Reaching the Quality of SVD for Low-Rank Compression Through QR Variants

Esragul Korkmaz, Mathieu Faverge, Grégoire Pichon, Pierre Ramet

Reaching the Quality of SVD for Low-Rank Compression Through QR Variants

Esragul Korkmaz, Mathieu Faverge, Grégoire Pichon, Pierre Ramet

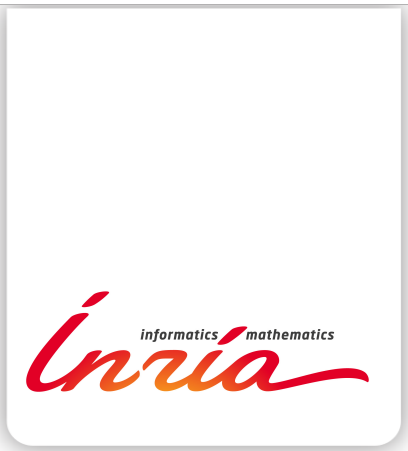
RESEARCH REPORT

N° 9476

September 2022

Project-Team HIEPACS

ISSN 0249-6399 ISRN INRIA/RR--9476--FR+ENG



Reaching the Quality of SVD for Low-Rank Compression Through QR Variants

Esragul Korkmaz[*†‡§], Mathieu Faverge[†‡*§], Grégoire Pichon[¶], Pierre Ramet [§†‡*]

Project-Team HIEPACS

Research Report n° 9476 — September 2022 — 44 pages

Abstract: Solving linear equations of type $Ax = b$ for large sparse systems frequently emerges in science/engineering applications, which is the main bottleneck. In spite that the direct methods are costly in time and memory consumption, they are still the most robust way to solve these systems. Nowadays, increasing the amount of computational units for the supercomputers became trendy, while the memory available per core is reduced. Thus, when solving these linear equations, memory reduction becomes as important as time reduction. For this purpose, compression methods are introduced within sparse solvers to reduce both the memory and time consumption. In this respect, Singular Value Decomposition (SVD) is used to reach the smallest possible rank, but it is too costly in practice. Rank revealing QR decomposition variants are used as faster alternatives, which can introduce larger ranks. Among these variants, column pivoting or matrix rotation can be applied on the matrix A, such that the most important information in the matrix is gathered to the leftmost columns and the remaining unnecessary information can be omitted. For reducing the communication cost of the QR decomposition with column pivoting, blocking versions with randomization are suggested as an alternative to find the pivots. In these randomized variants, the matrix A is projected on a lower dimensional matrix by using an i.i.d. Gaussian matrix so that the pivoting/rotational matrix can be computed on the lower dimensional matrix. In addition, to avoid unnecessary updates of the trailing matrix at each iteration, a truncated randomized method is suggested to be more efficient for larger matrix sizes. Thanks to these methods, closer results to SVD are obtained with reduced compression cost. In this report, we compare all these methods in terms of complexity, numerical stability, obtained rank, performance and accuracy.

Key-words: Low-rank compression, randomization

[*] Inria Bordeaux - Sud-Ouest, Talence, France
[†] Bordeaux INP, Talence, France
[‡] CNRS (Labri UMR 5800), Talence, France
[§] University of Bordeaux, Talence, France
[¶] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP

RESEARCH CENTRE BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Atteindre la qualité d'une SVD avec une compression de rang faible pour différentes variantes de la factorisation QR

Résumé : La résolution d'équations linéaires de type $Ax = b$ pour de grands systèmes creux apparaît fréquemment dans les applications scientifiques et constitue le principal goulot d'étranglement. Bien que les méthodes directes soient coûteuses en temps et en mémoire, elles restent la méthode la plus robuste pour résoudre certains de ces systèmes. De nos jours, on note une augmentation du nombre d'unités de calcul pour les superordinateurs alors que la mémoire disponible par cœur est réduite. Par conséquent, lors de la résolution de ces systèmes linéaires, la réduction de la mémoire devient aussi importante que la réduction du temps. À cette fin, des méthodes de compression pour les blocs denses, apparaissant lors de la factorisation des matrices creuses, ont été introduites pour réduire la consommation de mémoire, ainsi que le temps de résolution. En recherchant le rang de compression le plus faible possible, la décomposition en valeurs singulières (SVD) donne le résultat optimal. Elle est cependant trop coûteuse et nécessite une factorisation complète pour trouver le rang résultant. Les variantes de la décomposition QR sont moins coûteuses, mais peuvent conduire à des rangs plus importants. Parmi ces variantes, le pivotage des colonnes ou l'application d'une rotation peuvent être appliqués à la matrice A, de telle sorte que les informations les plus importantes de la matrice soient rassemblées dans les premières colonnes et de permettre de négliger le reste de la sous-matrice. Pour réduire le coût de communication de la décomposition QR classique avec pivotage des colonnes, des versions par bloc et utilisant des techniques de randomisation sont proposées comme solution alternative pour trouver les pivots. Dans ces variantes randomisées, la matrice A est projetée sur une matrice de dimension beaucoup plus faible en utilisant une matrice gaussienne dont chaque variable aléatoire a la même distribution de probabilité que les autres et toutes sont mutuellement indépendantes. Ainsi, la matrice de pivotage/rotation peut être calculée en dimensions réduites. En outre, pour éviter les mises à jour inutiles sur la sous-matrice à chaque itération, une méthode randomisée tronquée est proposée et s'avère plus efficace pour les matrices de grande taille. Grâce à ces méthodes, des résultats proches de SVD peuvent être obtenus et le coût de la compression peut être réduit. Dans ce rapport, nous comparerons toutes ces méthodes en termes de complexité, stabilité numérique, rang obtenu, performance et précision.

Mots-clés : Compression de rang faible, randomisation

Contents

1 Introduction

Many fields like data mining, signal processing and computer vision need to handle large matrices. As the storage and matrix operations are too expensive for these matrices, some approximation (compression) techniques are proposed to reduce the cost. These techniques aim to represent the original matrix, A, in the form

$$A_{m\times n} \approx U_{m\times r} V^T_{r\times n} \tag{1}$$

such that

$$||A_{m\times n} - U_{m\times r} V^T_{r\times n}|| < \epsilon ||A||. \tag{2}$$

Here, the objective is to exhibit an approximation at a required precision (ϵ), where r is much smaller than $min(m, n)$. Unfortunately, determining the most suitable compression technique for arbitrary problems is difficult. Therefore, each application needs to come up with a convenient method according to the requirements like memory, time, stability and accuracy.

In this report, we aim to provide a descent literature view, although not complete, on the existing compression kernels. Then, we compare some stable kernels to find the fastest kernel which can obtain small enough r values at a given precision. We work on the domain of sparse low-rank direct solvers, where the dense parts of the sparse matrix are clustered into blocks of similar sizes. In this clustering, we compress each admissible block independently in the form (1) to improve the memory and time requirements of the solver. Therefore, our experiments cover only the blocks sizes we are interested in our PaStiX solver [20], i.e. $m, n < 2000$. As we do not use large enough block sizes to take advantage of parallel environments, we provide our experiments in a sequential environment.

In Section 2, we provide all the necessary background on the compression kernels. In this section, the motivation of different kernels, especially the ones on which we will focus, is also explained. The literature review specific to the four compression kernels that we concentrate on is provided in Section 3. In Section 4, we define some notations to be used in the algorithms of this report. Then, in Section 5, four different compression methods are studied in detail. In Section 6, their complexities are shortly summarized to show the big picture. In Section 7, the numerical results of some generated matrices are observed in terms of stability, performance, compression ranks and accuracy, whereas in Section 8, the real-life case matrix results are discussed. Finally, in Section 9, the conclusion of this report is provided.

2 Background

In order to understand the compression kernels, we first need to provide the meaning of r in the approximation (1). Mathematically, the rank of a matrix

(r) stands for the number of linearly independent columns/rows of the matrix. Then, if we represent a matrix as multiplication of two matrices as

$$A_{m\times n} = U_{m\times r} V^T_{r\times n}, \tag{3}$$

r can be determined through the singular value decomposition (SVD), which has the form

$$A_{m\times n} = \hat{U}_{m\times m} \Sigma_{m\times n} \hat{V}_{n\times n}. \tag{4}$$

Here, $\hat{U}$ and $\hat{V}$ are orthogonal matrices, while Σ is a diagonal matrix with the singular values, σ_j, on the diagonal with $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ and $\sigma_{r+1} = \sigma_{r+2} = ... = \sigma_n = 0$. Here, the rank is equal to the number of non-zero singular values in Σ.

When we work in finite precision, we are usually interested in only the large values and we omit the small values depending on our precision requirement. In this case, an approximated rank is determined by using either a fixed-rank or a fixed-precision based criterion to satisfy the condition (2), where r is the rank of the approximation and ϵ stands for the precision. Observe that this rank is numerically computed and it can have different values for different precisions. From now on, r will only represent approximation ranks in this report. Determining a convenient fixed-rank to ensure a targeted accuracy is not trivial in practice. Thus, the fixed-precision based criterion provides better solutions for abstract problems. We will adopt a fixed-precision based criterion in our work.

If the equation (4) is written in the block form

$$A = \begin{bmatrix} \hat{U}_1 & \hat{U}_2 \end{bmatrix} \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} \end{bmatrix} \begin{bmatrix} \hat{V}_1 \\ \hat{V}_2 \end{bmatrix},$$

the approximation (1) can be written as

$$A \approx UV^T = \hat{U}_1 \Sigma_{11} \hat{V}_1, \tag{5}$$

where $U = \hat{U}_1$, $V^T = \Sigma_{11}\hat{V}_1$ and Σ_{11} is of size $r \times r$.

In this report, we will use either

$$\frac{||\Sigma_{22}||_2}{||A||_F} = \frac{\sigma_{r+1}}{||A||_F} < \epsilon \tag{6}$$

or

$$\frac{||\hat{U}_2 \Sigma_{22} \hat{V}_{22}^T||_F}{||A||_F} = \frac{||\Sigma_{22}||_F}{||A||_F} = \frac{\sqrt{\sum_{i=r+1}^{n} \sigma_i^2}}{||A||_F} < \epsilon \tag{7}$$

to numerically find the approximation rank r through SVD.

The SVD method is not the only way to compress the matrices in the form (1). Depending on the application requirements, different compression methods can be adopted. For example, for large sparse matrices, some methods

that maintain the sparse structure can be used [3]. These methods are out of our scope. In our work we specifically focus on compressing the dense blocks with the block sizes that can arise in the block low-rank representation of sparse matrices in the PaStiX solver [20]. Additionally, we are interested in compressing in a sequential environment since these blocks are not large enough to take advantage of parallel environments. As the precision based numerical stopping criterion (presented in Section 2.1) in our work ensures the precision of the compression, our main aim is to choose a stable and fast kernel for our solver, which can keep the representative data with a rank as small as possible.

At a given fixed numerical rank, the Singular Value Decomposition (SVD) reaches to the optimal error, through both the spectral and Frobenius norms [10]. However, the cost of SVD is too high for large matrices, as it requires all the decomposition to complete to determine the rank. Therefore, some alternative methods are proposed to obtain a rank as close as possible to the one obtained using SVD, while being much faster at a given precision. These methods include but are not limited to the adaptive cross approximation (ACA) [2], column pivoted QR (QRCP) [9], rank revealing QR (RRQR) [5], interpolative decomposition [13, 17], UTV [15], and randomized SVD [12].

The ACA method is proposed as a very efficient method for smooth kernel matrix approximations. However, since the correct construction of the low-rank basis is not ensured, it might fail to converge [4]. Therefore, despite its low cost, we prefer more stable methods.

Some methods like interpolative decomposition and UTV are powerful in terms of the approximation quality. Nonetheless, the QR factorization with column pivoting (QRCP) methods can produce ranks close to the one of SVD, while being much faster. For example, in [15], the authors show that the randomized UTV method they propose is competitive to QRCP, as they have the same asymptotical complexity. However, it can run faster than QRCP only in a parallel environment with matrices larger than the ones we are interested in. In a sequential environment with matrix sizes of our interest (dimensions smaller than 2000), QRCP can run more than 2× faster than UTV. Therefore, QRCP methods are commonly adopted in our context.

The disadvantage of the QRCP methods is that some ill-conditioned matrices, like Kahan matrix, can reach larger ranks than expected [11]. However, it is worth noting that this problem occurs very rarely in real life. Nevertheless, some rank revealing QR (RRQR) methods are proposed, with an additional cost, to avoid this issue. From now on, we focus on some promising QR methods in our context.

2.1 QR based compression kernels as an alternative to SVD

In this section, we briefly present the QR based compression methods before providing more specific details on the ones that we will study in this report. A QR decomposition has the form

$$A = QR,$$

where Q is orthonormal and R is an upper trapezoidal matrix (non-square matrix, where all elements under the diagonals are zero). Opposite to SVD, this decomposition, by its nature, does not provide a rank revealing feature. For this purpose, an orthogonal matrix Q_G (i.e. $Q_G Q_G^T = Q_G^T Q_G = I$, where I is the identity matrix) should be applied to the original matrix A. In this way, we gather the representative data on the left side to omit the remaining unnecessary ones on the right. Then, considering the block form of the QR compression method as

$$AQ_G = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \approx Q_1 \begin{bmatrix} R_{11} & R_{12} \end{bmatrix}, \tag{8}$$

where R_{11} is of dimension $r \times r$, we obtain the low-rank form (1) with $U = Q_1$ and

$$V^T = [R_{11} R_{12}] Q_G^T.$$

Remember that SVD method is the optimal compression kernel. Therefore, as the QR based compression kernels aim to produce an approximation as close as possible to SVD, the desired approximation criterion of the QR methods can be based on the Frobenius norm of the trailing matrix as

$$||Q_2 R_{22}|| = ||R_{22}|| \approx ||\Sigma_{22}||, \tag{9}$$

where $||Q_2 R_{22}||$ is the trailing matrix norm of the block QR representation in the equation (8). Then, similar to the criterion (7), the compression rank of the QR methods can be determined numerically through the criterion

$$\frac{||R_{22}||}{||A||} < \epsilon.$$

Note that, opposite to SVD, whenever the stopping criterion is satisfied (if we can evaluate it), the QR process terminates and Q_2 and R_{22} are not computed as they are not needed. That is why, QR compression kernels are cheap alternatives to SVD when they are used to satisfy 9.

