CROQuant: Complex Rank-One Quantization Algorithm

Maël CHAUMETTE[1] Rémi GRIBONVAL[1] Elisa RICCIETTI[2]

[1]Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France

[2]ENS de Lyon, CNRS, Inria, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France

Résumé – Cet article présente un algorithme de quantification pour les matrices de rang un à valeurs complexes, qui exploite les invariances par remise à l'échelle du problème pour obtenir un meilleur résultat que la quantification au plus proche voisin. En s'appuyant sur cet algorithme on propose aussi une approche pour la quantification des matrices creuses à structure papillon à valeurs complexes apparaissant par exemple dans la transformée de Fourier rapide. Comparé à la quantification au plus proche voisin élément par élément, on obtient ainsi une réduction de 30% du nombre de bits nécessaires pour une précision donnée sur les matrices papillons, tout en maintenant une complexité polynomiale en la dimension des matrices.

Abstract – This paper presents a quantization algorithm for complex-valued rank-one matrices, which exploits rescaling-invariances of the problem to obtain better results than round-to-nearest strategy. This algorithm can be used as a building block for an heuristic stategy to quantize complex-valued butterfly-structured sparse matrices appearing for example in the fast Fourier transform. Compared to element-wise round-to-nearest quantization we reduce by 30% the number of bits for a given precision on butterfly matrices, while maintaining a polynomial time complexity in the dimension of the matrices.

1 Introduction

Quantization is a fundamental issue in computer science and machine learning. With the ever-increasing size of models and data, the need to reduce memory by lowering numerical precision while maintaining acceptable performance is becoming crucial. In particular, quantization reduces the memory required to store matrices and speeds up the computations. These advantages are widely needed for deep neural networks, which are mainly composed of huge, dense matrices.

Rank-one matrices play a central role in many compression problems in machine learning and signal processing. Recently, an optimal quantization approach for *real-valued* rank-one matrices, which exploits rescaling-invariances to minimize the quantization error, was introduced in [7]. In this work, we extend these results to the case of *complex-valued* matrices, an essential generalization to treat several applications in signal processing, such as the Fast Fourier Transforms (FFT). Indeed, the FFT involves a butterly structure, corresponding to a product of structured sparse matrices often used to factorize dense matrices, such as Hadamard and discrete cosine transform matrices [8], to speed up matrix-vector products and reduce memory space [10]. Quantization of such matrices further amplifies these computational gains. Their particular structure allows quantization to be heuristically decomposed into a succession of rank-one quantization problems exploiting our approach.

In Section 2, we formalize the quantization problem for complex rank-one matrices and explain why this problem cannot be solved simply with the optimal quantization algorithm applied to the real and imaginary parts.

We propose in Section 3 an algorithm adapted to this complex setting. We show that the properties of invariance by rescaling persist in the complex setting, and allow for more accurate quantization than a naive nearest-neighbor rounding approach. Our algorithm depends on a parameter d_{m}, which controls the accuracy and affects the computational time (the higher d_{m} the higher the accuracy but the longer the computational time). However, we empirically show that small values of the parameter give good results. This algorithm is also much faster than the brute-force method of testing all possible combinations for each element of the rank-one matrix: the latter provides an algorithm with exponential time complexity in t, the number of significant bits, and in the dimensions m, n of the matrix; whereas our algorithm has a time complexity of $\mathcal{O}(nm\min(n,m)d_{\mathrm{m}}^2 2^{2t})$, which is simply polynomial in the dimension of the rank-one matrix. The exponential dependence on t is tractable in this context because we are interested in small values of t, typically $t = 4$ or 6 for applications in modern float formats typically used in machine learning.

In Section 4, we discuss the quantization of complex butterfly matrices that play a central role in the FFT. Our algorithm provides quantization that is also more efficient than the naive rounding approach. At a given accuracy, our algorithm will need 30% fewer bits than the naive rounding method.

