



HAL
open science

Mixed precision iterative refinement for low-rank matrix and tensor approximations

Marc Baboulin, Oguz Kaya, Théo Mary, Matthieu Robeyns

► To cite this version:

Marc Baboulin, Oguz Kaya, Théo Mary, Matthieu Robeyns. Mixed precision iterative refinement for low-rank matrix and tensor approximations. 2023. hal-04115337

HAL Id: hal-04115337

<https://inria.hal.science/hal-04115337v1>

Preprint submitted on 2 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

1 **MIXED PRECISION ITERATIVE REFINEMENT FOR**
2 **LOW-RANK MATRIX AND TENSOR APPROXIMATIONS***

3 MARC BABOULIN[†], OGUZ KAYA[†], THEO MARY[‡], AND MATTHIEU ROBEYNS^{†,§}

4 **Abstract.** We present a new mixed precision algorithm to compute low-rank matrix and ten-
5 sor approximations, a fundamental task in numerous applications in scientific computing and data
6 analysis. Our algorithm is reminiscent of the iterative refinement framework for linear systems: we
7 first compute a low-rank approximation in low precision and then refine its accuracy by iteratively
8 updating it. We carry out an error analysis of our algorithm which proves that we can reach a high
9 accuracy while performing most of the operations in low precision. We measure the computational
10 cost of the algorithm, which depends on the numerical rank of the input (matrix or tensor) as well
11 as the speed ratio between low and high precision arithmetic. We identify two situations where our
12 method has a strong potential : when the hardware provides fast low precision matrix multiply-
13 accumulate units, and when the numerical rank of the input is small at low accuracy levels. We
14 confirm experimentally the potential of our algorithm for computing various low-rank matrix and
15 tensor decompositions such as SVD, QR, Tucker, hierarchical Tucker, and tensor-train.

16 **Key words.** Matrix and tensor computations, low-rank approximations, mixed precision algo-
17 rithms, iterative refinement, randomized SVD, tensor decompositions.

18 **AMS subject classifications.** 65F55, 65G50, 65Y20, 15A69

19 **1. Introduction.** Low-rank approximations (LRA) are a powerful tool used in
20 many scientific applications to reduce the dimension of large scale data. For example,
21 an $n \times n$ matrix X may be approximated by a low-rank product UV^T of $n \times r$ matrices
22 U and V , reducing the storage cost from $O(n^2)$ to $O(nr)$. This storage cost is even
23 more critical for tensors [20, 6, 14], the higher dimensional generalization of matrices
24 : a d th order tensor X requires storage in $O(n^d)$ that is exponential in the number of
25 dimensions d . This *curse of dimensionality* can be tackled via LRA by decomposing
26 the full tensor as a product of tensors of lower order and lower rank. Various such
27 decompositions exist, such as the Tucker [28, 9], the tensor-train (TT) [27], and the
28 hierarchical Tucker (HT) [15, 13] formats.

29 However, computing matrix or tensor low-rank decompositions at a controlled
30 approximation accuracy ε is computationally expensive; it represents the bottleneck
31 of many LRA-based applications. Therefore, developing high performance algorithms
32 for computing LRA is an important problem that has been the object of many studies;
33 see, for instance, these surveys on matrices [19] and tensors [20, 14, 6].

34 Our work is specifically motivated by the recent and rapid emergence of *low*
35 *precision arithmetics*, in particular half precision floating-point arithmetics such as
36 the IEEE fp16 and bfloat16 formats. These low precision arithmetics provide great
37 computational benefits, in terms of storage, speed, and energy [17]. On modern
38 hardware, and in particular on Graphics Processing Unit (GPU) accelerators, low
39 precision arithmetics can be orders of magnitude faster than the standard single (fp32)
40 or double (fp64) precision arithmetics. However, using low precision also degrades
41 the accuracy of the computations; for example half precision arithmetics provide at
42 best between 3 and 4 digits of accuracy, depending on the format. This motivates

*Version of June 2, 2023.

[†]University Paris-Saclay, CNRS, LISN, Gif-sur-Yvette, France, (marc.baboulin,oguz.kaya,matthieu.robeyns@universite-paris-saclay.fr)

[‡]Sorbonne Université, CNRS, LIP6, Paris, France, (theo.mary@lip6.fr)

[§]Corresponding author.