There are different ways to compute the QR decomposition: Gram-Schmidt process, Householder transformation or Givens rotations method [11, 23]. As the Householder transformations are more stable than the Gram-Schmidt and cheaper than Givens rotations, we adopt the Householder based QR approaches in our work.

Note that in this report we focus on panel-wise QR factorizations (instead of factorizing the matrix index by index) for exploiting the level-3 BLAS [7] operations to be more efficient on top of modern architectures. The panel-wise QR is illustrated in Figure 1 on the left, where b stands for block size. Here, the current panels of b columns and rows, respectively, are in yellow and the trailing matrix is in green. Then, during the factorization, the trailing matrix is updated

at the end of each panel iteration through the panel contributions at once, instead of at each column contribution separately. Although this panel-wise update operation improves the performance, the column pivoted version of the QR factorization requires an additional storage to be able to update the column norms, which is necessary to determine the next column pivot [22]. Now let us focus on more specific background on the compression kernels that we will compare in this report.

2.2 Different features of QR based compression kernels

In this section, we briefly discuss the differences of the QR based compression kernels that we will study in the remainder of the report. The first difference is the way of gathering the representative data to the left part of the matrix. The second one is whether the randomized sampling is used or not, while the third one is based on when the trailing matrix updates are performed. Let us respectively explain these three differences in three sub-sections.

Gathering the data to the left of the matrix

The first difference of the compression kernels in the following sections is the choice of the Q_G matrix in the approximation (8). This choice affects how good we can approximate the SVD method. It can simply be chosen as a column permutation matrix to gather the representative data only to the left part of the matrix. For example, the column with maximum 2-norm can be chosen as the pivot at each iteration. Alternatively, the Q_G matrix can be selected as a convenient rotation matrix to have more control over the quality of the approximation.

No matter whether a permutation or a general rotation matrix is chosen, the computation of the Q_G matrix is inefficient and it is a bottleneck for large matrices. Therefore, in the next section we explain a technique to reduce this cost.

Randomization technique

The second difference of our kernels, randomized sampling, aims to reduce the cost of generating the Q_G matrix. This technique projects the original large matrix $A_{m\times n}$ on a lower dimensional representative matrix $B_{d\times n}$, where $d \ll m$. Through this projection, the cost of memory-bound operations can be reduced.

As an example, we can observe an original matrix and its representative (sample) matrix in a column pivoted QR factorization in Figure 1. Here, j stands for an arbitrary index during the factorization and b is the block size. Green color represents the sub-matrices which will be used in the next panel iterations, while the yellow color stands for the panels which are completed in the current iteration. Observe that the row dimension of the sample matrix is chosen as $b+p$. Here, the oversampling size, p, stands for a small fixed number, which is used to improve the quality of the projection. Then, thanks to this

projection, we can efficiently compute the panel pivots through the matrix B instead of computing the column pivots through the original matrix. As a result, we can improve the performance for large matrices since column pivoting is a bottleneck in the QR method.

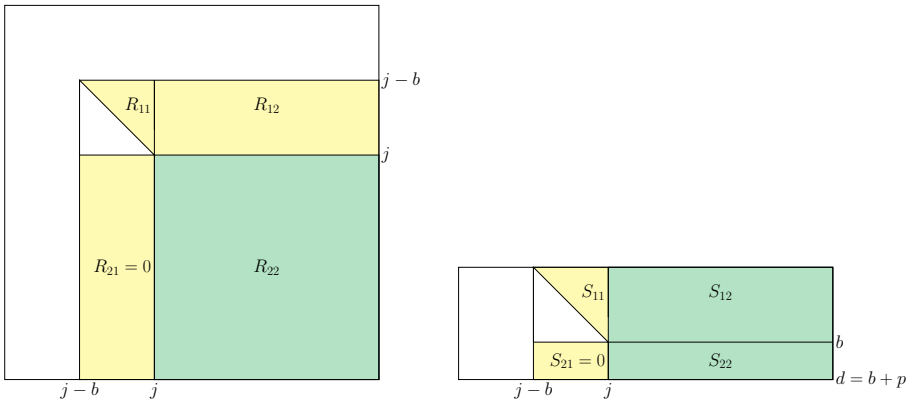


Figure 1: Panel-wise QR with column pivoting on the original matrix $A_{m\times n}$ (on the left) and on the sample matrix $B_{(b+p)\times n}$ (on the right). Here, $b + p \ll m$, where b stands for the block size and p is the oversampling size which is used to obtain more representative data on the sample matrix. The yellow panels represent the parts which are completed after the current panel iteration. The green parts illustrate the trailing matrices which will be used in the next panel iterations. The upper triangular yellow matrices stand for the factorized partial R matrices.

Left-looking and right-looking approaches

The last difference of our methods is whether the decomposition is right-looking or left-looking. This feature is based on when the trailing matrix is updated. Let us explain it through Figure 2.

On the left of Figure 2, the left-looking approach is presented. Here, j stands for the current index. In this method, the already computed yellow panel contributions are used to update the current green panel. Here, the current panel is updated just before being factorized. In this way, the unnecessary full trailing matrix updates at each panel iteration are avoided. However, in the left-looking approach an additional storage is required compared to the right-looking version. That is, in the classical QRCP algorithm, the column norms should be downdated at each column iteration to be able to determine the next pivot column. When the trailing matrix update is postponed during the QRCP factorization, this downdate operation causes a problem. As explained in [22], although the block-wise update operation requires the earlier row values of A before the block-wise column reductions, these rows should have been already updated for downdating the column norms. Therefore, this problem

can be solved at the expense of an extra storage for the previous reflector accumulations.

On the right of Figure 2, the right-looking approach is illustrated. Here, at the end of each panel iteration, all the trailing matrix (green part) is updated through the contributions from the current panel (yellow parts) in the figure. This feature is costly, especially for large matrices with small ranks, as the trailing matrix after the approximation rank is unused. However, since the panels are small and the updated parts are large for these matrices, this feature is promising for parallel environments.

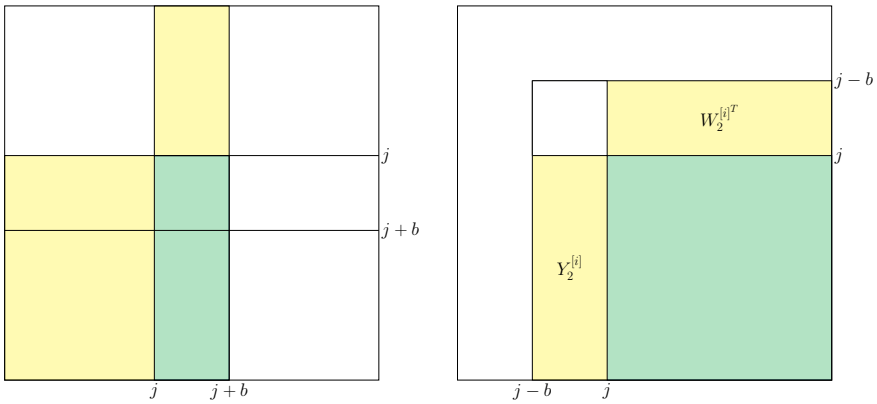


Figure 2: Inter-panel left-looking and inter-panel right-looking approaches. The former is illustrated on the left, while the latter is on the right. The contributions of the yellow color parts are used for updating the green parts. Right-looking approach updates all the trailing matrix at the end of each panel iteration, whereas the left-looking one only updates the current panel at the beginning of each panel iteration.

2.3 Specifications

In this section, we provide necessary specifications to understand the following sections better. For the sake of simplicity, we only study the matrices with real values, where the superscript T represents their transpose. However, all formulas can be easily adapted to complex matrices. In the following sections, the computational complexities of the methods stand for the square matrices of dimension $n \times n$.

In the algorithms, we use indices starting from 0. At the end of the algorithms, R is stored in-place at the upper triangular part of A. Therefore, by applying the inverse permutation or rotation, depending on the method used, we obtain the V^T matrix. The $U = Q$ matrix can be obtained by using both the reflectors, stored in-place in the lower part of the matrix, and their scalar factors vector. This can be performed easily through the $xUNGQR$ routine of LAPACK [1] for example.

The stopping criterion (6) allows to reach smaller ranks with SVD compared to the criterion (7). However, we use the Frobenius norm of the trailing matrix when applying our stopping criterion within the QR methods. Therefore, for being consistent with the QR methods of this report, we will mostly focus on the latter criterion (7) with Frobenius norm.

3 Related work

In this section, we explain the related work on the four QR methods that we study in this report, which differ from each other through the features discussed in Section 2.2.

In [9], the authors propose a panel-wise randomized QR with column pivoting (RQRCP) method with a new sample matrix update formula, where they adopt a fixed-rank based stopping criterion. The update formula in this work improves the randomization method cost by eliminating the resampling at each panel iteration. Moreover, they also offer a truncated randomized QR with the column pivoting (TQRCP) method to avoid the unnecessary trailing matrix updates. Through these methods, they also come up with an efficient truncated SVD algorithm. In this report, we will study the RQRCP and TQRCP methods in [9] respectively in Sections 5.2 and 5.3.

In [16], the authors independently conduct a similar work to [9] on the RQRCP method that is based on Householder reflections, with a fixed rank criterion. Here, they also contribute to the RQRCP method with a sample matrix update, instead of resampling, which is mathematically equivalent to [9].

In [24], both [9] and [16] RQRCP methods are investigated with a fixed rank based criterion. The authors analyze the efficiency of the sample matrix updates from these two papers, as well as analyzing the reliability of the randomized QR with column pivoting. In addition, they propose a spectrum revealing QR method for the rank revealing problem of the ill-conditioned matrices, thanks to some extra interchanges.

In [14], the author proposes column pivoted and rotational rank revealing QR algorithms, as well as a power method for a better range convergence, with a fixed rank criterion. Depending on the convergence to the singular values, the randomized rotational QR (RQRRT) method can give closer results to SVD by collecting most of the data mass on the diagonals. Although the RQRRT method is more expensive than the QR with column pivoting methods, we will include this method in our study, in Section 5.4, to see if a good trade-off between performance and approximation quality can be reached.

In [18], the authors propose numerically determined rank procedures. Here, the algorithms are more general randomized methods and can be adopted as partial SVD, CUR, interpolative decomposition or RRQR. The offered rank revealing QR can be seen as a modified version of column pivoted Gram-Schmidt. This method is using the numerical stopping criterion similar to us. In this work, they also propose a power method to improve the accuracy when handling slowly decreasing singular values.

In [12], the authors conduct a detailed study of different randomized methods. Here, also a randomized numerical rank decision criterion is proposed.

Note that all the methods we explain in this report are existing methods. However, we aim to implement them in a similar fashion for a better experimental comparison. As mentioned before, we determine the compression ranks in a numerical way through our stopping criterion to exploit low-rank features, without any underlying problem knowledge requirement. This criterion is first time applied to the TQRCP and RQRRT methods in the literature. Now let us provide some necessary notations that will be used in the following sections.

4 Notations

For the sake of simplicity, we use some notations for avoiding detailed index specifications inside the algorithms of this report:

- $A^{[i]}$ represents the sub-matrix part that we use in the i^{th} panel iteration. For example, considering we already factorized $2b$ columns of A, $A^{[2]}$ shows $A_{2b:m;2b:n}$. The notation can be trivially converted to the vectors.

- At the i^{th} panel iteration, $A^{[i]}$ is represented in the block form

$$A^{[i]} = \begin{bmatrix} A_{11}^{[i]} & A_{12}^{[i]} \\ A_{21}^{[i]} & A_{22}^{[i]} \end{bmatrix},$$

where $A_{22}^{[i]}$ stands for the trailing matrix and it is used as $A^{[i+1]}$. We illustrate the matrix and vector (N) representations at the i^{th} panel iteration in Figure 3.

- $A^{(j)}$ represents the sub-matrix part that we use in the j^{th} column iteration. For example, considering we already factorized $2b+3$ columns of A, $A^{(2b+3)}$ represents the matrix $A_{(2b+3):m;(2b+3):n}$. The notation can be trivially converted to the vectors.

- At the j^{th} column iteration, $A^{(j)}$ is represented in the block form

$$A^{(j)} = \begin{bmatrix} a_{11}^{(j)} & A_{12}^{(j)} \\ A_{21}^{(j)} & A_{22}^{(j)} \end{bmatrix},$$

where $A_{22}^{(j)}$ stands for the trailing matrix and it is used as $A^{(j+1)}$. We illustrate the matrix and vector (N) representations at the j^{th} column iteration in Figure 4.

- $A_{(l)}$ represents the l^{th} column of the matrix A. In case of a vector, it stands for the l^{th} index of the vector.

Observe that both $A^{[0]}$ and $A^{(0)}$ represent the original matrix A in our notations. Now let us focus on the compression kernels that we will compare in this report.

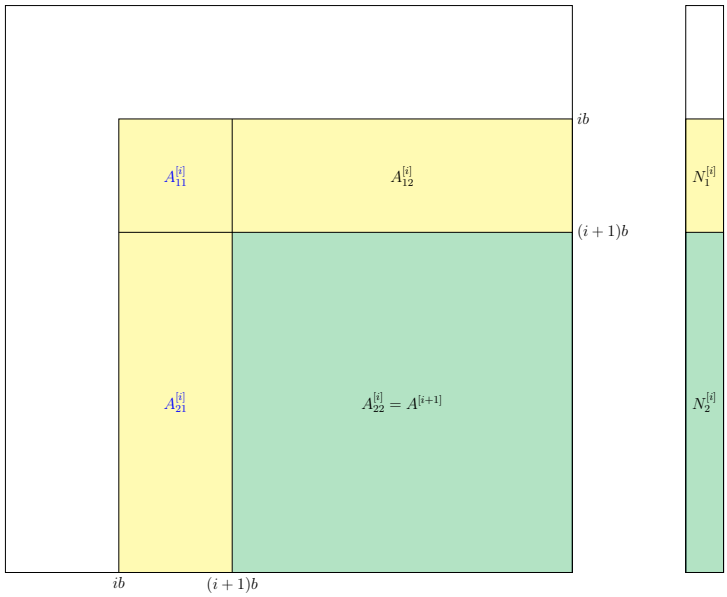


Figure 3: Our notations and the sub-parts they stand for during the factorization. On the left we present the panel-wise matrix (A) notations, while on the right we show the vector notations on an arbitrary vector N.

