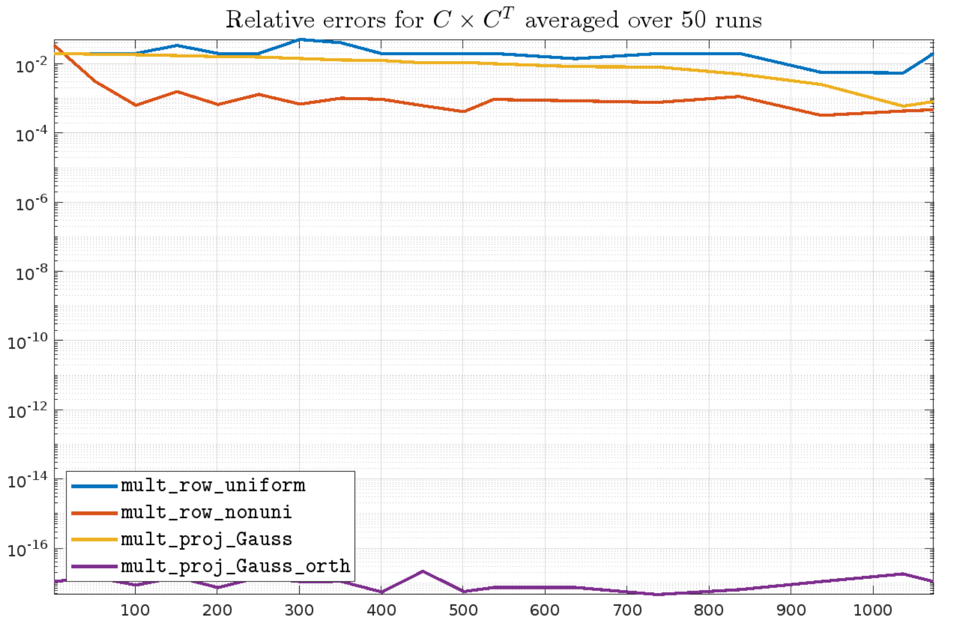

Figure 4: res_C.png

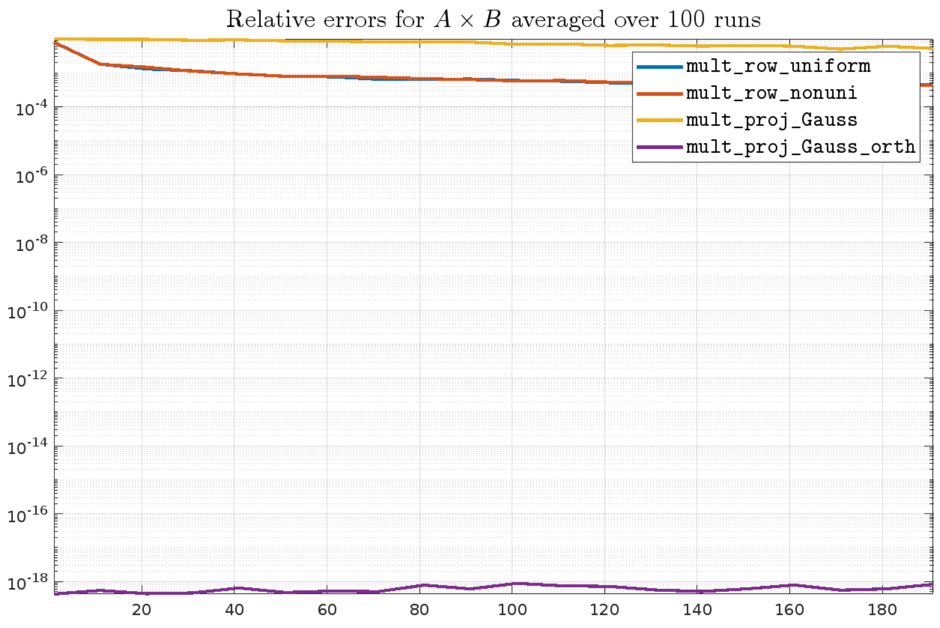

Figure 5: res_AB_dropped.png

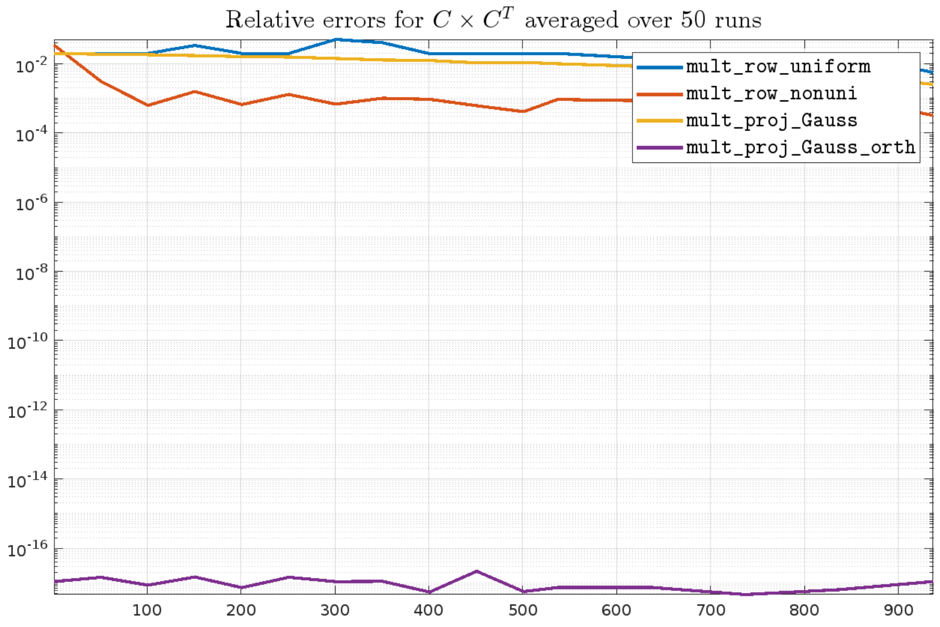

Figure 6: res_C_dropped.png

c)

Runtime Comparison:

As it can be seen clearly from the Figure 1 and 2, **mult_proj_Gauss_orth** is the slowest one because it includes QR factorization (for Gram-Schmidt process) and more multiplication operations. Similarly, **mult_proj_Gauss_orth** is slower compared to **mult_row_uniform** and **mult_row_nonuni**, but it is faster compared to **mult_proj_Gauss_orth** because it does not includes QR factorization which is done for orthogonalization. As expected, **mult_row_uniform** and **mult_row_nonuni** are the fastest ones and they do not include any costly operations such as QR factorization, multiple matrix multiplication for projection etc. . Moreover, **mult_row_uniform** and **mult_row_nonuni** behaved quite similarly since only the difference between them is the fact that they sample from the different distributions which does not have any effect on the runtime. In addition to these, results were quite close for both $A \times B$ and $C \times C^T$. In addition, except the **mult_row_nonuni**, all the algorithms' runtimes increase as the constant **c** gets larger because bigger **c** means more calculations.

Relative Error Comparison:

As it can be seen from the Figures 3-4 and 5-6, **mult_proj_Gauss_orth** is by far the best one in terms of the relative error. For both cases ($A \times B$ and $C \times C^T$), its average relative error is about 10^{-17} and this can be accepted as the perfect approximation in most of applications. It is clear that other methods have more or less the same performance, which is around 10^{-3} in terms of errors. But one thing to notice is that for all of the methods, they have a more smooth line for $A \times B$ although they have higher variance for $C \times C^T$.