43 the need for *mixed precision algorithms*, which combine multiple precision formats
44 with the goal of achieving the high performance of the low precision while preserving
45 the high accuracy of the higher precision. The emergence of low precision on modern
46 hardware has generated much recent interest in mixed precision algorithms, with many
47 successful examples in numerical linear algebra; see the survey [17] for an overview of
48 this field.

49 In this paper, we seek to develop mixed precision algorithms for computing LRA.
50 Contrary to other linear algebra routines such as the solution of linear systems, there
51 has been relatively little work on designing mixed (or even low) precision algorithms
52 for LRA. Amestoy et al. [2] describe a mixed precision matrix LRA that partitions the
53 low-rank factors into several block columns stored in different precisions depending
54 on the singular values of the matrix; this approach can make use of low precisions
55 for matrices with rapidly decaying singular values. A similar approach is proposed
56 by Ooi et al. [23] for \mathcal{H} -matrices. Connolly et al. [7] perform the rounding error
57 analysis of the randomized singular value decomposition (SVD) for matrix LRA, in
58 particular for the adaptive algorithm of Martinsson and Voronin [22] which builds an
59 LRA by increasing its rank until the prescribed accuracy is reached. The analysis in
60 [7] reveals that lower precision can be used when the norm of the approximation error
61 becomes small enough; this once more corresponds to the case where the matrix has
62 rapidly decaying singular values. Finally, a recent paper of Ootomo and Yokota [25]
63 also proposes a mixed precision randomized SVD; their approach stores the random
64 projection matrix in half precision while keeping the input matrix in single precision,
65 and obtains speedup by emulating single precision computations with GPU tensor
66 cores [11, 24]. There have also been very few attempts to develop mixed precision
67 LRA for tensors. We can mention a recent work by Yang et al. [30] that proposes an
68 iterative CP decomposition using mixed precision stochastic gradient descent. In a
69 context different from LRA, the recent work of Agullo et al. [1] is worth mentioning:
70 they consider the solution of linear systems $Ax = b$ via GMRES, where A can be
71 approximated under tensor format [8]. This approach can then benefit from a mixed
72 precision implementation of GMRES.

73 In this paper, we propose a new method for computing LRA in mixed precision
74 arithmetic. Our approach is applicable to basically any LRA algorithm, involving
75 either matrices or tensors. It is reminiscent of the iterative refinement framework
76 used for solving linear systems: the idea is to first compute an LRA in low precision,
77 then evaluate the error (or residual) from this first LRA, and re-apply the same LRA
78 kernel to this error term to obtain a correction term that is used to refine the accuracy
79 of the LRA. This can be repeated iteratively to reach any level of desired accuracy.
80 The refined LRA is obtained as the sum of the original low precision LRA and the
81 correction term, and is thus of larger yet still of low-rank. In order to contain the
82 rank growth and maintain the optimal rank throughout the iterations, our method
83 employs a “recompression” strategy that is performed in high precision but whose
84 cost stays asymptotically smaller than that of LRA.

85 We carry out an error analysis of our method which proves that it can reach a
86 high accuracy while performing most of the operations in low precision. In particular,
87 we show that the precision used for the LRA kernel—which is the computational
88 bottleneck of the whole method—only affects the convergence speed of the process,
89 but not its attainable accuracy. In order to assess under which conditions we can
90 expect our method to be beneficial, we perform a complexity analysis that measures
91 the cost of the method as a function of the numerical rank of the input as well as the
92 speed ratio between the low and high precision arithmetic. We identify two situations

93 where our method has a strong potential. The first is when the hardware provides
 94 fast low precision matrix multiply–accumulate units, also called block FMA units [4],
 95 which allow for computing the low precision LRA at very high speed. The second is
 96 when the numerical rank of the input is small at low accuracy levels, which means
 97 that the singular values of the matrix or tensor are rapidly decaying; in this case, the
 98 first iterations of our method becomes inexpensive.

99 We apply our method to various low-rank matrix and tensor decompositions :
 100 SVD, QR, Tucker, HT, and TT. We perform some MATLAB experiments to confirm
 101 that our method is able to compute an LRA at any desired level of accuracy, while
 102 mostly using the low precision arithmetic.

103 The rest of this paper is organized as follows. In section 2, we describe our
 104 main algorithm of iterative refinement for LRA and provide the corresponding error
 105 and complexity analysis. Then we apply this method to various LRA algorithms for
 106 matrices and tensors in section 3 and section 4, respectively. We validate our method
 107 experimentally in section 5. Finally, we provide concluding remarks in section 6.