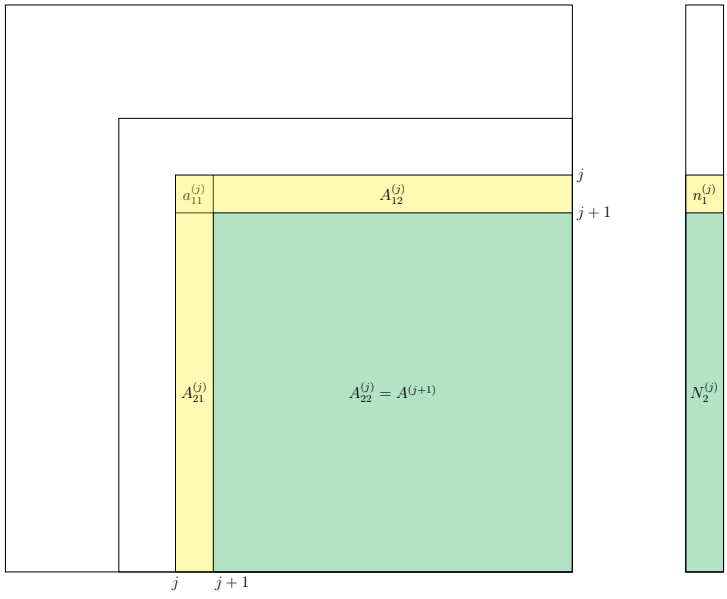


Figure 4: Our notations and the sub-parts they stand for during the factorization. On the left we present the column-wise matrix (A) notations, while on the right we show the vector notations on an arbitrary vector N.

5 QR based compression kernels

In this section, we study four different QR based compression methods in detail. We explain our most basic method, QR with column pivoting (QRCP), in Section 5.1. In Section 5.2, the randomized QR with column pivoting (RQRCP) is studied. Then, the left-looking version of the RQRCP method is provided in Section 5.3. Finally, a rank revealing QR based compression kernel which adopts a rotational matrix is explained in Section 5.4.

5.1 QR with column pivoting (QRCP)

QR with column pivoting (QRCP) can be seen as the most basic method in this report. The QRCP we adopt is based on the LAPACK $xGEQP3/xLAQPS$ methods, similar to [8]. We start with a version of $xGEQP3$ which was modified by Alfredo Buttari for the MUMPS BLR solver [19]. This method uses panel-wise approach to take advantage of the level-3 BLAS operations and thus reduces the slow memory-cache passes of large matrices.

QRCP has an intra-panel left-looking feature. That is, each column of the panel is updated just before being factorized. On the other hand, it uses a right-looking approach for the inter-panel operations. Therefore, at the end of each panel iteration, the whole trailing matrix is updated with the current panel contributions. In Figure 5, we illustrate the intra-panel left-looking feature, while in Figure 6 we show the inter-panel right-looking feature of QRCP. In the figures, j stands for the current index and r is the last index of the previous panel. The yellow parts are used for updating the green ones similarly to Figure 2.

Algorithm 1 illustrates the QRCP method. Here, C stands for the column norms array (of size n), while P represents the permutation matrix. In practice, P is an array, which keeps all the corresponding pivot indices, $p^{(j)}$, to swap the columns cheaply. In our code, the column norms are either updated or explicitly computed (only when required) as in [8]. Then, as we have the 2-norm values of the columns thanks to the column norms array C, whenever our pivot column norm is smaller than the required precision, we compute the trailing matrix norm. More precisely, we compute the Frobenius norm of the trailing matrix by computing the 2-norm of array C. Here, if the trailing matrix norm is also smaller than the required precision (our stopping criterion), we set the numerical rank to the current index value and terminate the function.

As seen in the algorithm, we start each column iteration by checking our stopping criterion. If it is not satisfied, the column which has the maximum norm-2 value is chosen as the pivot. As the algorithm is left-looking intra-panels, at each column iteration j, we update the current column by using the computed panel reflector matrix, $Y^{(j)}$, and corresponding part of the panel accumulation matrix, i.e. ${W_1^{(j)}}^T$. In Figure 5, the yellow parts illustrate these contributing parts to the current green column. After generating the elementary Householder reflection for the current column, we compute the accumulations caused by the

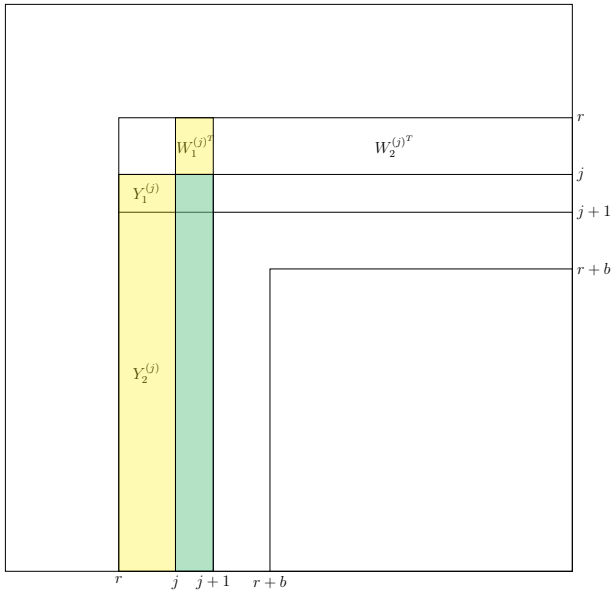


Figure 5: Left-looking intra-panel update of QRCP. The updated column is in green, whereas the contributing parts are in yellow.

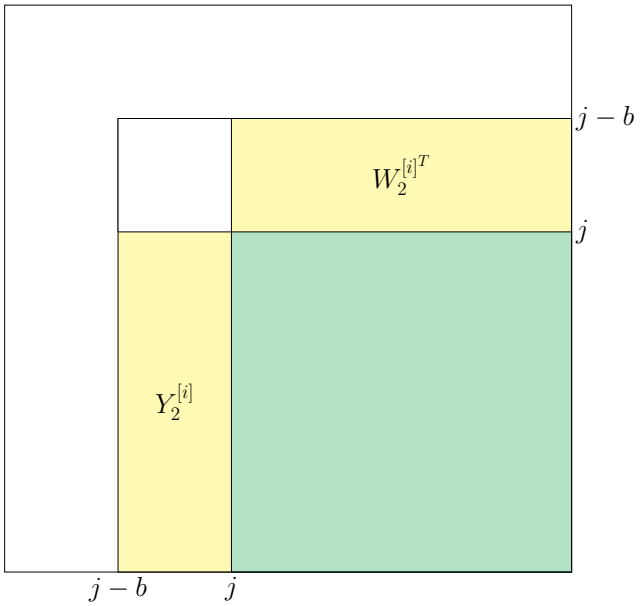


Figure 6: Right-looking inter-panel update of QRCP. The updated parts are in green, whereas the contributing ones are in yellow.

Algorithm 1 Block QR with Column Pivoting

function QRCP(A, ϵ, b)
  for l = 1, 2, ..., n **do**
    $C_{(l)} = ||A_{(l)}||_2$ ▷ Initialize column norms array
  for i = 0, b, 2b, ..., min(m,n) **do** ▷ Panel iteration
    for j = i, i+1, ..., min(i+b-1, min(m,n)) **do** ▷ Column iteration
      if $||C^{(j)}|| \leq \epsilon||A||$ **then return** $r = j$ ▷ Check the stopping criterion
      $UpdateNorms(C^{(j)})$ ▷ Update the column norms
      $p^{(j)} = \underset{l \in [j,\ldots,n]}{\operatorname{argmax}} C_{(l)}$
      $swap(A_{(j)}, A_{(p^{(j)})})$, $swap(W_{(0)}^{(j)^T}, W_{(p^{(j)}-j)}^{(j)^T})$ ▷ Swap pivot column
      $\begin{bmatrix} a_{11}^{(j)} \\ A_{21}^{(j)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(j)} \\ A_{21}^{(j)} \end{bmatrix} - Y^{(j)} W_{(0)}^{(j)^T}$ ▷ Update the current column
      $H^{(j)} = I - y^{(j)} \tau^{(j)} y^{(j)^T}$ ▷ Compute the Householder reflection
      $w^{(j)T} = \tau^{(j)}(y^{(j)^T} \begin{bmatrix} A_{12}^{(j)} \\ A_{22}^{(j)} \end{bmatrix} - (y^{(j)^T} Y^{(j)}) W^{(j)^T})$ ▷ Find accumulation
      $R_{12}^{(j)} = A_{12}^{(j)} - Y_1^{(j)} W_2^{(j)^T}$ ▷ Update current row
    $A^{[i+1]} = A^{[i+1]} - Y_2^{[i]} W_2^{[i]^T}$ ▷ Update the trailing matrix
  Output : [Q, R, P, r]

current row, $w^{(j)^T}$. As we mentioned before, this storage for the accumulations is needed in the panel-wise column pivoted QR since the trailing matrix is not updated at each iteration, while the column norms should be updated. Note that, the reflector and accumulation matrices are the concatenation of each column reflection vector, $Y = [y^{(0)}, y^{(1)}, ...]$, and accumulation vector, $W^T = [w^{(0)}, w^{(1)}, ...]^T$, respectively. Finally, each column iteration terminates by updating the current row of the matrix.

Let us represent the panel reflectors at the end of the i^{th} panel iteration as $Y^{[i]} = \begin{bmatrix} Y_1^{[i]} \\ Y_2^{[i]} \end{bmatrix}$ and the accumulation matrix as $W^{[i]^T} = \begin{bmatrix} W_1^{[i]^T} & W_2^{[i]^T} \end{bmatrix}$. Then, $Y_2^{[i]}$ stands for the $A_{21}^{[i]}$ part of Figure 3, while $W_2^{[i]^T}$ represents the contributions from $A_{12}^{[i]}$. In other words, the left-most and the upper yellow parts of Figure 6 illustrate $Y_2^{[i]}$ and $W_2^{[i]^T}$, respectively. In Algorithm 1, every time the inner loop ends, the computation of $R_{11}^{[i]}$, $R_{12}^{[i]}$, $Y^{[i]}$ and $W^{[i]}$ is completed. Then, at each panel iteration, the whole trailing matrix is updated by using the contributions from these yellow parts as it has an inter-panels right-looking approach.

Note that all the methods in this report use intra-panel left-looking approach.

Therefore, from now on, the algorithms will be called left-looking or right-looking according to their inter-panel update features.

Observe that Line 12 in the panel-wise column pivoted version of the QR algorithm (Algorithm 1) requires large level-2 BLAS operations, i.e. ${y^{(j)}}^T \begin{bmatrix} A_{12}^{(j)} \\ A_{22}^{(j)} \end{bmatrix}$ and ${y^{(j)}}^T Y^{(j)}$. These level-2 BLAS operations are inefficient because of the memory passes for large matrices at each column iteration. Therefore, we will explain a technique to avoid these inefficient memory passes in the next section.

The QRCP method reduces the complexity compared to SVD (with the complexity $\mathcal{O}(n^3)$) to $\mathcal{O}(n^2 r)$. Here, the pivot selection operation costs $\mathcal{O}(n^2 r)$, while applying the pivot is as cheap as simple column swaps. As QRCP is right-looking inter-panels, the whole trailing matrix updates are performed at a cost of $\mathcal{O}(n^2 r)$.

5.2 Randomized QR with column pivoting (RQRCP)

As mentioned in the previous section, the need of pivoting leads to many inefficient level-2 BLAS operations in the panel-wise QRCP factorization, which reduces the performance for large matrices. In this respect, some randomization methods are proposed to use a lower dimensional representative matrix, instead of the original one, to determine the pivot columns of the panels. For this purpose, a Gaussian matrix with independent and identically distributed random variables (Gaussian i.i.d.), Ω, can be used to project the original matrix onto the sample matrix, B, as

$$B_{d \times n} = \Omega_{d \times m} A_{m \times n},$$

where $d \ll m$. Here, we need to get the representative data with small uncertainity to be able to determine the correct pivoting for the current panel. That is, let us consider the j^{th} column of A, $a_{(j)}$, and the corresponding sample matrix column $b_{(j)}$. Then, $\frac{||b_{(j)} - b_{(i)}||_2^2}{d||a_{(j)} - a_{(i)}||_2^2} - 1 \leq \tau$, with $0 < \tau < 0.5$ being the threshold and $0 \leq i, j < n$, should be satisfied with high probability [9]. This is why we do not choose the row dimension of B as block size b. Instead, we use a pre-defined oversampling size, p. Then, the row dimension is $d = b + p$. In [24], it is shown that the failure probability of RQRCP is exponentially decreasing for larger oversampling sizes. Although we need this parameter for the quality of the sampling, we should be careful to choose it as small as possible, while respecting the error bounds, to efficiently improve the performance of the randomized kernel.

Note that we have to resample the matrix at each panel iteration in the classical approach to be able to determine the panel pivots. However, it is too expensive in practice. Instead, we can update the sample matrix in a cheaper way and use it at each iteration as proposed in [9, 16]. In these works, the authors prove that the performance results of these randomized techniques become close to the unpivoted QR methods, while the accuracy is very close to

the one of unrandomized methods. Therefore, in this section and in Section 5.3, we adopt the randomized methods through the updated sample versions as in these works.