Notations. $\mathbb{F}_t$ is a set of floating-point numbers with t-bit of significand, as in [7]. We denote vectors in lowercase boldface ($\boldsymbol{x}$) and matrices in uppercase boldface letters ($\boldsymbol{X}$). $\|\cdot\|$ is the Frobenius norm. Re, Im denote the real and imaginary parts of a complex number, which we quantize separately. We therefore define $\mathbb{CF}_t := \mathbb{F}_t + i\mathbb{F}_t$.

2 Problem formulation and baselines

Given $\boldsymbol{x} \in \mathbb{C}^m$ and $\boldsymbol{y} \in \mathbb{C}^n$, we formulate the quantization problem as the following minimization problem:

$$\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \arg\min_{\hat{\boldsymbol{x}} \in \mathbb{CF}_t^m, \hat{\boldsymbol{y}} \in \mathbb{CF}_t^n} \underbrace{\|\boldsymbol{x}\boldsymbol{y}^H - \hat{\boldsymbol{x}}\hat{\boldsymbol{y}}^H\|^2}_{C_{\boldsymbol{x},\boldsymbol{y}}(\boldsymbol{x},\boldsymbol{y})}. \tag{1}$$

This project was supported by the SHARP project of the PEPR-IA (ANR-23-PEIA-0008, financed by France 2030), and the project ANR MEPHISTO ANR-24-CE23-7039. The authors would like to thank the Centre Blaise Pascal at ENS de Lyon (Lyon, France) for computing facilities. The platform uses the SIDUS solution [12] developed by Emmanuel Quemener.

Like in [7], the naive approach, called round-to-nearest (RTN), involves mapping each element of $\boldsymbol{x}$ and $\boldsymbol{y}$ to its nearest neighbor in $\mathbb{CF}_t$. We define the function $\mathrm{round}(\cdot)$ that maps the real and the imaginary parts of a complex number to their nearest neighbor in $\mathbb{F}_t$. For a vector, the round function is applied element-wise. This method does not take into account that the problem is invariant by rescaling: $\forall \lambda \in \mathbb{C}^*, \boldsymbol{x}\boldsymbol{y}^H = (\lambda\boldsymbol{x})(\frac{1}{\lambda}\boldsymbol{y})^H$. Another possible method is to apply separately the optimal algorithm of [7] to the real and imaginary parts of $\boldsymbol{x}$ and $\boldsymbol{y}$. However, this method is not optimal either, since it omits cross-terms in the product of the complex numbers. We therefore need to find another method to obtain optimal quantization of complex rank-one matrices.

3 Proposed quantization algorithm

In this section, after characterizing the optimal solution to the quantization problem (1) we build an algorithm providing an efficient approximate solution, discuss its complexity, and finally present its performance against the RTN method.

Characterizing the optimal solution. The following lemma is a key result to address (1).

Lemma 1. *Given $\boldsymbol{x} \in \mathbb{C}^m$, $\boldsymbol{y} \in \mathbb{C}^n$ and $t \geq 1$, it holds*

$$\inf_{\hat{\boldsymbol{x}} \in \mathbb{CF}_t^m, \hat{\boldsymbol{y}} \in \mathbb{CF}_t^n} \|\boldsymbol{x}\boldsymbol{y}^H - \hat{\boldsymbol{x}}\hat{\boldsymbol{y}}^H\|^2 = \inf_{\lambda \in \mathbb{C}} f(\lambda) \tag{2}$$

$$\textit{with } f(\lambda) := \max_{\hat{\boldsymbol{x}} \in \mathrm{round}(\lambda \boldsymbol{x})} \|\boldsymbol{x}\boldsymbol{y}^H - \hat{\boldsymbol{x}}\,\mathrm{round}(\mu(\hat{\boldsymbol{x}})\boldsymbol{y})^H\|^2 \tag{3}$$

$$\textit{where } \mu(\hat{\boldsymbol{x}}) := \begin{cases} \frac{\langle \boldsymbol{x}, \hat{\boldsymbol{x}} \rangle}{\|\hat{\boldsymbol{x}}\|^2} & \text{if } \hat{\boldsymbol{x}} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The lemma and its proof are the complex-valued analog of [7, Lemma 4.2], that was shown to allow to reduce the $4mn$-variable optimization problem (1) to a one-variable problem: to find an optimal λ^* of f defined in (3), assuming it exists.