Use-case Comparison:

For the application in which you need to take important actions based on the result of a computation (nearly perfect results) but you can tolerate long-running times, it is better to use **mult_proj_Gauss_orth**. Non-time-critical control systems are example of such applications. For the time-critical tasks that some amount of relatively small error can be tolerated, it is better to use **mult_row_uniform** or **mult_row_nonuni**.

d) **mult_row_uniform** performs worse while calculating $C \times C^T$ in comparison to $A \times B$ and this is related to the relative importance of rows and columns in C and C^T. By saying "importance", I mean some of the rows and columns may encode more information. Remember the gray-scale images in the HW3, pixels located at the corners was not including much information. Therefore, they do not much affect the quality in case of low-rank approximations. This example illustrates importance notion in this context. Considering this question, let's assume we sample a row from A and a column from B. It is likely that row is important but column is unimportant or column is important but row is unimporant. Because of this reason, they balance each other even if they are sampled uniformly. However, when we are sampling a row and column from C and C^T, sampled row and the column are both the same (actually, one is the transpose of the other). Therefore, if the row is very important then the column is very important too. Similarly, if the row is very unimportant then the column is very important too. If we use uniform sampling, this effect gets larger. That is, if they are unimportant, the result would be even less important. Similarly, if they are important, the result would be even much important. Therefore, $C \times C^T$ deviates too much from the correct values.