As explained in [9], considering the block factorization of the matrix B

$$BP^{[i]} = \Omega A P^{[i]} = Q_B^{[i]} \begin{bmatrix} S_{11}^{[i]} & S_{12}^{[i]} \\ 0 & S_{22}^{[i]} \end{bmatrix},$$

the update of the sample matrix can be computed as

$$\begin{bmatrix} B_1^{[i]} \\ B_2^{[i]} \end{bmatrix} = \Omega^{[i]} A^{[i]} = \begin{bmatrix} S_{12}^{[i]} - W_{11}^{[i]} R_{12}^{[i]} \\ S_{22}^{[i]} \end{bmatrix} = \begin{bmatrix} S_{12}^{[i]} - S_{11}^{[i]} {R_{11}^{[i]}}^{-1} R_{12}^{[i]} \\ S_{22}^{[i]} \end{bmatrix}.$$

Algorithm 2 Randomized QR with Column Pivoting

1: **function** RQRCP(A, ϵ, b)
2: $\Omega^{[0]}$ ▷ Generate the Gaussian i.i.d. matrix
3: $B^{[0]} = \Omega^{[0]} A^{[0]}$ ▷ Compute the sample matrix
4: **for** $i = 0, 1, ..., \ulcorner \frac{min(m,n)}{b} \urcorner - 1$ **do**
5: $[Q_B^{[i]}, S^{[i]}, P_B^{[i]}, r] = QRCP(B^{[i]}, \sqrt{(b+p)}\epsilon, b)$ ▷ Compute permutation
6: $A = AP_B^{[i]}$ ▷ Apply the permutation
7: $[Q^{[i]}, R_{11}^{[i]}] = QR(A^{[i]})$ ▷ Apply QR on the permuted matrix A
8: $\begin{bmatrix} R_{12}^{[i]} \\ A^{[i+1]} \end{bmatrix} = {Q^{[i]}}^T \begin{bmatrix} A_{12}^{[i]} \\ A_{22}^{[i]} \end{bmatrix}$ ▷ Update rightmost matrix
9: $B_1^{[i+1]} = S_{12}^{[i]} - S_{11}^{[i]} {R_{11}^{[i]}}^{-1} R_{12}^{[i]}$ and $B_2^{[i+1]} = S_{22}^{[i]}$ ▷ Update the sample matrix

Output : $[Q, R, P, r]$

Now, let us observe the randomized QRCP method, with the sample matrix update formula, in Algorithm 2. Here, the stopping criterion is applied implicitly through the QRCP function, which is used for computing the permutation matrix P_B through the sample matrix. As the permutation is computed through the sample matrix, the threshold should also be arranged according to B. As explained in [9], the column norms of B and A have a constant relative ratio through the Chi-squared distribution [6]. As a consequence, their residual error ratio is approximately $\sqrt{\frac{1}{b+p}}$, with $b+p$ being the row dimension of the sample matrix. Therefore, assuming ϵ is the threshold for the matrix A, the threshold for the sample matrix B is computed as

$$\epsilon_B = \sqrt{(b+p)}\epsilon.$$

As seen in the algorithm, the RQRCP method is similar to the QRCP method (Algorithm 1), except for the computation of the permutation matrix and the sample matrix generation/update operations. Then, in this algorithm,

the additional cost to QRCP is the cost of generating the sample matrix $B_{(b+p)\times n}$ in the beginning, $\mathcal{O}(n^2 b)$, as well as its update at all iterations, $\mathcal{O}(nrb)$. However, the pivot finding operation using the input matrix A, $\mathcal{O}(n^2 r)$, is avoided. The pivoting operation through the sample matrix reduces this cost to $\mathcal{O}(nrb)$. Through these computational complexities, we can observe the randomization method does not aim to reduce the theoretical complexity compared to QRCP. Instead, it targets to reduce the time consuming data movements that occur when computing the column norms through large matrices.

5.3 Truncated randomized QRCP (TQRCP)

As explained in Section 2.2, the left-looking feature aims to reduce the complexity by avoiding the whole trailing matrix updates. In [9], the authors propose the left-looking version of the RQRCP method for this purpose. In this section, we study this left-looking randomized method, which is called truncated randomized QRCP (TQRCP).

Algorithm 3 Truncated Randomized Block QR with Column Pivoting

1: **function** TQRCP(A, ϵ, b)
2: $\Omega^{[0]}$ ▷ Generate the Gaussian i.i.d. matrix
3: $B^{[0]} = \Omega^{[0]} A^{[0]}$ ▷ Compute the sample matrix
4: **for** $i = 0, 1, ..., \ulcorner \frac{min(m,n)}{b} \urcorner - 1$ **do**
5: $[Q_B^{[i]}, S^{[i]}, P_B^{[i]}, r] = QRCP(B^{[i]}, \sqrt{(b+p)}\epsilon, b)$ ▷ Compute permutation
6: $A = AP_B^{[i]}$ and ${W_G^{[i]}}^T = {W_G^{[i]}}^T P_B^{[i]}$ ▷ Apply the permutation
7: $\begin{bmatrix} A_{11}^{[i]} \\ A_{21}^{[i]} \end{bmatrix} = \begin{bmatrix} A_{11}^{[i]} \\ A_{21}^{[i]} \end{bmatrix} - \begin{bmatrix} Y_{G1}^{[i]} \\ Y_{G2}^{[i]} \end{bmatrix} {W_{G1}^{[i]}}^T$ ▷ Update current panel
8: $[Q^{[i]}, R_{11}^{[i]}, Y^{[i]}] = QR(\begin{bmatrix} A_{11}^{[i]} \\ A_{21}^{[i]} \end{bmatrix})$ ▷ Apply QR on the current panel
9: ${W^{[i]}}^T = {T^{[i]}}^T ({Y^{[i]}}^T A - ({Y^{[i]}}^T Y_G^{[i]}) {W_G^{[i]}}^T)$ ▷ Compute the accumulations
10: $R_{12}^{[i]} = A_{12}^{[i]} - Y_{G1}^{[i]} {W_{G2}^{[i]}}^T$ ▷ Update the row panel
11: $B_1^{[i+1]} = S_{12}^{[i]} - S_{11}^{[i]} {R_{11}^{[i]}}^{-1} R_{12}^{[i]}$ and $B_2^{[i+1]} = S_{22}^{[i]}$ ▷ Update the sample matrix

Output : $[Q, R, P, r]$

As explained before, in the right-looking approach, we use panel-wise reflectors and panel-wise accumulations to update the whole trailing matrix. In the left-looking approach, we use global reflectors, Y_G, and global accumulations, W_G^T, to update only the current panel. In Figure 7, we illustrate the Y_G and W_G^T matrices of the left-looking approach at the beginning of the i^{th} panel iteration. Here, the yellow global reflectors and yellow global accumulations are used to update the current green panel.

In Algorithm 3, we provide the code for the left-looking TQRCP method. As

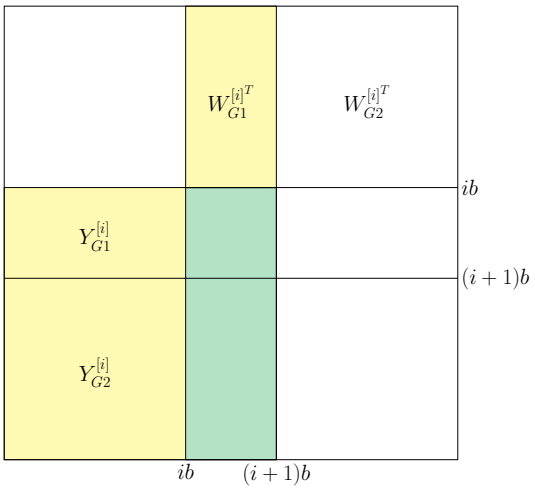


Figure 7: Global reflectors, Y_G, and global accumulations, W_G^T, at the i^{th} panel iteration. Here, the yellow parts are used to update the current green panel.

explained in [9], TQRCP follows the same logic as converting the column-wise QRCP to the panel-wise QRCP, but at a panel-wise level. That is, let us consider the composition of all the global reflectors, $Y_G^{[i+1]}$, as

$$Y_G^{[i+1]} = \begin{bmatrix} Y_G^{[i]} & Y^{[i]} \end{bmatrix},$$

the block reflectors are updated as

$$(I - Y_G^{[i]} T_G^{[i]} {Y_G^{[i]}}^T)(I - Y^{[i]} T^{[i]} {Y^{[i]}}^T) = I - Y_G^{[i+1]} T_G^{[i+1]} {Y_G^{[i+1]}}^T,$$

where $T_G^{[i+1]}$ is

$$T_G^{[i+1]} = \begin{bmatrix} T_G^{[i]} & -T_G^{[i]} {Y_G^{[i]}}^T Y^{[i]} T^{[i]} \\ 0 & T^{[i]} \end{bmatrix}.$$

Then, the global accumulation matrix, which results in extra storage, for this panel-wise left-looking code is

$$W_G^{[i+1]^T} = T_G^{[i+1]^T} Y_G^{[i+1]^T} A = \begin{bmatrix} W_G^{[i]^T} \\ W^{[i]^T} \end{bmatrix} = \begin{bmatrix} T_G^{[i]^T} Y_G^{[i]^T} A \\ T^{[i]^T} (Y^{[i]^T} A - (Y^{[i]^T} Y_G^{[i]}) W_G^{[i]^T}) \end{bmatrix}.$$

As seen in the algorithm, TQRCP is indeed the panel-wise left-looking version of RQRCP. That is, after each panel iteration, RQRCP updates $\begin{bmatrix} A_{12}^{[i]} \\ A_{22}^{[i]} \end{bmatrix}$, while TQRCP stores the contribution accumulations and only updates $A_{12}^{[i]}$.

Observe that TQRCP also applies the stopping criterion implicitly, similarly to RQRCP.

Compared to RQRCP, there is an extra storage cost for the matrix W_G^T, of size $r \times n$, in the TQRCP method. However, at each panel iteration of the TQRCP method, the update of $A_{22}^{[i]}$, with size $(n-r-b) \times (n-r-b)$, is avoided. This reduces the total trailing matrix update cost to $\mathcal{O}(nr^2)$ in TQRCP compared to $\mathcal{O}(n^2r)$ in RQRCP. The complexity of computing W_G^T in TQRCP is $\mathcal{O}(n^2b)$. As the sample matrix generation, of complexity $\mathcal{O}(n^2b)$, is performed only once at the beginning of the algorithm, it is not necessarily dominant since actually it has a constant 1. Then, thanks to the low cost trailing matrix updates, TQRCP has the lowest computational complexity among the four methods that we study. For large matrices with low ranks, this method highly reduces the update cost of the trailing matrix.

5.4 Randomized QR with rotation (RQRRT)

In this section, the rotational QR method in [14] is studied. This method is proposed to fix the rank revealing issue of the column pivoted QR method on some matrices.

Remember that the rank revealing QR factorization aims to produce an approximation as close as possible to the optimal approximation (SVD) (see the approximation criterion 9). Then as explained in [14], we can apply a convenient orthonormal matrix to A to produce a rank closer to SVD to satisfy this criterion, instead of only permuting the columns.

In the randomized QR factorization with rotation (RQRRT) method, the sample matrix is used similarly to the randomized QRCP. However, this time the original matrix, A, is not permuted through the permutation that is obtained by using QRCP of the sample matrix, B. Instead, an orthonormal matrix is generated through the unpivoted QR factorization of B^T. This is indeed an effective method to get the mass to the left of the matrix, since the sample matrix is generated in a way that it really represents the linear dependencies of the original matrix. Therefore, this orthonormal matrix represents the rotation of the input matrix when the QR factorization is applied. When this rotation matrix is applied to matrix A, its representative data moves to the diagonals to have closer results to SVD.

Algorithm 4 stands for the RQRRT compression kernel. Here, the output rotational matrix is equivalent to $Q_{rot} = Q_B^{[0]} Q_B^{[1]} Q_B^{[2]} ... Q_B^{[\lceil \frac{r}{b} \rceil - 1]}$. Let us compare this RQRRT algorithm to the randomized QRCP (Algorithm 2). Their first difference is the way of gathering the representative data. In the randomized QRCP algorithm, we find

$$AP_B = Q_A R_A,$$

where P_B is obtained through the QRCP algorithm as

Algorithm 4 Randomized Block QR with Rotation

1: **function** RQRRT(A, ϵ, b)
2: $\Omega^{[i]}$ ▷ Generate the Gaussian i.i.d. matrix
3: **for** $i = 0, 1, 2, ..., \ulcorner \frac{min(m,n)}{b} \urcorner - 1$ **do**
4: $B^{[i]} = \Omega^{[i]} A^{[i]}$ ▷ Resample B
5: $[Q_B^{[i]}, S^{[i]}] = QR({B^{[i]}}^T, b)$ ▷ Compute the rotation matrix
6: $A^{[i]} = A^{[i]} Q_B^{[i]}$ ▷ Apply the rotation
7: $[Q^{[i]}, R_{11}^{[i]}] = QR(A^{[i]})$ ▷ Apply QR on A
8: $\begin{bmatrix} R_{12}^{[i]} \\ A^{[i+1]} \end{bmatrix} = {Q^{[i]}}^T \begin{bmatrix} A_{12}^{[i]} \\ A_{22}^{[i]} \end{bmatrix}$ ▷ Update the trailing matrix
9: $r = StoppingCriterion(R_{11}^{[i]}, R_{22}^{[i]}, ||A^{[0]}||_F, \epsilon, b)$ ▷ Check stopping criterion

Output : [Q, R, Q_{rot}, r]

$$BP_B = Q_B R_B.$$

However, in the RQRRT method, the $AQ_B^{'}$ matrix is used as

$$AQ_B^{'} = Q_A^{'} R_A^{'},$$

where the orthogonal matrix is obtained from the partial QR of the sample matrix as

$$B^T = Q_B^{'} R_B^{'}.$$

This shows the column orthonormalization of B^T. Therefore, the leftmost b column spanning the matrix A will have almost the same spanning as the leading b singular vectors of A. Therefore, it gets a closer result to SVD in terms of rank revealing quality [14].