108 **1.1. Notations.** We denote as X the object of interest, a matrix or a tensor,
 109 and as F its low-rank factors that we seek to compute. If $X \in \mathbb{R}^{m \times n}$ is a matrix, then
 110 $F = UV^T$ is usually implicitly represented by the product of two matrices $U \in \mathbb{R}^{m \times r}$,
 111 $V \in \mathbb{R}^{n \times r}$ of smaller dimensions, where r is the rank of F . Similarly, if $X \in \mathbb{R}^{n_1 \times \dots \times n_d}$
 112 is a tensor of order d , its low-rank factors F are represented using a number of matrices
 113 and/or tensors of smaller order; the precise expression of F depends on the chosen
 114 format. In particular, in this article we will consider three different low-rank tensor
 115 decompositions:

- 116 • the Tucker format [9], which uses d matrices $U^{(1)} \in \mathbb{R}^{n_1 \times r_1}, \dots, U^{(d)} \in$
 117 $\mathbb{R}^{n_d \times r_d}$, where $r_i \leq n_i$, and a d th-order core tensor $G \in \mathbb{R}^{r_1 \times \dots \times r_d}$;
- 118 • the TT format [27], which uses d 3rd-order tensors $U^{(1)} \in \mathbb{R}^{r_0 \times n_1 \times r_1}, U^{(2)} \in$
 119 $\mathbb{R}^{r_1 \times n_2 \times r_2}, \dots, U^{(d)} \in \mathbb{R}^{r_{d-1} \times n_d \times r_d}$, where $r_0 = r_d = 1$;
- 120 • the HT format [15, 13], which recursively applies the Tucker format to the
 121 core tensor G to obtain a binary tree of tensors. Leaf nodes of the tree are
 122 matrices $U^{(1)} \in \mathbb{R}^{n_1 \times r_{d-1}}, \dots, U^{(d)} \in \mathbb{R}^{n_d \times r_{2d-2}}$ and the rest are 3rd-order
 123 tensors $G^{(1)} \in \mathbb{R}^{r_0 \times r_1 \times r_2}, \dots, G^{(d-1)} \in \mathbb{R}^{r_d \times r_{2d-3} \times r_{2d-2}}$, where $G^{(1)}$ is the
 124 root and $r_0 = 1$.

125 Note that our main iterative refinement method described in the next section does not
 126 require any specific knowledge or property of the tensor decomposition, and simply
 127 denotes its low-rank factors as F .

128 To control the accuracy of the LRA, we will distinguish two types of parameters,
 129 denoted as ε and u . The truncation threshold ε controls the low-rank truncation error
 130 and thus the accuracy of the LRA in exact arithmetic. The unit roundoff u controls
 131 the rounding errors and thus the additional loss of accuracy incurred by the use of
 132 finite precision arithmetic. Actually, in our iterative refinement method, we will have
 133 four parameters $\varepsilon, u, \varepsilon_\ell$, and u_ℓ , where ε_ℓ and u_ℓ denote the truncation threshold and
 134 unit roundoff used by the low precision LRA, whereas ε and u denote the truncation
 135 threshold and unit roundoff used by the rest of the operations performed in high
 136 precision, and correspond to the final target accuracy.

137 For a matrix X , we denote by $\text{RANK}(X, \varepsilon)$ the numerical rank of X at accuracy
 138 ε , which is the smallest integer r_ε such that there exists low-rank factors F of rank
 139 r_ε satisfying a relative accuracy ε , that is, $\|X - F\| \leq \varepsilon \|X\|$ for a given choice of
 140 norm. This definition of the numerical rank extends to the Tucker, TT, and HT
 141 representations of tensors by defining $\text{RANK}(X, \varepsilon)$ as the vector of length \bar{d} whose

142 coefficients correspond to the “inner” dimensions connecting the underlying matrices
 143 and/or tensors in the decomposition. The length \bar{d} depends on both d and the type
 144 of decomposition:

$$145 \quad \bar{d} = \begin{cases} d & \text{(Tucker)} \\ d - 1 & \text{(TT)} \\ 2d - 2 & \text{(HT)} \end{cases}. \quad (1.1)$$

146 Technically, the rank vector is determined by a vector of error tolerances ε_i , $i = 1: \bar{d}$,
 147 whose sum yields the target accuracy $\varepsilon = \sum_{i=1}^{\bar{d}} \varepsilon_i$. In practice, the truncation error
 148 is uniformly distributed by setting $\varepsilon_i = \varepsilon/\bar{d}$ [27, 29, 13].