Minimizing f: restricting to compact domain. It is easy to show that $f(2\lambda) = f(i\lambda) = f(\lambda)$ for every $\lambda \in \mathbb{C}$, so the study of the function can be limited to $\mathbb{C}^{++}_{[1,2]} := \{z \in \mathbb{C}, |z| \in [1,2], \arg(z) \in [0, \frac{\pi}{2}]\}$ or any compact domain $\Omega \subset \mathbb{C}$ generating a tiling of the complex plane via dilations and rotation (i.e. with the sets $i^k 2^j \Omega$, $0 \leq k < 3$, $j \in \mathbb{Z}$).

Using the piecewise constant nature of f. Furthermore, since the function $\lambda \mapsto \mathrm{round}(\lambda\boldsymbol{x})$ is piecewise constant, f is also piecewise constant, so finding a minimizer on $\mathbb{C}^{++}_{[1,2]}$ amounts to finding on which of the corresponding pieces f attains the smallest value. The simplest way to individuate these pieces is to study the *breakpoints* of the function $\lambda \in \mathbb{C} \mapsto \mathrm{round}(\lambda\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{C}^m$. The breakpoints correspond to λ values for which there is at least one $\boldsymbol{x}_i$ such that $\mathrm{round}(\lambda\boldsymbol{x}_i)$ corresponds to a tie in the choice of the nearest neighbor for the real or the imaginary part.

Characterization of *breaklines*. The following result describes these breakpoints.

Lemma 2. *Consider $\boldsymbol{x} \in \mathbb{C}^m$. For each entry $\boldsymbol{x}_k := a + ib$, the breakpoints of $\lambda \in \mathbb{C} \mapsto \mathrm{round}(\lambda\boldsymbol{x}_k)$ are straight lines in the complex plane with equations*

$$\begin{cases} a\,\mathrm{Im}(\lambda) = -b\,\mathrm{Re}(\lambda) + s(k + \frac{1}{2})2^{-d-t} \\ b\,\mathrm{Im}(\lambda) = a\,\mathrm{Re}(\lambda) + s(k + \frac{1}{2})2^{-d-t} \end{cases} \tag{5}$$
$$\forall k \in [\![2^{t-1}, 2^t - 1]\!],\ \forall s \in \{-1, +1\},\ \forall d \in \mathbb{Z}$$

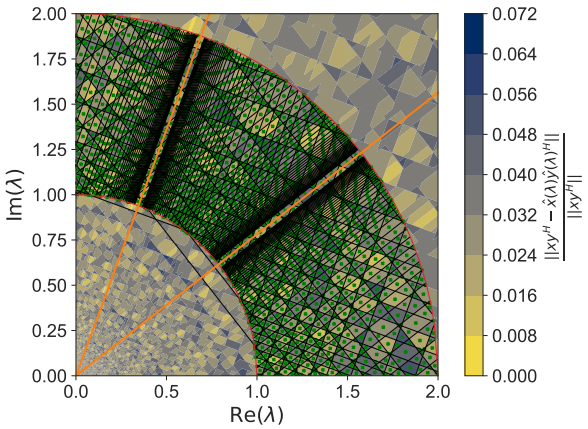

Figure 1 – Function $\lambda \in \mathbb{C} \mapsto \frac{\|\boldsymbol{x}\boldsymbol{y}^H - \hat{\boldsymbol{x}}(\lambda)\hat{\boldsymbol{y}}(\lambda)^H\|}{\|\boldsymbol{x}\boldsymbol{y}^H\|}$ with $\hat{\boldsymbol{x}}(\lambda) := \mathrm{round}(\lambda\boldsymbol{x})$, $\hat{\boldsymbol{y}}(\lambda) := \mathrm{round}(\mu(\hat{\boldsymbol{x}}(\lambda))\boldsymbol{y})$ and $\mu(\hat{\boldsymbol{x}}(\lambda))$ defined in (4). Breaklines of the function $\lambda \in \mathbb{C}^{++}_{[1,2]} \mapsto \mathrm{round}(\lambda\boldsymbol{x})$ in black (corresponding to $d \leq 4$ in Lemma 2), centroids in green, accumulation lines in orange. The search space $\mathbb{C}^{++}_{[1,2]}$ is outlined in red. Here, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^2$ were drawn with real and imaginary parts following a uniform distribution in $[0, 1]$ and $t = 4$.