The second difference of the rotational method is the resampling operation. In the column pivoted method, the sample matrix is cheaply updated at each panel iteration. However, in the rotational method, as its transpose is used, there is no update formula (up to our knowledge).

The third difference of the rotational method is the explicitly implemented stopping criterion (at Line 9), since we do not call QRCP function anywhere in this code. Here, the input matrix $R_{22}^{[i]}$ stands for $A^{[i+1]}$.

The detailed stopping criterion in RQRRT code is given in Algorithm 5. Everytime this function is called, the trailing matrix norm on Line 2 is explicitly computed. Here, it is important to observe that at the end of the panel iteration, if the criterion (on Line 2) is satisfied, we check the previous trailing matrix norms by adding each column contribution starting from the last column of the panel (on Line 4). Bear in mind that the value $||(R_{11}^{[i]})_{(l)}||_2$ does not have to be computed explicitly since it is equal to the magnitude of the first element of

this column. If we check the trailing matrix norms with the accumulations of the left-most columns first, we can lose stability because of the round-off errors. Note that in the QRCP method, the stability issue is already taken care of and it is out of scope in this thesis.

In Figure 8, the stability issue is observed. Here, the result is obtained by using the *S-shape* matrix (see Section 7.1) of size 500×500, where the generation rank is 25. In the figure, RQRRT (right) represents our algorithm and RQRRT (left) stands for the unstable accumulation version. As a reference, the trailing matrix norms at each panel index at generation, $\sqrt{\sum_{i=K}^{n} \sigma_i^2}$, is also shown (called Reference). The y-axis stands for $||A - U_K V_K^T||_F$, while x-axis shows the indices, K, of the first panel iteration. Here, U_K is of dimension $m \times K$ and represents the K columns of the matrix U. Similarly, V_K^T stands for the first K rows of V^T.

Figure 8 shows that indeed the right-most accumulation gets closer results to Reference. As opposed to our approach, the left-most accumulation diverges after the index 23 supporting our statement.

Algorithm 5 Stopping criterion in RQRRT

function *StoppingCriterion*($R_{11}^{[i]}$, $R_{22}^{[i]}$, $||A^{[0]}||_F$, ϵ, b)
  if $||R_{22}^{[i]}||_F \leq \epsilon ||A^{[0]}||_F$ **then**
    for $l = b-1, b-2, ..., 1$ **do** ▷ Accumulations from right-most column
      $||R_{22}^{(ib+l)}||_F = \sqrt{||R_{22}^{(ib+l+1)}||_F^2 + ||(R_{11}^{[i]})_{(l)}||_2^2}$
      if $||R_{22}^{(ib+l)}||_F > \epsilon ||A^{[0]}||_F$ **then return** $r = ib + l + 1$
Output : $[r]$

For RQRRT, finding the rotation matrix has a complexity of $\mathcal{O}(nrb)$, while applying it to the matrix A is of cost $\mathcal{O}(n^2 r)$. However, for RQRCP, the pivot finding cost is $\mathcal{O}(nrb)$ and the AP_B operation is only as cheap as swapping b columns of A. Another cost difference of RQRRT is the resampling at all iterations, $\mathcal{O}(n^2 r)$, compared to the sample matrix update cost in RQRCP, $\mathcal{O}(nrb)$. As seen here, the complexity differences between these two methods can be significant for large matrix sizes, which makes RQRRT the most costly QR method that we study.

6 Summary of the complexities

In this section, we aim to summary and comment on the complexities of the compression kernels studied in the previous sections. Table 1 illustrates the computation complexities of the compression methods for a generic matrix of size $n \times n$, with a block size b and a compression rank r. Here, only the operations leading to a difference between the methods are shown for the sake of simplicity. The third column of the table stands for the total cost of each operation, while

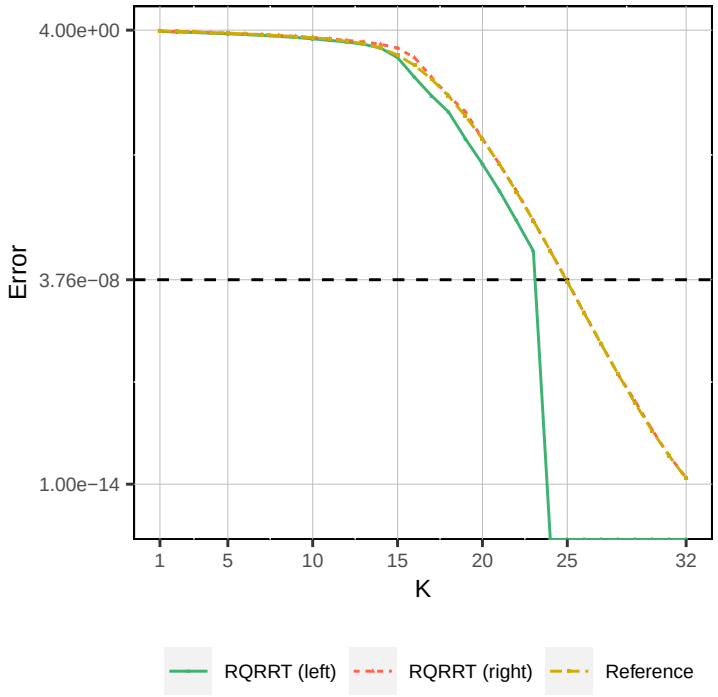


Figure 8: The round-off error impact when the trailing matrix norm columns are accumulated starting from right or left. The y-axis stands for $||A - U_K V_K^T||_F$ in logarithmic scale, while x-axis stands for each index in the first panel iteration. The result represents the *S-shape* matrix (see Section 7.1) of size 500×500, where the generation rank is 25.

Method	Operation	Total cost	Overall cost
SVD			$\mathcal{O}(n^3)$
QRCP	Pivot finding	$\mathcal{O}(n^2r)$	$\mathcal{O}(n^2r)$
	Trailing matrix update	$\mathcal{O}(n^2r)$	
RQRCP	Sample matrix generation (beginning)	$\mathcal{O}(n^2b)$	$\mathcal{O}(n^2r)$
	Pivot finding	$\mathcal{O}(nrb)$	
	Update of sample matrix B	$\mathcal{O}(nrb)$	
	Trailing matrix update	$\mathcal{O}(n^2r)$	
TQRCP	Sample matrix generation (beginning)	$\mathcal{O}(n^2b)$	$\mathcal{O}(n^2b)$
	Pivot finding	$\mathcal{O}(nrb)$	
	Update of current panel	$\mathcal{O}(nr^2)$	
	Generating accumulation matrix (W^T)	$\mathcal{O}(n^2b)$	
	Update of sample matrix B	$\mathcal{O}(nrb)$	
RQRRT	Resampling (all iterations)	$\mathcal{O}(n^2r)$	$\mathcal{O}(n^2r)$
	Rotation computing	$\mathcal{O}(nrb)$	
	Rotation of A	$\mathcal{O}(n^2r)$	
	Trailing matrix update	$\mathcal{O}(n^2r)$	

Table 1: Important complexity changes between the compression methods. Here, n, r and b stand for the matrix size, the compression rank and the block size, respectively.

the last column shows the overall cost of each method. When determining the values in the last column, we assume $r > b$.

As seen in the table, the truncated method has a total of $\mathcal{O}(n^2b)$ complexity, while all other QR variants have a $\mathcal{O}(n^2r)$ complexity. As in general $b < r$ and TQRCP reduces the trailing matrix update cost, this truncated method is the cheapest procedure for large matrices with low ranks, in a sequential environment. Since SVD has a $\mathcal{O}(n^3)$ complexity, it is the most costly method, as expected.

For the QRCP, RQRCP and RQRRT methods, the overall complexity is asymptotically the same. However, when each operation complexity is observed, one can see that the usage of the rotational feature in RQRRT introduces large additional costs compared to the other two methods. That is why, it is the most costly QR variant. In RQRCP, the pivoting cost is reduced compared to QRCP. However, there are additional costs for sample matrix generation (in the beginning) and for updating it at each iteration. Thus, this method is advantageous for large matrices, where mostly the communication of the column norms computations are costly. To conclude this table, we expect in our experiments to observe the performance order as (from worst to best, in a sequential environment): $SVD \ll RQRRT < RQRCP < QRCP \ll TQRCP$.

7 Experiments on generated matrices

In this section, we compare the compression methods on different generated matrices, including challenging cases. For the sake of simplicity, the row and column dimensions are chosen to be equal. From now on, the letter r_G will stand for the generation rank of a matrix, whereas r will represent its approximation rank.

All the experiments are performed through the block low-rank supernodal direct solver PaStiX [21], using `miriel` nodes of the *Plafrim*[1] supercomputer. Each `miriel` node is equipped with two Intel `Xeon E5-2680v3` 12-cores running at 2.50 GHz and 128 GB of memory.

As a matter of interest, the compression kernels are in sequential within the PaStiX framework. Therefore, the results in this section demonstrates the sequential results, since parallelism is not worth for the matrix sizes we are interested in. For the RQRCP and TQRCP methods, the block size is set as $d = b + p$, where $b = 27$ and the oversampling parameter is set to $p = 5$. For the other methods, the block size is $d = b = 32$. Let us emphasize that we implement all the compression methods in a similar fashion and we determine the compression ranks numerically. Here, let us also note that setting $b = 32$ and $p = 5$ would be a better choice when comparing RQRCP and TQRCP to the other methods, where $d = b = 32$. However, it will not affect our results observably.

In Section 7.1, we first provide the information about how we generate the experimental matrices. Then, in Sections 7.2, 7.3, 7.4 and 7.5, we discuss the stability, performance, numerical ranks and accuracy results on these matrices, respectively.

7.1 Generation of the experimental matrices

In this section, five different type of generated matrices are used. These matrices are similar to the ones in [14], and are shown in Figure 9, Figure 10 and Figure 11. In these figures, the x-axis stands for the matrix index, K. The figures on the left illustrate the singular value distribution of a matrix of size 500×500, where the generation rank is 100 at the generation precision $\epsilon = 10^{-8}$. The figures on the right stand for the relative Frobenius norm of the trailing matrix,

$$\frac{\sqrt{\sum_{i=K}^{n} \sigma_i^2}}{||A||_F}$$

, of the same matrices. The solid black lines in the figures represent the generation precision.

The first matrix type is called *k-rank* and it is observed in Figure 9. In order to generate this matrix, a normalized Gaussian i.i.d. matrix is used, where the singular values decrease slowly and suddenly drop to ignorable values at the

[1] https://www.plafrim.fr

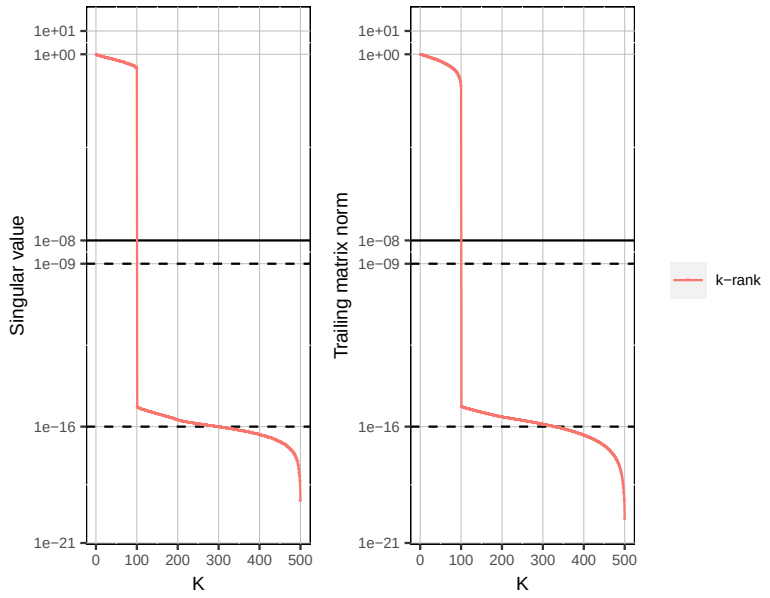


Figure 9: The *k-rank* matrix of size 500×500, where the generation rank is $r_G = \frac{20n}{100} = 100$ at the generation precision $\epsilon = 10^{-8}$. The Frobenius norms of the trailing matrices at each index ($\frac{\sqrt{\sum_{i=K}^{n} \sigma_i^2}}{||A||_F}$) are on the right, while the singular values at each index (σ_K) are on the left. The x-axis stands for the matrix indices, K. The singular values firstly decay slowly, then suddenly drop to very low values.

generation rank. Because of the sudden drop to ignorable values at r_G, this matrix type is trivial for the compression methods.

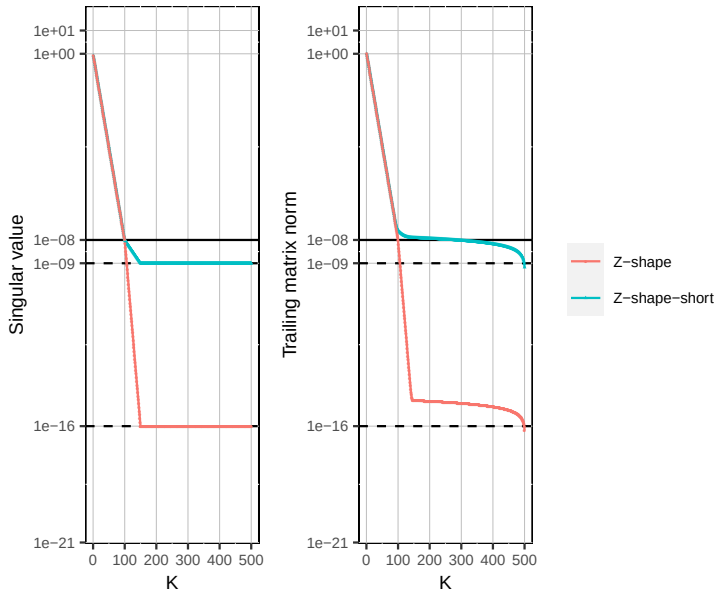


Figure 10: The *Z-shape* and *Z-shape-short* matrices of size 500×500, where the generation rank is $r_G = \frac{20n}{100} = 100$ at the generation precision $\epsilon = 10^{-8}$. The Frobenius norms of the trailing matrices at each index ($\frac{\sqrt{\sum_{i=K}^{n} \sigma_i^2}}{||A||_F}$) are on the right, while the singular values at each index (σ_K) are on the left. The x-axis stands for the matrix indices. Both matrices have the same singular values up to the index $r_G = 100$. However, after this point, *Z-shape* case singular values decrease faster and reach to almost machine precision at the index $\frac{3}{2}r_G$, whereas the *Z-shape-short* case singular values reach to $\frac{\epsilon}{10} = 10^{-9}$ at this index.