149 **2. Iterative refinement for low-rank approximation.** In this section, we
 150 propose a mixed precision algorithm to compute LRA at high accuracy while per-
 151 forming most of the computations in low precision.

We consider the problem of computing low-rank factors F of a matrix or tensor
 X satisfying a relative accuracy ε , that is, $\|X - F\| \leq \varepsilon\|X\|$. In finite precision
 arithmetic, we must carefully select the precision in order to achieve this accuracy. In a
 uniform precision context (where we perform all computations in the same precision),
 we must use a precision whose unit roundoff u is safely smaller than ε , that is, $u = \theta\varepsilon$,
 with $\theta \leq 1$ some parameter. Indeed, we may then expect the computed factors \widehat{F} to
 satisfy

$$\|X - \widehat{F}\| \leq (\varepsilon + cu)\|X\| = (1 + c\theta)\varepsilon\|X\|$$

152 for some constant c , where the term $c\theta$ accounts for the effect of rounding errors and
 153 can be made as small as needed by decreasing θ (decreasing u , that is, increasing the
 154 precision).

155 For applications requiring relatively high accuracy (small values of ε), this uniform
 156 precision approach therefore cannot make use of lower precisions. This motivates
 157 us to propose a mixed precision method, outlined below, which uses low precision
 158 to compute a first approximate set of factors F_0 , and then refines them into more
 159 accurate factors F_1 as follows.

- 160 1. Compute low-rank factors F_0 of X at low accuracy ε_ℓ and in low precision
- 161 $u_\ell = \theta\varepsilon_\ell$.
- 162 2. Compute the error $E = X - F_0$ in high precision u .
- 163 3. Compute low-rank factors F_E of E at low accuracy ε_ℓ and in low precision
- 164 $u_\ell = \theta\varepsilon_\ell$.
- 165 4. Update the low-rank factors of X to $F_1 = F_0 + F_E$ in high precision u .

This approach is based on the observation that, while the first factors F_0 will
 achieve a low accuracy ε_ℓ relative to $\|X\|$,

$$\|X - F_0\| \leq \varepsilon_\ell\|X\|,$$

the factors F_E of the error E will achieve a low accuracy ε_ℓ relative to $\|E\|$,

$$\|E - F_E\| \leq \varepsilon_\ell\|E\|.$$

Neglecting for now the effect of rounding errors, we can expect to have $\|E\| \approx \varepsilon_\ell\|X\|$,
 and therefore the combined factors $F_0 + F_E$ will satisfy an accuracy of about ε_ℓ^2 :

$$\|X - F_1\| \lesssim \varepsilon_\ell^2\|X\|.$$

166 This idea is reminiscent of the iterative refinement for linear systems $Ax = b$,
 167 where step 1 corresponds to computing an initial solution x_0 , step 2 corresponds to

168 computing the residual $r = b - Ax_0$, and step 3 corresponds to solving another linear
 169 system $Ad = r$ for a correction term d , which is used to update $x_1 = x_0 + d$ (our step
 170 4). We therefore baptize this approach “iterative refinement” for LRA.

171 One of the crucial insights that makes this method effective is that the error E
 172 has a low numerical rank whenever X has one too, since it is the sum of X and F_0 ,
 173 which is low-rank by construction. To be specific, we will prove in [subsection 2.2](#)
 174 that $\text{RANK}(E, \varepsilon_\ell)$ is at most $2 \text{RANK}(X, \varepsilon_\ell^2)$ and, for the same reason, the rank of
 175 the refined factors F_1 is bounded by $3 \text{RANK}(X, \varepsilon_\ell^2)$. The factors F_1 can then be
 176 cheaply recompressed into factors of optimal rank $\text{RANK}(X, \varepsilon_\ell^2)$ in high precision u
 177 by exploiting their (suboptimal) low-rank structure.

178 If accuracy higher than ε_ℓ^2 is needed, we can apply the method again on F_1 to ob-
 179 tain an accuracy of ε_ℓ^3 , and so on; we obtain [Algorithm 2.1](#), which repeats this process
 180 until the desired accuracy is achieved. [Algorithm 2.1](#) is described within a general
 181 iterative refinement framework that uses an arbitrary LRA algorithm: $\text{LRA}(X, \varepsilon)$ re-
 182 turns low-rank factors F of X at accuracy ε . We also require a `DECOMPRESS` kernel
 183 which transforms low-rank factors F back to a full-size object, and a `RECOMPRESS`
 184 kernel which takes low-rank factors F and ε as input, and computes their optimal
 185 LRA at accuracy ε .