The proof is an adaptation of the proof in [7, Lemma 5.1]. We will call *breaklines* the lines defined by (5). There are infinitely many such breaklines approximating "accumulation lines" defined by taking $d \to +\infty$, with equations $a\,\mathrm{Im}(\lambda) + b\,\mathrm{Re}(\lambda) = 0$ (resp. $b\,\mathrm{Im}(\lambda) - a\,\mathrm{Re}(\lambda) = 0$). Figure 1 shows the typical shape of f on $\mathbb{C}^{++}_{[1,2]}$ with its breaklines (corresponding to $d \leq 4$), and accumulation lines displayed in orange.

Construction of the algorithm. As in the real-valued case, since f is piecewise constant, minimizing it amounts to finding on which of the pieces (delimited by its breaklines) it attains the smallest value. However, contrary from the real-valued case where it suffices to visit a finite number of breakpoints, here in the complex-valued case there are infinitely many pieces even in the bounded region $\mathbb{C}^{++}_{[1,2]}$. It is therefore not possible to explicitly evaluate f in all these pieces in order to find $\hat{\boldsymbol{x}}^*$ and $\hat{\boldsymbol{y}}^*$. For this reason we define a parameter d_{m}, and choose to consider just the breaklines corresponding to $d \leq d_{\mathrm{m}}$ in Lemma 2. This defines a finite number of pieces, which we can enumerate and iterate over using the python library *shapely*. Each of the resulting pieces has a centroid, and the collection of such centroids is denoted $\mathcal{C}_{d_{\mathrm{m}}}(\boldsymbol{x})$. Algorithm 1 describes the resulting near-minimization algorithm.

Role of the parameter. The larger the parameter d_{m}, the closer the resulting pair $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ is to being optimal, but this will have an impact on time complexity because this will increase the number of tested centroids. We will see in the experiment paragraph that $d_{\mathrm{m}} = 0$ is enough to get better results than the RTN approach.

Algorithm 1: Complex rank-one quantization algorithm

Data: $\boldsymbol{x} \in \mathbb{C}^m$, $\boldsymbol{y} \in \mathbb{C}^n$, $t \geq 1$, $d_{\mathrm{m}} \in \mathbb{Z}$.
Result: $\hat{\boldsymbol{x}}^* \in \mathbb{CF}_t^m$, $\hat{\boldsymbol{y}}^* \in \mathbb{CF}_t^n$, $\lambda^* \in \mathbb{C}$.

Initialize $\hat{\boldsymbol{x}}^* \leftarrow 0$, $\hat{\boldsymbol{y}}^* \leftarrow 0$;
if $\boldsymbol{x}, \boldsymbol{y} \neq 0$ **then**
  Build set of centroids $\mathcal{C}_{d_{\mathrm{m}}}(\boldsymbol{x})$ (Lemma 2 and call to `shapely`);
  for $\lambda \in \mathcal{C}_{d_{\mathrm{m}}}(\boldsymbol{x})$ **do**
    $\hat{\boldsymbol{x}} \leftarrow \mathrm{round}(\lambda \boldsymbol{x})$;
    $\mu \leftarrow \frac{\langle \boldsymbol{x}, \hat{\boldsymbol{x}} \rangle}{\|\hat{\boldsymbol{x}}\|^2}$;
    $\hat{\boldsymbol{y}} \leftarrow \mathrm{round}(\mu \boldsymbol{y})$;
    if $C_{\boldsymbol{x},\boldsymbol{y}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) < C_{\boldsymbol{x},\boldsymbol{y}}(\hat{\boldsymbol{x}}^*, \hat{\boldsymbol{y}}^*)$ **then**
      $\hat{\boldsymbol{x}}^* \leftarrow \hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}^* \leftarrow \hat{\boldsymbol{y}}$, $\lambda^* \leftarrow \lambda$;