The second and third matrix types are called *Z-shape* and *Z-shape-short*, which are shown in Figure 10. Both of these matrices are computed as $A = UDV^T$, where U and V are random orthonormal matrices. The singular values (diagonal values of the matrix D) are the same for both cases and decrease rapidly up to the generation rank. However, the *Z-shape* drops faster to ignorable values at the index $\frac{3}{2}r_G = 150$, while *Z-shape-short* stagnates at 10^{-9}. Here, we expect the methods to find close approximation ranks to $r_G = 100$ for the *Z-shape* because of the fast drop after the generation rank. Opposite to *Z-shape*, as seen in the right-most figure, the trailing matrix of the *Z-shape-short* matrix contains important data at r_G with respect to the compression precision 10^{-8}. Thus, this case is challenging for our stopping criterion to determine the compression rank close to the generation one.

The last matrix types are called *S-shape* and *S-shape-short*, which are shown

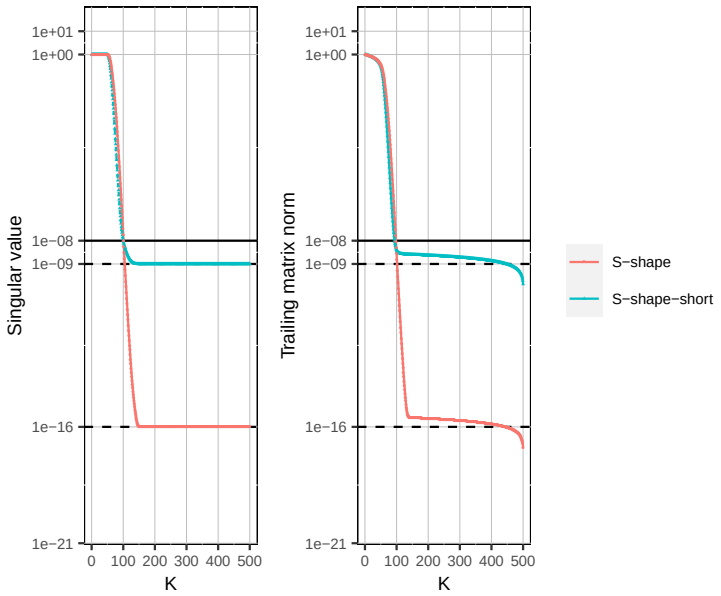


Figure 11: The *S-shape* and *S-shape-short* matrices of size 500×500, where the generation rank is $r_G = \frac{20n}{100} = 100$ at the generation precision $\epsilon = 10^{-8}$. The Frobenius norms of the trailing matrices at each index ($\frac{\sqrt{\sum_{i=K}^{n} \sigma_i^2}}{||A||_F}$) are on the right, while the singular values at each index (σ_K) are on the left. The x-axis stands for the matrix indices. The singular values of the *S-shape* and *S-shape-short* matrices firstly stagnate at the value 1, then they decrease fast and in the end they stagnate again after a heavy tail. Both of the singular values reach the value $\epsilon = 10^{-8}$ at the index $r_G = 100$. However, the last stagnation point is $\epsilon^2 = 10^{-16}$ for the *S-shape* matrix, whereas it is $\frac{\epsilon}{10} = 10^{-9}$ for the *S-shape-short* matrix.

in Figure 11. These matrices are computed as $A = UDV^T$, with random orthonormal matrices U and V. For these matrices, the singular values stagnate at 1 up to the index $\frac{r_G}{2}$, and then they drop rapidly to reach $\epsilon = 10^{-8}$ at index r_G. After that point, the *S-shape* case stagnates at the value $\epsilon^2 = 10^{-16}$, while *S-shape-short* stagnates at $\frac{\epsilon}{10} = 10^{-9}$, which is close to the generation precision. These matrices are challenging experiments for the randomization methods as they have a heavy tail.

In the following sections, we use the compression precision $\epsilon = 10^{-8}$, which is equal to the generation precision. We observe the SVD method with two different stopping criteria. One of them uses the Frobenius norm of the trailing matrix to determine the compression rank as in 7, while the other uses the spectral norm for this purpose as in 6. The former, SVDF, provides better comparison to the QR methods, whereas the latter, SVD2, leads to closer ranks to the generation ones.

7.2 Stability

In this section, we observe the stability of the compression kernels. In Figure 12, the relative residual with respect to the initial matrix, $\frac{||A-U_K V_K^T||_F}{||A||_F}$, is observed at each index K. As these values show the approximated values of the relative Frobenius norms that were presented on the right side of Figures 9, 10 and 11, we want to obtain close results to them.

As seen in Figure 12, all the methods have good stability. As RQRCP and TQRCP use a relative stopping criterion, which uses an approximation ratio between the original and sample matrix columns, there are more magnitude changes between the panel iterations. However, it will be shown in Section 7.5 that these methods still ensure a good accuracy, so this does not indicate a problem. As we can see in the figures, the RQRRT method indeed obtains very close results to SVD.

Another interesting point in these figures is the black horizontal line, which stands for the compression precision $\epsilon = 10^{-8}$. That is, observe that our stopping criterion (see Section 2) compares the y-axis values in Figure 12 to the ϵ value to determine the approximation rank. Therefore, whenever a residual value reaches this black line in the figures, we know that the compression rank is the corresponding index. Therefore, through these stability figures, we observe that for the *Z-shape-short* case, all the methods get much larger compression ranks than $r_G = 100$. Thus, this case is challenging for our numerical stopping criterion. However, for all the other cases, all the methods reach close approximation ranks to the generation one.

In Figure 13, the same information as in Figure 12 is shown. However, here the relative residual with respect to the SVDF is presented to better observe the stability compared to the SVD method.

The main observation in Figure 13 is that QRCP, RQRCP and TQRCP can have a relative error around 2.5× larger than the one of the SVD around the index r_G. However, this relative error is very close to SVDF for the

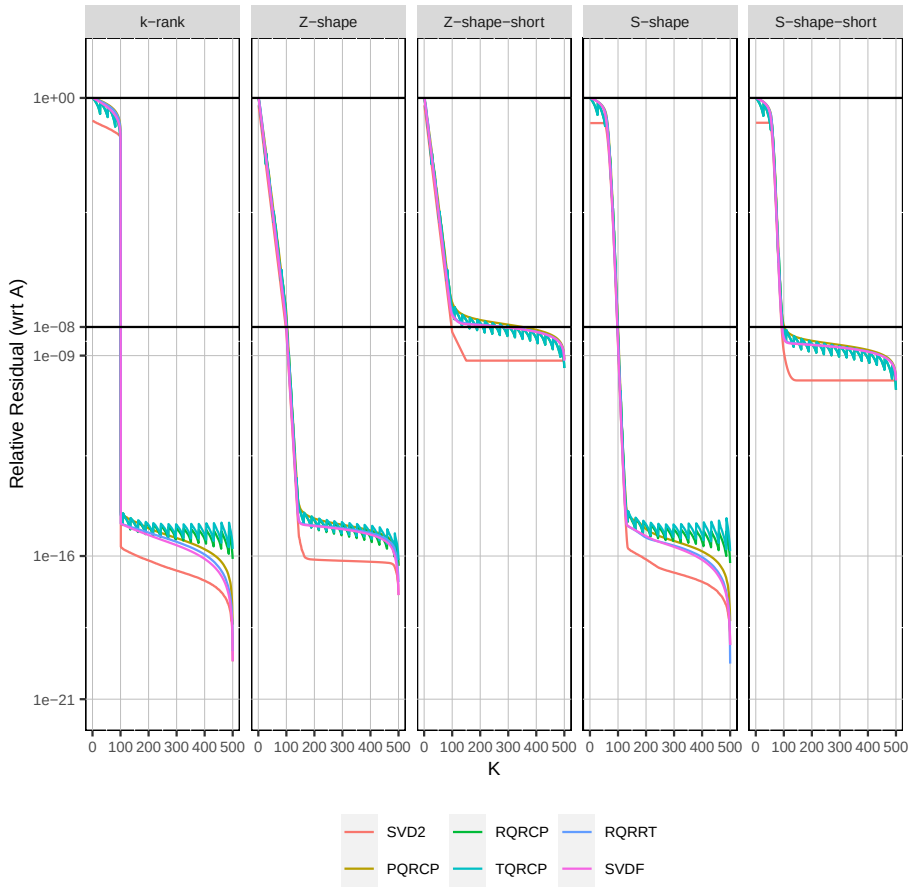


Figure 12: In the stability figures, the relative residual value with respect to the initial matrix, $\frac{||A-U_K V_K^T||_F}{||A||_F}$, at each index K is illustrated.

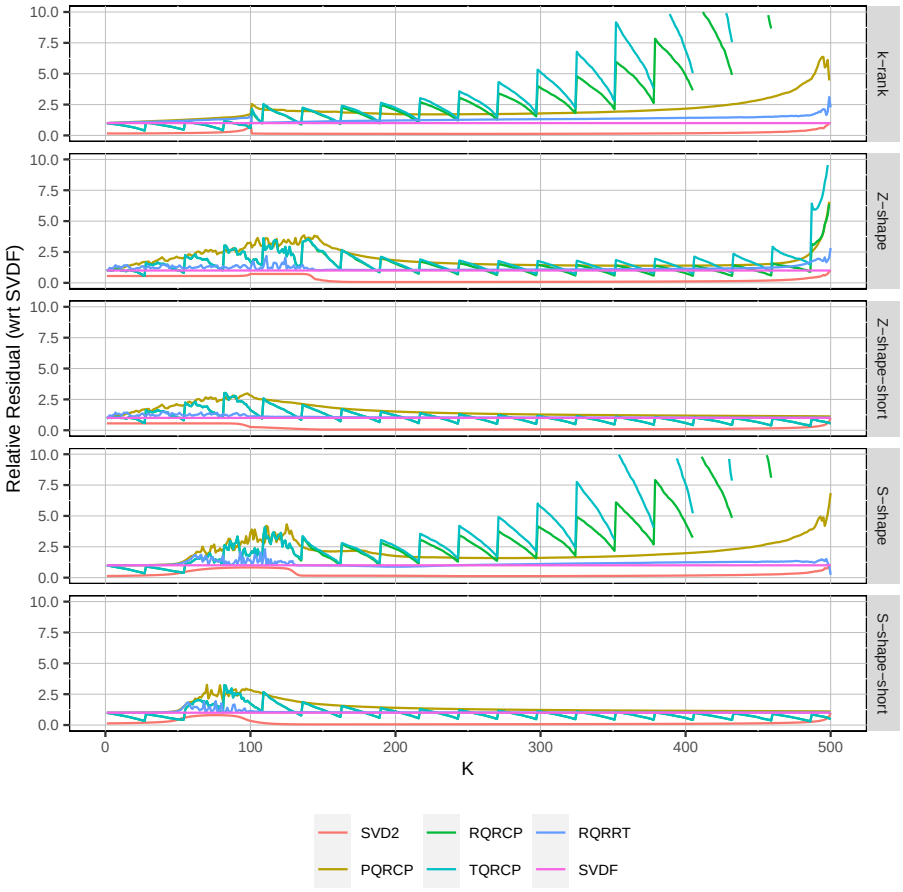


Figure 13: In the relative stability graphs, the relative residual value with respect to the SVDF, $\frac{||A-U_K V_K^T||_F}{||A-U_K V_K^T||_F^{SVDF}}$, at each index K is illustrated.

rotational QR variant, which confirms that indeed RQRRT provides closer quality approximations to SVD.

Note that for the *k-rank*, *Z-shape* and *S-shape* test cases, the last values are around the machine precision. Therefore, the extreme relative ratios at large indices are not representative, so we dropped them in the figure.

7.3 Performance

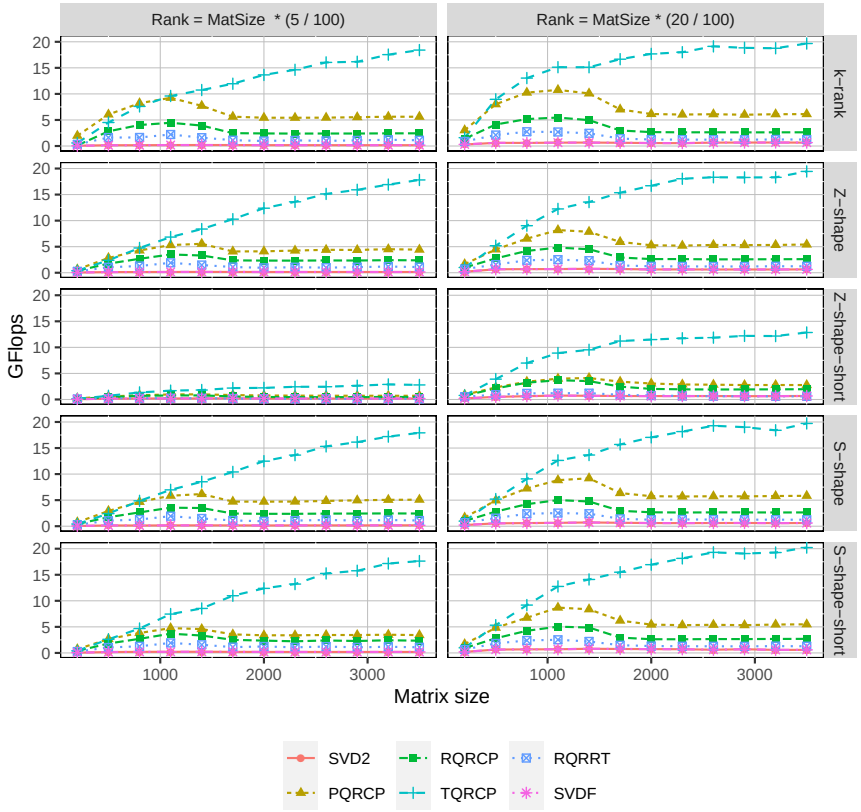


Figure 14: In the performance figures, $GFlops = \frac{MinGFlop}{t}$ is observed for different matrix sizes and r_G values. In this figure, each row illustrates a different test case, whereas each column represents a different generation rank. Here, we applied the stopping criterion within the compression methods.