Algorithm 2.1 Iterative refinement for LRA.

Input: X , the matrix or tensor of interest;
 ε , the desired relative accuracy for the approximation;
 n_{it} , the maximum number of iterations;
 LRA, a low-rank approximation algorithm;
 DECOMPRESS, RECOMPRESS: decompression and recompression algorithms.

Output: F , the low-rank factors X .

```

1: Compute  $F = \text{LRA}(X, \varepsilon_\ell)$  in precision  $u_\ell$ .
2: for  $i = 1$  to  $n_{\text{it}}$  do
3:   Compute  $E = X - \text{DECOMPRESS}(F)$  in precision  $u$ .
4:   Compute  $\alpha = \|E\|$  in precision  $u$ .
5:   If  $\alpha \leq \varepsilon \|X\|$ , exit.
6:   Scale  $E \leftarrow \alpha^{-1} E$  in precision  $u$ .
7:   Compute  $F_E = \text{LRA}(E, \varepsilon_\ell)$  in precision  $u_\ell$ .
8:   Scale back  $F_E \leftarrow \alpha F_E$  in precision  $u$ .
9:   Compute  $F = \text{RECOMPRESS}(F + F_E, \varepsilon)$  in precision  $u$ .
10: end for

```

186 [Algorithm 2.1](#) incorporates two additional steps to make the method effective.
 187 First, the error E is scaled by the inverse of its norm ([Line 6](#)) before computing its
 188 low-rank factors F_E , and then scaled back after ([Line 8](#)). This is done to prevent the
 189 elements of E from underflowing when converted to the lower precision, in the case
 190 where the arithmetic uses not only a reduced number of bits in the significand but also
 191 in the exponent, such as is the case for the IEEE half precision fp16 arithmetic. Sec-
 192 ond, the factors F are recompressed ([Line 9](#)) to their optimal low-rank representation
 193 to avoid the rank growing out of control during the iterations. Indeed, as mentioned
 194 above, the rank of F is bounded by $3 \text{RANK}(X, \varepsilon_\ell^2)$ after one iteration. Therefore, in
 195 absence of recompression, the rank would grow as $3^k \text{RANK}(X, \varepsilon_\ell^2)$ after k iterations,
 196 and would quickly make the method unaffordable. Fortunately, this recompression
 197 can be performed inexpensively for most LRA algorithms of interest.

198 [Algorithm 2.1](#) has four parameters that control its accuracy: ε , ε_ℓ , u , and u_ℓ .
199 • ε indicates the target accuracy for the final factors, and is prescribed by the
200 user as input to [Algorithm 2.1](#).
201 • u is the unit roundoff of the high precision, which should be taken to be the
202 lowest possible such that u is still safely smaller than ε : $u = \theta\varepsilon$, $\theta \leq 1$.
203 • u_ℓ is the unit roundoff of the low precision, which is used to perform the most
204 expensive parts of the computation, the calls to LRA. Its choice depends
205 on both the available arithmetics on the target hardware, and the target
206 accuracy ε . Indeed, lowering the precision makes each iteration faster but
207 requires more of them.
208 • Finally, ε_ℓ is the tolerance used by the LRA kernel; since LRA is performed
209 in precision u_ℓ , ε_ℓ should be set such that u_ℓ is safely smaller than ε_ℓ , that
210 is, $\varepsilon_\ell = u_\ell/\theta_\ell$, $\theta_\ell \leq 1$. We note that this is necessary because using an ε_ℓ too
211 close to u_ℓ prevents most LRA algorithms to reliably detect the correct rank,
212 due to the noise introduced by rounding errors.

213 In the rest of this section, we first perform an error analysis to determine the
214 attainable accuracy and convergence rate of [Algorithm 2.1](#). We then perform a com-
215 plexity analysis to determine under which conditions the algorithm can be expected to
216 be faster than a standard uniform precision LRA performed entirely in high precision
217 u .