Complexity. The complexity of the algorithm is determined in several stages. Firstly, the number of breaklines is bounded by $4md_{\mathrm{m}}2^{t-1}$. Then, according to [5], the maximum number of regions defined by k straight lines is $\frac{k(k+1)}{2} + 1$. In our case, we can conclude that there are $\mathcal{O}(m^2 d_{\mathrm{m}}^2 2^{2t})$ centroids. For each centroid, we need to calculate $C_{\boldsymbol{x},\boldsymbol{y}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$, which has a cost of $\mathcal{O}(m+n)$, by exploiting the same trick as in the real-valued case [7]. The total cost of the algorithm is therefore $\mathcal{O}\left((m+n)m^2 d_{\mathrm{m}}^2 2^{2t}\right)$. But, since $\boldsymbol{x}$ and $\boldsymbol{y}$ play symmetrical roles, we can swap their roles, which will result in a time complexity of $\mathcal{O}\left((m+n)n^2 d_{\mathrm{m}}^2 2^{2t}\right)$. Thus Algorithm 1 has a time complexity $\mathcal{O}(nm\min(n,m)d_{\mathrm{m}}^2 2^{2t})$.

Experiments and results. We consider 50 pairs $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{C}^{12} \times \mathbb{C}^{12}$, where the real and imaginary parts of each component follow a uniform distribution on $[0, 1]$. We set $t = 4$. Table 1 shows the average of the relative quantization error of our algorithm $\rho_{d_m} := \frac{\|\boldsymbol{x}\boldsymbol{y}^H - \hat{\boldsymbol{x}}^*(\hat{\boldsymbol{y}}^*)^H\|}{\|\boldsymbol{x}\boldsymbol{y}^H\|}$ and the computation time for different values of d_{m}. The computation time increases rapidly, while the relative variation between the error at $d_{\mathrm{m}} = 0$ and $d_{\mathrm{m}} = 6$ is less than 1%. This empirical proof encourages us to use $d_{\mathrm{m}} = 0$ in the rest of our experiments.

Table 1 – Average of the quantization error and the calculation time of Algorithm 1 over 50 pairs $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{C}^{12} \times \mathbb{C}^{12}$ with uniformly distributed real and imaginary parts in $[0, 1]$ for different values of d_{m} with $t = 4$.

	$d_{\mathrm{m}} = -2$	$d_{\mathrm{m}} = 0$	$d_{\mathrm{m}} = 2$	$d_{\mathrm{m}} = 6$
Error ($\times 10^{-2}$)	3.514	2.280	2.259	2.258
Time (s)	1.573e−3	6.711	60.09	172.9

We compare the quantization error of our algorithm, ρ_{d_m}, and the RTN strategy error $\rho_{\mathrm{rtn}} := \frac{\|\boldsymbol{x}\boldsymbol{y}^H - \mathrm{round}(\boldsymbol{x})\,\mathrm{round}(\boldsymbol{y})^H\|}{\|\boldsymbol{x}\boldsymbol{y}^H\|}$ in Figure 2 with the same setup as before but here we consider 100 pairs of $(\boldsymbol{x}, \boldsymbol{y})$. Algorithm 1 is more efficient than the RTN strategy especially for small n. Indeed, when $n = 4$, ρ_{d_m} is approximately 5 times smaller than ρ_{rtn}. Furthermore, as n increases, the point clouds become more concentrated.

We also compared Algorithm 1 to the naive application of the optimal algorithm for the real-valued case on the real and imaginary parts of $\boldsymbol{x}$ and $\boldsymbol{y}$. The results (not displayed) is that the naive method yields a quantization error approximately 10 times larger than ρ_{d_m}.