In this section, we observe the performance results of the generated matrices that are explained in Section 7.1. In Figure 14, each test case is observed with different matrix sizes and different generation ranks.

In the figure, every grid row shows a different test case, while the columns stand for different generation ranks. The y-axis of figures stands for $GFlops = \frac{MinGFlop}{t}$. Here, t (in seconds) represents the time to compress the matrix with the compression rank r for each method, whereas $MinGFlop$ stands for the minimum number of GFlop to compress the matrix of size $n \times n$ and rank r. We compute it as

$$MinGFlop = GFlop_{GEQRF}(n,r) + GFlop_{UNMQR}(n,n-r,r) + GFlop_{UNGQR}(n,r,r).$$

Here, $GEQRF$, $UNMQR$ and $UNGQR$ names are taken from LAPACK. The $GFlop_{GEQRF}(n,r)$ represents the number of flops to perform a QR factorization on a $n \times r$ matrix. $GFlop_{UNMQR}(n,n-r,r)$ stands for the flops to apply the Q to the V part of size $r \times n$ of the approximation. Finally, $GFlop_{UNGQR}(n,r,r)$ represents the number of flops required to generate the final $n \times r$ matrix U from the Q matrix in the Householder form.

As seen in Figure 14, TQRCP has closer flops to the minimal flops and has always the best performance (up to 20 GFlops) for large matrix sizes, which proves the effectiveness of the left-looking approach in sequential environments.

On average, QRCP reaches the second best performance in the figures, as expected, while it has even the best performance for small matrix sizes. This is an interesting result since the randomization technique in RQRCP was introduced to increase the performance of QRCP by reducing the slow memory passes. However, since the tests run in sequential with matrix dimensions lower than 3500 and RQRCP introduces more computational complexity compared to QRCP, it is not an unexpected result. If the experiments were performed in a parallel environment for larger matrices, RQRCP would have better performance than QRCP.

As expected, RQRRT has the worst performance compared to the other QR variants because of its high complexity. However, SVD has the worst performance overall since no matter which stopping criterion is used, this method should apply the full decomposition to be able to determine the compression rank.

The important point in the figures is that the QRCP method has a better performance than TQRCP for small matrix sizes, as mentioned before. However, for the larger sizes TQRCP is the best. Therefore, tuning the methods according to the matrix size is also important since there is not an absolute best method.

Another important point is that in the stability section, we have seen that for the test cases, all the compression ranks are close to the generation rank, except for the *Z-shape-short* case. This causes a significant performance reduction for all the methods through this test case. Especially, for the small ranks, the performance loss is very critical for the *Z-shape-short* case. However, we can also observe that even if the performance is reduced significantly, the performance order of the methods is still the same as the other test cases, which shows a consistency among all the figures.

7.4 Compression ranks

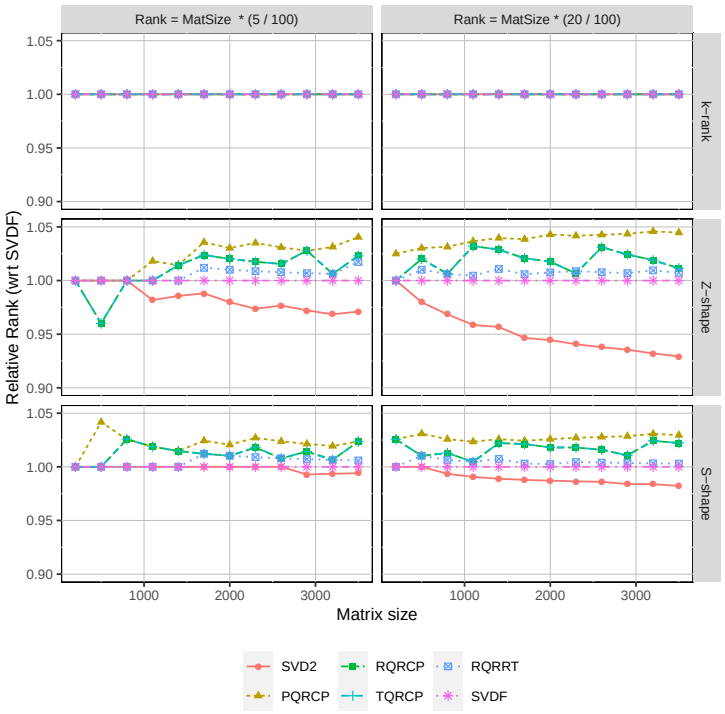


Figure 15: In the relative rank graphs, the relative rank with respect to SVDF, $\frac{r}{r_{SVDF}}$, for different matrix sizes and r_G values is observed. In this figure, each row illustrates a different test case, whereas each column represents a different generation rank. Here, the stopping criterion is applied in the compression methods.

In this section, we observe the compression rank comparison of the methods for the five generated matrices we introduced. In Figure 15 and Figure 16, the relative rank with respect to SVDF, $\frac{r}{r_{SVDF}}$, is observed for different matrix sizes.

As expected, all the methods reach to the generation rank for the *k-rank* case. On the other hand, for the *Z-shape-short* and *S-shape-short* cases, the rank differences are higher since these cases have large trailing matrix norms after the generation rank as observed in Figures 10 and 11, respectively. That is, they are more challenging for our numerical stopping criterion, which uses the Frobenius norm of the trailing matrix. As these test cases result in more rank differences than other cases, we show them separately in Figure 16.

In the figures, the lowest possible ranks are reached by SVD2 since it uses a more strict stopping criterion than the ones that adopt the Frobenius norm of

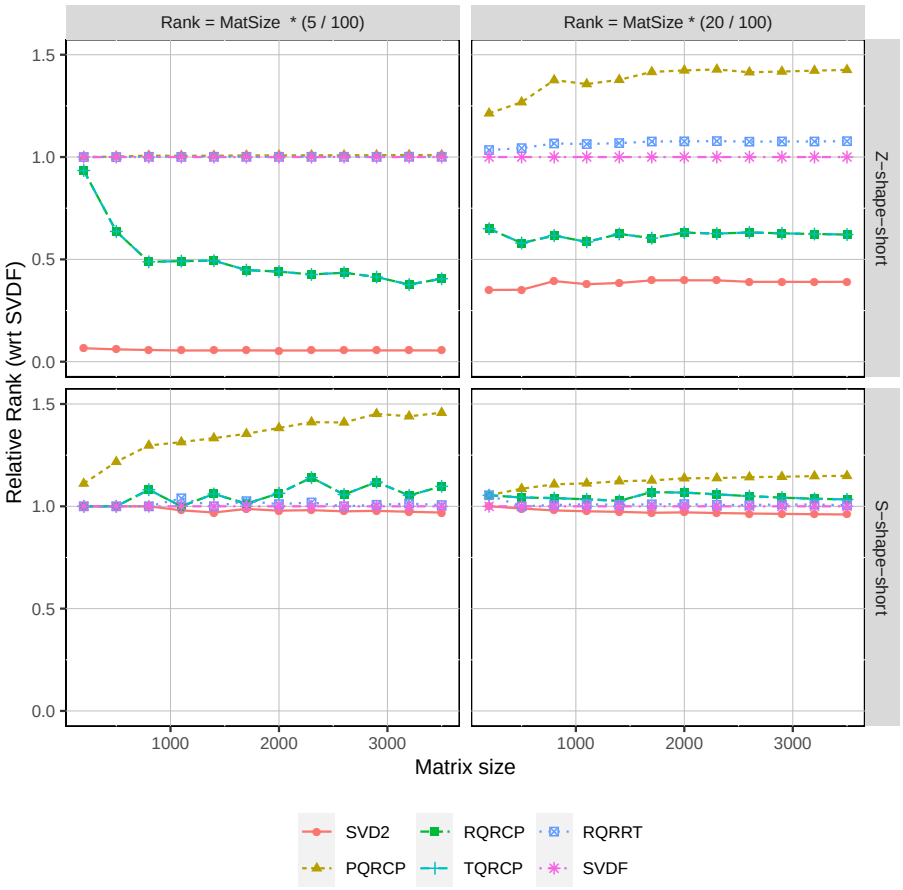


Figure 16: In the relative rank graphs, the relative rank with respect to SVDF, $\frac{r}{r_{SVDF}}$, for different matrix sizes and r_G values is observed. In this figure, each row illustrates a different test case, whereas each column represents a different generation rank. Here, we applied the stopping criterion within the compression methods.

the trailing matrix.

As the rank differences are higher for the tricky case *Z-shape-short*, we can easily observe that RQRCP and TQRCP reach closer results to SVD2 compared to all other methods (including SVDF). This is caused by the higher residual norm differences within the panels for RQRCP and TQRCP as seen in Figure 12. The high residual norm differences can be related with the stopping criterion used in RQRCP and TQRCP, which is based on a sample matrix, whereas other methods use the original matrix norms to terminate. In addition, this difference can be related to the accumulated round-off effects of small numbers. It is also interesting to observe that the round-off effects of small numbers lead RQRRT to not always reach to the closest results to SVDF. However, as we will observe in Figure 17, all the methods provide a good accuracy since we work with fixed precision.

7.5 Accuracy

In this section, we observe the accuracy of the compression methods through our numerical stopping criterion. In Figure 17, the error, $\frac{||A-U_r V_r^T||_F}{\epsilon||A||_F}$, for different matrix sizes and r_G values is observed.

As seen in the figure, since for the *k-rank* case it is very trivial to reach to the generation rank, there is no observable error difference between the compression methods.

In the previous section, we observed that RQRCP and TQRCP tend to have smaller ranks than QRCP. Therefore, in Figure 17, they tend to have more data in the trailing matrix after the compression. Similarly, since SVD2 finds the smallest ranks in general, the data in the trailing matrix (error) tends to be more than the other methods.

When tackling very small numbers through the stopping criterion within RQRRT, the resulting round-off errors can slightly affect the approximation rank and error. As observed in the figure, this does not even cause one digit accuracy changes for this method, since our numerical stopping criterion guarantees a good accuracy. Therefore, we do not observe a noteworthy accuracy difference between the compression methods.

8 Experiments on real-life case matrices

In this section, we compare the compression methods through 810 real-life case matrices. These matrices are downloaded from https://sparse.tamu.edu/ with the criteria that their row and column dimensions are less than 2000. Among them, there are 802 matrices with real data and 8 matrices with complex data. There are both square and rectangular matrices. That is why, in this section, the matrix dimensions are noted as $m \times n$ instead of $n \times n$.

Since our stopping criterion ensures a good accuracy, as we observed on the generated matrices, we only present the performance and obtained rank results in this section. In addition, we only provide the SVDF method results (not

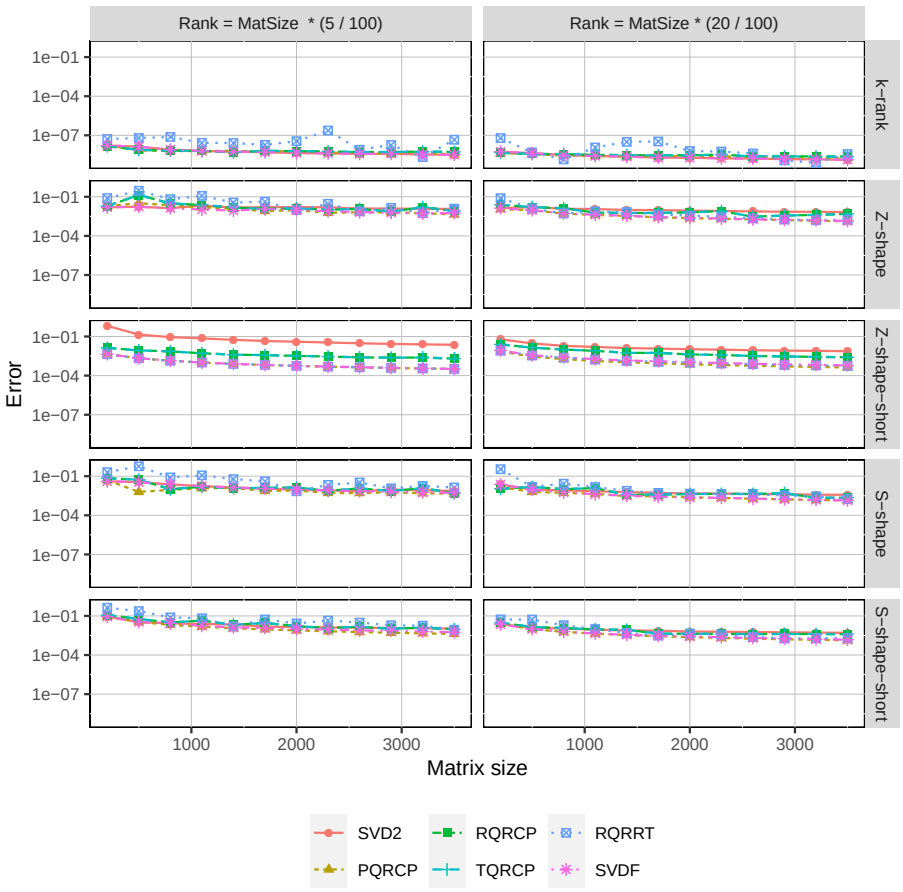


Figure 17: In the accuracy figures, the error, $\frac{||A-U_r V_r^T||_F}{\epsilon||A||_F}$, for different matrix sizes and r_G values is observed. Here, we applied the stopping criterion within the compression methods.