218 **2.1. Error analysis.** In order to carry out the error analysis of [Algorithm 2.1](#),
219 we will use the standard model of floating-point arithmetic [[16](#), sect. 2.2]. In addition,
220 we also need to make the following three assumptions on the numerical behavior of
221 the LRA, DECOMPRESS, and RECOMPRESS kernels. First, we assume that computing
222 $F = \text{LRA}(X, \varepsilon_\ell)$ in precision u_ℓ yields computed factors \widehat{F} satisfying

$$223 \quad \|X - \widehat{F}\| \leq (\varepsilon_\ell + b_1 u_\ell) \|X\| = (1 + b_1 \theta_\ell) \varepsilon_\ell \|X\|. \quad (2.1)$$

224 Second, we assume that computing $F = \text{DECOMPRESS}(F)$ in precision u yields com-
225 puted factors \widehat{F} satisfying

$$226 \quad \|F - \widehat{F}\| \leq b_2 u \|F\| = b_2 \theta \varepsilon \|F\|. \quad (2.2)$$

227 Third, we assume that computing $F = \text{RECOMPRESS}(F, \varepsilon)$ in precision u yields com-
228 puted factors \widehat{F} satisfying

$$229 \quad \|F - \widehat{F}\| \leq (\varepsilon + b_3 u) \|F\| = (1 + b_3 \theta) \varepsilon \|F\|. \quad (2.3)$$

230 For simplicity, we ignore any rounding errors associated with the scaling by $\alpha = \|E\|$,
231 which are negligible and can in fact be prevented by rounding α to the nearest power
232 of two (in binary floating-point arithmetic).

233 We are now ready to prove the following result.

234 **THEOREM 2.1.** *If [Algorithm 2.1](#) is applied to X with LRA, DECOMPRESS, and*
235 *RECOMPRESS kernels satisfying (2.1)–(2.3), then after k iterations, the computed*
236 *factors \widehat{F} satisfy*

$$237 \quad \|X - \widehat{F}\| \leq (\phi^{k+1} + \xi + O(\varepsilon_\ell \varepsilon)) \|X\|, \quad (2.4)$$

238 *with $\phi = (1 + b_1 \theta_\ell) \varepsilon_\ell + (b_2 + 1) \theta \varepsilon$ and $\xi = (1 + (b_2 + b_3 + 2) \theta) \varepsilon$.*

Proof. Defining \widehat{F}_i as the computed factors after i iterations, our goal is to obtain
a bound of the form

$$\|X - \widehat{F}_{i+1}\| \leq \phi \|X - \widehat{F}_i\| + \xi \|X\|, \quad \phi < 1,$$

239 which will allow us to conclude that the error contracts by a factor ϕ at each iteration,
 240 until it converges to its maximum attainable accuracy ξ .

241 Given \widehat{F}_i , the first step is to decompress it and compute E at [Line 3](#); by [\(2.2\)](#),
 242 the computed \widehat{E} satisfies

$$243 \quad \|X - \widehat{F}_i - \widehat{E}\| \leq \theta\varepsilon((b_2 + 1)\|\widehat{F}_i\| + \|X\|), \quad (2.5)$$

244 with an extra term $\theta\varepsilon(\|\widehat{F}_i\| + \|X\|)$ on the right-hand side coming from the rounding
 245 error incurred by the subtraction. Using the triangle inequality $\|\widehat{F}_i\| \leq \|X - \widehat{F}_i\| +$
 246 $\|X\|$, we can rearrange [\(2.5\)](#) as

$$247 \quad \|X - \widehat{F}_i - \widehat{E}\| \leq \theta\varepsilon((b_2 + 1)\|X - \widehat{F}_i\| + (b_2 + 2)\|X\|). \quad (2.6)$$

248 Then we compute $\text{LRA}(\widehat{E}, \varepsilon_\ell)$ at [Line 7](#); by [\(2.1\)](#) the computed \widehat{F}_E satisfies

$$249 \quad \|\widehat{E} - \widehat{F}_E\| \leq (1 + b_1\theta_\ell)\varepsilon_\ell\|\widehat{E}\|. \quad (2.7)$$

250 By using [\(2.6\)](#) and the triangle inequality, we rearrange [\(2.7\)](#) as

$$251 \quad \|\widehat{E} - \widehat{F}_E\| \leq (1 + b_1\theta_\ell)\varepsilon_\ell(\|X - \widehat{F}_i\| + \|X - \widehat{F}_i - \widehat{E}\|) \quad (2.8)$$

$$252 \quad = (1 + b_1\theta_\ell)\varepsilon_\ell\|X - \widehat{F}_i\| + O(\varepsilon_\ell\varepsilon), \quad (2.9)$$