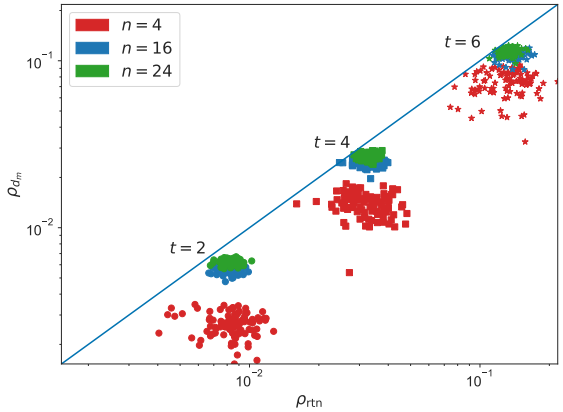

Figure 2 – Scatterplot of ρ_{d_m}, ρ_{rtn} for 100 pairs $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{C}^n \times \mathbb{C}^n$ with uniformly distributed real and imaginary parts in $[0, 1]$ for different values of n and t with $d_{\mathrm{m}} = 0$.

4 Application to butterfly quantization

We will now apply our algorithm to quantize butterfly matrices similar to those appearing in the FFT. The FFT accelerates the discrete Fourier transform (DFT) using the Cooley-Tukey algorithm [1]. The main idea is to use the divide-and-conquer strategy, which recursively factorizes the DFT of size $n \times n$ into $\log_2(n)$ sparse matrices (for simplicity we consider here a dimension n that is a power of two, but more flexible butterfly factorizations can be defined [9]), the butterfly matrices. If we denote $\boldsymbol{F} \in \mathbb{C}^{n \times n}$ the DFT matrix, then its factorization associated to the Cooley-Tukey algorithm is

$$\boldsymbol{F} = \boldsymbol{B}_1 \cdots \boldsymbol{B}_L$$

where $L := \log_2(n)$ and the $\boldsymbol{B}_i$ are structured sparse matrices with the so-called *Kronecker-sparse structure* [9].

To quantize the factors $\boldsymbol{B}_i$, in the spirit of what was done in the real-valued case [7], we aim to leverage Algorithm 1 to work on a product of complex-valued butterfly factors.

Quantization algorithm for butterfly matrices. First, let us take a detour into optimal quantization for a product of *two* (possibly sparse) matrices $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{C}^{n \times r}$: the quantization problem then reads

$$\boldsymbol{X}^*, \boldsymbol{Y}^* \in \arg \min_{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}} \in \mathcal{CF}_t} \|\boldsymbol{X}\boldsymbol{Y}^H - \hat{\boldsymbol{X}}\hat{\boldsymbol{Y}}^H\|^2 \qquad (6)$$

where $\mathcal{CF}_t$ is the set of matrices with coefficient in $\mathbb{CF}_t$ and the same support as $\boldsymbol{X}, \boldsymbol{Y}$. The support of a matrix $\boldsymbol{A}$ is the set of coordinates (i, j) where $\boldsymbol{A}_{i,j} \neq 0$. To the best of our knowledge, there is no method for finding the optimal quantization for this problem. However, if the rank-one matrices, $\boldsymbol{x}_i \boldsymbol{y}_i^*$, where $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are the corresponding columns of $\boldsymbol{X}$ and $\boldsymbol{Y}$, $1 \leq i \leq n$, have disjoint supports, then the problem (6) can be decomposed into r independent complex rank-one matrix quantization problems:

$$\|\boldsymbol{X}\boldsymbol{Y}^H - \hat{\boldsymbol{X}}\hat{\boldsymbol{Y}}^H\|^2 = \sum_{i=1}^{r} \|\boldsymbol{x}_i \boldsymbol{y}_i^H - \hat{\boldsymbol{x}}_i \hat{\boldsymbol{y}}_i^H\|^2. \qquad (7)$$

Each sub-problem can be addressed using Algorithm 1.