SVD2) as a reference, since our main point is to compare the QR variants, which use the Frobenius norm of the residuals in the stopping criterion. Note that, in this section, we use the same experimental settings as in Section 7.

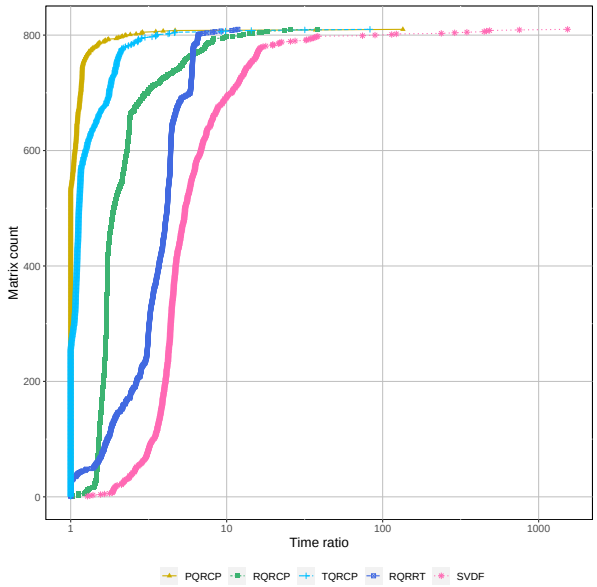


Figure 18: Time profiles of the real-life case test matrices. The x-axis ratios are computed as $\frac{t_{current}}{t_{min}}$ for each method, while the y-axis stands for each matrix as a number from 1 to 810. Here, the x-axis is in logarithmic scale for the sake of clarity of the figure.

In Figure 18, we observe the time ratio, $\frac{t_{current}}{t_{min}}$, of each method for all matrices. In this time profile, a method is better on average, if its curve is closer to the line $y = 1$.

As expected, SVDF is the slowest method for all the cases, where RQRRT is the slowest QR variant for most of the matrices. For some very small cases, we cannot observe the expected method performances, since advantages of the methods cannot be exploited at these levels. It is interesting to observe that QRCP is the fastest method for the majority of matrices, while TQRCP is the second fastest method. The reason for this is that the real-life case matrices we used are mostly small matrices. In Section 7.3, we observed that TQRCP is advantageous for large matrices, so this is an expected result. RQRCP never runs as the fastest method in these experiments. As we do not use large enough matrices to benefit from this method, as well as we work in a sequential environment, this method is not promising in our context.

In Figure 19, we observe the compression rank ratio, $\frac{r_{current}}{r_{min}}$, of each method for some of the 810 matrices. This figure is a zoomed version of the original one in a way that it only stands for 160 matrices, which have observable difference

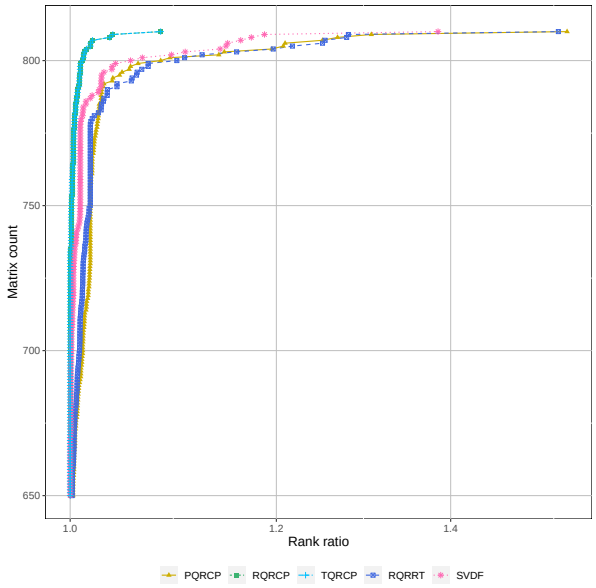


Figure 19: Zoomed obtained rank profiles of the real-life case test matrices. The x-axis ratios are computed as $\frac{r_{current}}{r_{min}}$ for each method, while y-axis represents each matrix as a number from 650 to 810. The y-axis values (from 0 to 649) are omitted to zoom the figure since the methods get very close results at this interval.

in the figure. That is, it shows the top-most part (y-axis from 650 to 810) of the original figure. The previous y-axis values (from 0 to 649) are omitted in the figure as the methods obtain similar results at this interval.

In Figure 19, the values of RQRCP and TQRCP intersect, so RQRCP does not show up. Both of these methods find the lowest compression ranks for 734 cases. Supporting the results in Section 7.4, even SVD finds larger ranks compared to RQRCP and TQRCP for 140 matrices because of the sample matrix based stopping criterion. Thus, RQRCP and TQRCP seem to be the most promising compression methods to find the lowest ranks possible. Considering also the performance profiles results, TQRCP seems to be better than RQRCP in sequential environments (for the matrices with dimensions less than 2000). That is, TQRCP does not only obtain the lowest ranks possible, but this left-looking method also compresses all the matrices faster than RQRCP (see Figure 18).

We can compare the quality of using a rotation matrix or a permutation matrix through QRCP and RQRRT. As both use a stopping criterion through the original matrix, the figure proves that RQRRT slightly improves the obtained ranks by getting closer to SVD. However, taking the performance results into account, we do not get good enough trade-off between rank and performance, in our context, through RQRRT. QRCP seems to be the least promising method to find the minimal rank. However, it is worth noting that the rank differences between the methods are not important for most of the matrices. Thus, QRCP is still promising for the small matrices, thanks to its performance.

To conclude, as good accuracy is ensured by our stopping criterion, we need to mostly focus on the obtained rank and performance results. Then, QRCP and TQRCP seem to be the most promising compression methods for us. Tuning between these methods according to the matrix size and the compressibility of the matrix (if known) can contribute to minimize the total time of the compression process, as well as the compression rank.

9 Conclusion

In this report, we study and compare four different stable QR decomposition techniques, which aim to be fast alternatives to SVD when compressing dense matrices.

The most basic QR version is a panel-wise right-looking QRCP method. The second one is the randomized version of the QRCP method, so it is called randomized QRCP (RQRCP). It aims to avoid the costly data movements for large matrices when computing the column norms during the pivoting. Here, we adopt a version, where the resampling is avoided thanks to a cheaper sample matrix update formula. The third method is the left-looking version of the RQRCP method, which is called truncated randomized QRCP (TQRCP). This method reduce the computational complexity by eliminating expensive full trailing matrix updates at each iteration. Therefore, in a sequential environment, this method is the cheapest one especially for large matrices with low ranks. The last method we study is a rotational version of RQRCP. Here, we use a rotation

matrix instead of column pivoting, so that we can better approximate SVD.

All the studied methods in this report are existing methods. However, we implement them in a similar fashion for a better comparison. In addition, we determine the compression ranks through our numerical stopping criterion at a given precision. This stopping criterion was never used for two of the studied methods, namely TQRCP and RQRRT. We conduct all the experiments in the framework of the PaStiX sparse direct solver, and as a matter of interest, we observe the methods in a sequential environment.

In our experiments, we show that all the methods are accurate thanks to the nature of our numerical stopping criterion. Thus, we mostly focus on the performance and obtained rank results.

We observe that the obtained ranks for the RQRCP and TQRCP methods might be even better than SVDF thanks to the sample matrix based stopping criterion, while still satisfying the accuracy requirement. As TQRCP method also outperforms other methods for larger matrices, it is the best method for these matrix sizes in a sequential environment. Bear in mind that all the compression methods obtain close ranks for majority of the matrices. Therefore, performance results determine the best method for small matrices, which is QRCP. Then, tuning between QRCP and TQRCP is necessary according to the block sizes we use in our solver.

The QRCP method has more performance than RQRCP even for the largest matrices we used. Therefore, RQRCP is not an advantageous method for us. Nonetheless, note that it is still a promising method in other contexts. Here, we could not benefit from this compression method as all the experiments are in sequential and our matrix sizes might not be large enough to exploit it.

We prove the efficiency of the QRCP methods on obtaining small enough ranks. Thus, we do not need to adopt more expensive approaches (like RQRRT) in our context to reach closer results to SVD. For this reason, our further studies should mainly target to improve the performance, rather than the obtained ranks, compared to the QR methods we observed.

References

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J.W., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., et al.: LAPACK Users' Guide. SIAM, Philadelphia, PA, 1992 (2007) 10

[2] Bebendorf, M., Rjasanow, S.: Adaptive low-rank approximation of collocation matrices. Computing **70**, 1–24 (2003). https://doi.org/https://doi.org/10.1007/s00607-002-1469-6 6

[3] Berry, M.W., Pulatova, S.A., Stewart, G.W.: Algorithm 844: Computing sparse reduced-rank approximations to sparse matrices. ACM Trans. Math. Softw. **31**(2), 252–269 (jun 2005).

https://doi.org/10.1145/1067967.1067972, https://doi.org/10.1145/1067967.1067972 6

[4] Börm, S., Grasedyck, L.: Hybrid cross approximation of integral operators. Numerische Mathematik **101**, 221–249 (08 2005). https://doi.org/10.1007/s00211-005-0618-1 6

[5] Chan, T.F.: Rank revealing qr factorizations. Linear Algebra and its Applications **88-89**, 67 – 82 (1987). https://doi.org/https://doi.org/10.1016/0024-3795(87)90103-0, http://www.sciencedirect.com/science/article/pii/0024379587901030 6

[6] Devore, J.L.: Probability and Statistics for Engineering and Science - 8th edition. Brooks/Cole Publishing Co. (2012) 18

[7] Dongarra, J.J., Du Croz, J., Hammarling, S., Duff, I.S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**(1), 1–17 (1990) 7

[8] Drmac, Z., Bujanovic, Z.: On the failure of rank-revealing qr factorization software – a case study. ACM Trans. Math. Softw. **35** (07 2008). https://doi.org/10.1145/1377612.1377616 14

[9] Duersch, J.A., Gu, M.: True blas-3 performance qrcp using random sampling. arXiv: Numerical Analysis (2015) 6, 11, 17, 18, 19, 20

[10] Eckart, C., Young, G.: The approximation of one matrix by another of lower rank. Psychometrika **1**(3), 211–218 (1936) 6

[11] Golub, G.H., van Loan, C.F.: Matrix computations. JHU Press (2013) 6, 7

[12] Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review **53**(2), 217–288 (2011) 6, 12

[13] Liberty, E., Woolfe, F., Martinsson, P.G., Rokhlin, V., Tygert, M.: Randomized algorithms for the low-rank approximation of matrices. Proceedings of the National Academy of Sciences **104**(51), 20167–20172 (2007). https://doi.org/10.1073/pnas.0709640104, https://www.pnas.org/content/104/51/20167 6

[14] Martinsson, P.G.: Blocked rank-revealing qr factorizations: How randomized sampling can be used to avoid single-vector pivoting. arXiv preprint arXiv:1505.08115 (2015) 11, 21, 22, 26

[15] Martinsson, P.G., Quintana-Orti, G., Heavner, N.: randutv: A blocked randomized algorithm for computing a rank-revealing utv factorization. ACM Transactions on Mathematical Software **45** (03 2017). https://doi.org/10.1145/3242670 6

[16] Martinsson, P.G., Quintana-Orti, G., Heavner, N., van de Geijn, R.: Householder qr factorization with randomization for column pivoting (hqrrp). SIAM Journal on Scientific Computing **39**(2), C96–C115 (2017) 11, 17

[17] Martinsson, P.G., Tygert, M.: A randomized algorithm for the decomposition of matrices. Applied and Computational Harmonic Analysis **30**, 47–68 (01 2011). https://doi.org/10.1016/j.acha.2010.02.003 6

[18] Martinsson, P.G., Voronin, S.: A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. SIAM Journal on Scientific Computing **38**(5), S485–S507 (2016) 11

[19] Mary, T.: Block Low-Rank multifrontal solvers: complexity, performance, and scalability. Ph.D. thesis, Toulouse University, Toulouse, France (Nov 2017) 14

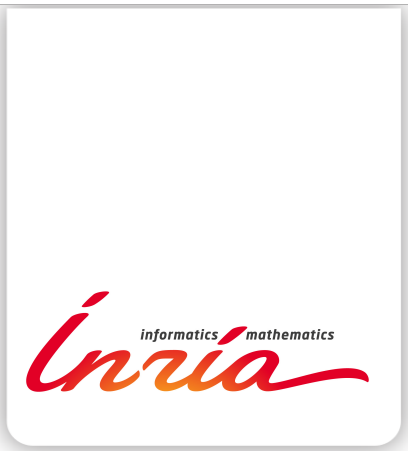
[20] Pichon, G.: On the use of low-rank arithmetic to reduce the complexity of parallel sparse linear solvers based on direct factorization technique. Ph.D. thesis, Universite de Bordeaux, Bordeaux, France (2018) 4, 6

[21] Pichon, G., Darve, E., Faverge, M., Ramet, P., Roman, J.: Sparse supernodal solver using block low-rank compression: Design, performance and analysis. International Journal of Computational Science and Engineering **27**, 255 – 270 (Jul 2018) 26

[22] Quintana-Ortí, G., Sun, X., Bischof, C.H.: A blas-3 version of the qr factorization with column pivoting. SIAM Journal on Scientific Computing **19**(5), 1486–1494 (1998). https://doi.org/10.1137/S1064827595296732, https://doi.org/10.1137/S1064827595296732 8, 9

[23] Trefethen, L., Bau, D.: Numerical linear algebra. SIAM (1997) 7

[24] Xiao, J., Gu, M., Langou, J.: Fast parallel randomized qr with column pivoting algorithms for reliable low-rank matrix approximations. In: 2017 IEEE 24th International Conference on High Performance Computing (HiPC). pp. 233–242. IEEE (2017) 11, 17



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399