254 where we do not keep track explicitly of high order terms in $O(\varepsilon_\ell\varepsilon)$ for the sake of
 255 readability. Finally, we obtain the next iterate F_{i+1} by recompressing $\widehat{F}_i + \widehat{F}_E$
 256 at [Line 9](#); by [\(2.3\)](#), the computed \widehat{F}_{i+1} satisfies

$$257 \quad \|\widehat{F}_i + \widehat{F}_E - \widehat{F}_{i+1}\| \leq (1 + b_3\theta)\varepsilon\|\widehat{F}_i + \widehat{F}_E\|. \quad (2.10)$$

258 By using [\(2.6\)](#), [\(2.9\)](#), and the triangle inequality, we rearrange [\(2.10\)](#) as

$$259 \quad \|\widehat{F}_i + \widehat{F}_E - \widehat{F}_{i+1}\| \leq (1 + b_3\theta)\varepsilon(\|X - \widehat{F}_i - \widehat{E}\| + \|\widehat{E} - \widehat{F}_E\| + \|X\|) \quad (2.11)$$

$$260 \quad = (1 + b_3\theta)\varepsilon\|X\| + O(\varepsilon_\ell\varepsilon). \quad (2.12)$$

262 Using the triangle inequality together with [\(2.6\)](#), [\(2.9\)](#), and [\(2.12\)](#), we obtain

$$263 \quad \|X - \widehat{F}_{i+1}\| = \|X - \widehat{F}_i - \widehat{E} + \widehat{E} - \widehat{F}_E + \widehat{F}_i + \widehat{F}_E - \widehat{F}_{i+1}\| \quad (2.13)$$

$$264 \quad \leq \|X - \widehat{F}_i - \widehat{E}\| + \|\widehat{E} - \widehat{F}_E\| + \|\widehat{F}_i + \widehat{F}_E - \widehat{F}_{i+1}\| \quad (2.14)$$

$$265 \quad \leq \phi\|X - \widehat{F}_i\| + \xi\|X\| + O(\varepsilon_\ell\varepsilon), \quad (2.15)$$

267 with

$$268 \quad \phi = (1 + b_1\theta_\ell)\varepsilon_\ell + (b_2 + 1)\theta\varepsilon \quad (2.16)$$

269 and

$$270 \quad \xi = (1 + (b_2 + b_3 + 2)\theta)\varepsilon. \quad (2.17)$$

271 Noting that the first factors \widehat{F}_0 computed at [Line 1](#) satisfy by [\(2.1\)](#)

$$272 \quad \|X - \widehat{F}_0\| \leq (1 + b_1\theta_\ell)\varepsilon_\ell\|X\| \leq \phi\|X\| \quad (2.18)$$

273 concludes the proof. \square

274 **Theorem 2.1** states that the approximation error $\|X - \widehat{F}\|$ contracts by a factor
 275 $\phi = O(\varepsilon_\ell)$ at each iteration, until it reaches its maximum attainable accuracy $\xi =$
 276 $O(\varepsilon)$. Thus, after k iterations, the error is of order $\varepsilon_\ell^{k+1} + \varepsilon$, which means that we
 277 can actually estimate in advance approximately how many iterations are needed to
 278 achieve the desired accuracy ε (up to constants):

$$279 \quad n_{\text{it}} = \log(\varepsilon) / \log(\varepsilon_\ell) - 1. \quad (2.19)$$

280 It is worth noting that, unlike iterative refinement for linear systems, there is
 281 no dependence on the condition number of X and thus the only condition for **Algo-**
 282 **rithm 2.1** to converge is that $\phi < 1$, which should always be the case as long as θ_ℓ
 283 is sufficiently small. Therefore, the algorithm is extremely general, can be applied to
 284 any matrix or tensor of low numerical rank, and can make use of potentially very low
 285 precisions.

286 From a numerical perspective, **Algorithm 2.1** is therefore quite appealing. We next
 287 discuss under which conditions it is also attractive from a computational perspective.

288 **2.2. Complexity analysis.** The cost of **Algorithm 2.1** will mainly depend on
 289 two factors: the relative speed for computing in different precisions on the target
 290 hardware, and the numerical ranks of the objects encountered during the iterations
 291 (X , E , and F).

292 We begin by bounding these ranks solely as a function of the numerical ranks of
 293 X at given accuracies. In order to do so, we will need to bound the numerical rank of
 294 $A+B$ as a function of that of A and B by using the following lemma. In what follows,
 295 recall that when X is a tensor, $\text{RANK}(X, \varepsilon)$ is a vector (see **subsection 1.1**) and all
 296 the (in)equalities involving this quantity should be interpreted componentwise.