We can leverage the two-factor setting: the crux is that when considering certain so-called *chainable* Kronecker-sparse factors [9], for any subset of consecutive factors, the

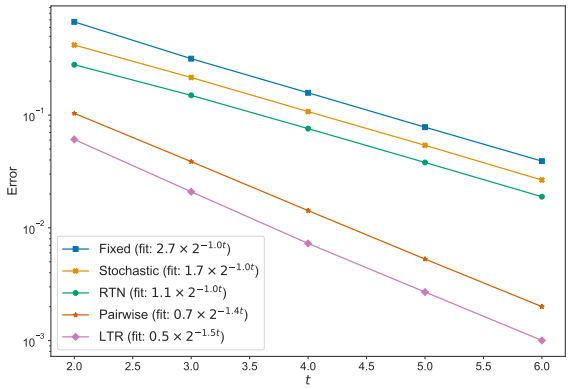

Figure 3 – Quantization error on the butterfly decomposition of 100 random matrices as a function of t ($n = 256$, $d_{\text{m}} = 2$).

product between $\boldsymbol{X} = \boldsymbol{B}_{l_0} \cdots \boldsymbol{B}_{l_1} \in \mathbb{C}^{n \times n}$ and $\boldsymbol{Y}^H = \boldsymbol{B}_{l_1+1} \cdots \boldsymbol{B}_{l_2} \in \mathbb{C}^{n \times n}$, with $1 \leq l_0 \leq l_1 \leq l_2 \leq L$, can be written as a sum of n rank-one matrices with disjoint support [8]. As suggested for the real-valued case [7], this can be used to heuristically decompose the product of L butterfly matrices into several products of two matrices to apply the optimal quantization of the problem (6). The quantization of each product will therefore be optimal, but the global quantization will not. As in [7] we consider two heuristics:

— pairwise: writing $(\boldsymbol{B}_1\boldsymbol{B}_2)(\boldsymbol{B}_3\boldsymbol{B}_4)\cdots(\boldsymbol{B}_{L-1}\boldsymbol{B}_L)$, quantization is applied to each pair of consecutive factors (if L is odd the last matrix is quantized using the RTN strategy).

— Left-to-Right (LTR): writing $\boldsymbol{B}_1\left(\boldsymbol{B}_2(\cdots(\boldsymbol{B}_{L-1}\boldsymbol{B}_L))\right)$, quantization is performed from left to right (details in [7]).

We also implement stochastic rounding [3] and fixed-point rounding [4], which work like RTN, i.e., quantization is applied element-wise without any rescaling.

Evaluation on random butterfly matrices. To compare these methods, we generate 10 random collections of Kronecker-sparse factors $\boldsymbol{B}_1, \ldots \boldsymbol{B}_L$ and quantize them with the methods described before to obtain $\hat{\boldsymbol{B}}_1 \cdots \hat{\boldsymbol{B}}_L$. Figure 3 shows the evolution of the quantization error $\rho := \frac{\|\boldsymbol{B}_1 \cdots \boldsymbol{B}_L - \hat{\boldsymbol{B}}_1 \cdots \hat{\boldsymbol{B}}_L\|}{\|\boldsymbol{B}_1 \cdots \boldsymbol{B}_L\|}$ for different values of t and $d_{\text{m}} = 0$. Despite the matrix product parenthesis heuristic, pairwise and LTR are more efficient than RTN, stochastic and fixed-point. Indeed, we see that pairwise and LTR need about 4 bits to achieve an error close to 10^{-2} while RTN, fixed-point and stochastic must have more than 6 bits to achieve this error. In general, according to the exponential fit, pairwise and LTR need $1 - \frac{1}{1.5} \approx 30\%$ fewer bits than RTN, fixed-point and Stochastic to achieve the same quantization error.

Evaluation on the quantization of the FFT. We also apply the five quantization algorithms to the (exact) Cooley-Tukey factorization $\boldsymbol{F} = \boldsymbol{B}_1 \cdots \boldsymbol{B}_L \in \mathbb{C}^{n \times n}$ where $\boldsymbol{F}$ is the DFT and look at the error made if we apply the quantized version of $\boldsymbol{F}$ to a vector $\boldsymbol{x}$ instead of the applying the original high-resolution DFT.

Let $\boldsymbol{x} \in \mathbb{R}^n$ be the signal and $\boldsymbol{y} := \boldsymbol{F}\boldsymbol{x} \in \mathbb{C}^n$ its Fourier transform. We also define $\hat{\boldsymbol{y}} := \hat{\boldsymbol{F}}\boldsymbol{x} = \hat{\boldsymbol{B}}_1 \cdots \hat{\boldsymbol{B}}_L \boldsymbol{x}$. The average of $\rho_{\text{fft}} := \frac{\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|}{\|\boldsymbol{y}\|}$ to over 10 standard Gaussian signals, with the five considered algorithms, are displayed in Table 2 with $n = 256$, $t = 5$ and $d_{\text{m}} = 0$. Like for the butterfly quantization error, the LTR and pairwise methods are more efficient than the RTN, stochastic and fixed-point quantizations.

Table 2 – ρ_{fft} with different quantization strategies for $n = 256$, $t = 5$ and $d_{\text{m}} = 0$.

	LTR	pairwise	RTN	Stochastic	Fixed
$\rho_{\text{fft}} \times 10^{-2}$	0.165	0.340	2.339	2.776	3.252

5 Conclusion

In this paper, we proposed an efficient quantization algorithm exploiting rescaling-invariances for complex-valued rank-one matrices. We showed that this approach enables more accurate quantization than a naive element-wise rounding. Applying this algorithm to butterfly matrices, which play a key role in many fast transforms such as the Fast Fourier Transform, we demonstrated the possibility to reduce quantization error for a given number of bits, or alternatively to reduce by 30% the required number of bits for a given precision.

Our algorithm depends on a parameter that controls the accuracy but also impacts the computation time. Empirically, this parameter is not very limiting, but since the real-valued case lead to a fully optimal algorithm with bounded complexity a natural challenge is to understand whether this remains possible in the complex-valued case via further mathematical analysis steps. Furthermore, throughout this paper, we have neglected underflow and overflow [2, 11]. A similar study taking these phenomena into account would be interesting.

This work opens the way to several perspectives: the first is the quantization of the product of matrices of any rank. The application to butterfly matrices leads us to consider applying our algorithm to other matrix decompositions, such as the Toeplitz matrices that models, for example, the covariance matrix of some time series [6]. Another challenge is to extend this work to quantize ReLU networks, which satisfy similar rescaling-invariances.

References

[1] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19(90):297–301, 1965.

[2] A. Cuyt, P. Kuterna, B. Verdonk, and D. Verschaeren. Underflow revisited. *Calcolo*, 39:169–179, 2002.

[3] E.-M. El Arar, D. Sohier, P. de Oliveira Castro, and E. Petit. Stochastic rounding variance and probabilistic bounds: A new approach. *SISC*, 45(5):C255–C275, 2023.

[4] M. Gerken. On fixed-point quantization schemes. In *38th MWSCAS*, volume 1, pages 350–353 vol.1, 1995.

[5] R. Graham, D. Knuth, and O. Patashnik. Concrete mathematics, a foundation for computer science. *Math. Gaz.*, 75(471):117–119, 1991.

[6] R. M. Gray. On unbounded toeplitz matrices and nonstationary time series with an application to information theory. *Information and control*, 24(2):181–196, 1974.

[7] R. Gribonval, T. Mary, and E. Riccietti. Optimal quantization of rank-one matrices in floating-point arithmetic—with applications to butterfly factorizations. 2023.

[8] Q.-T. Le, L. Zheng, E. Riccietti, and R. Gribonval. Fast learning of fast transforms, with guarantees. In *ICASSP*, pages 3348–3352. IEEE, 2022.

[9] Q.-T. Le, L. Zheng, E. Riccietti, and R. Gribonval. Butterfly factorization with error guarantees, November 2024. arXiv:2411.04506.

[10] L. Le Magoarou and R. Gribonval. Flexible multilayer sparse approximations of matrices and applications. *JSTSP*, 10(4):688–700, 2016.

[11] W. T. Padgett and D. V. Anderson. Quantization effects-round-off noise and overflow. In *Fixed-Point Signal Processing*, pages 83–111. Springer, 2009.

[12] E. Quemener and M. Corvellec. Sidus—the solution for extreme deduplication of an operating system. *Linux Journal*, 2013(235):3, 2013.