LEMMA 2.2.

$$297 \quad \text{RANK}(A+B, \varepsilon) \leq \text{RANK}\left(A, \frac{\varepsilon\|A+B\|}{2\|A\|}\right) + \text{RANK}\left(B, \frac{\varepsilon\|A+B\|}{2\|B\|}\right). \quad (2.20)$$

Proof. For the purpose of this proof only, we introduce an alternative definition
 of numerical rank that measures accuracy in an absolute sense, rather than a relative
 one. Let $\text{RANKABS}(X, \varepsilon)$ be the the smallest integer r_ε such that there exists F of
 rank r_ε satisfying $\|X - F\| \leq \varepsilon$. The RANKABS operator satisfies

$$\text{RANKABS}(X, \varepsilon\|X\|) = \text{RANK}(X, \varepsilon)$$

and

$$\text{RANKABS}(A+B, \varepsilon) \leq \text{RANKABS}(A, \varepsilon/2) + \text{RANKABS}(B, \varepsilon/2).$$

298 Therefore, we have that

$$\begin{aligned} 299 \quad \text{RANK}(A+B, \varepsilon) &= \text{RANKABS}(A+B, \varepsilon\|A+B\|) \\ 300 \quad &\leq \text{RANKABS}(A, \varepsilon\|A+B\|/2) + \text{RANKABS}(B, \varepsilon\|A+B\|/2) \\ 301 \quad &= \text{RANK}(A, \varepsilon\|A+B\|/2\|A\|) + \text{RANK}(B, \varepsilon\|A+B\|/2\|B\|). \quad \square \end{aligned}$$

303 Let us consider the i th iteration of **Algorithm 2.1**. Using (2.20), we can bound
 304 the rank of $E = X - F$ as

$$305 \quad \text{RANK}(E, \varepsilon_\ell) \leq \text{RANK}(X, \varepsilon_\ell\|E\|/2\|X\|) + \text{RANK}(F). \quad (2.21)$$

306 Since F is the low-rank factorization of X after i iterations, it achieves an accuracy
 307 of ε_ℓ^i and so

$$308 \quad \text{RANK}(F) = \text{RANK}(X, \varepsilon_\ell^i) \quad (2.22)$$

309 (we assume here that $\varepsilon_\ell^i \leq \varepsilon$ as otherwise the method would be stopped). After i
 310 iterations, $\|E\| \leq \varepsilon_\ell^i \|X\|$ and so overall (2.21) becomes

$$311 \quad \text{RANK}(E, \varepsilon_\ell) \leq \text{RANK}(X, \varepsilon_\ell^{i+1}/2) + \text{RANK}(X, \varepsilon_\ell^i). \quad (2.23)$$

312 Since $\text{RANK}(X, \varepsilon') \leq \text{RANK}(X, \varepsilon)$ for $\varepsilon \leq \varepsilon'$, assuming that $\varepsilon \leq \varepsilon_\ell^{i+1}/2$, we can obtain
 313 a simpler but weaker version of (2.23),

$$314 \quad \text{RANK}(E, \varepsilon_\ell) \leq 2 \text{RANK}(X, \varepsilon). \quad (2.24)$$

315 Thus, the rank of E at any iteration is at most twice as large as the numerical rank
 316 of X at accuracy ε .

317 With a similar argument, the rank of the refined factors $F + F_E$ is bounded by

$$318 \quad \text{RANK}(F + F_E) \leq \text{RANK}(F) + \text{RANK}(F_E) \quad (2.25)$$

$$319 \quad = \text{RANK}(F) + \text{RANK}(E, \varepsilon_\ell) \quad (2.26)$$

$$320 \quad \leq \text{RANK}(X, \varepsilon_\ell^{i+1}/2) + 2 \text{RANK}(X, \varepsilon_\ell^i), \quad (2.27)$$

322 where (2.27) follows from (2.22) and (2.23). Again, a simpler but weaker bound is

$$323 \quad \text{RANK}(F + F_E) \leq 3 \text{RANK}(X, \varepsilon), \quad (2.28)$$

324 showing that at any iteration the rank of the factors before recompression is at most
 325 three times the numerical rank of X at accuracy ε .

326 Now that we have bounded the ranks of the objects appearing in Algorithm 2.1,
 327 we are ready to analyze its cost. We denote as p and s the product and the sum
 328 of the dimensions of X , respectively. Thus, if $X \in \mathbb{R}^{m \times n}$ is a matrix, $p = mn$ and
 329 $s = m + n$; if $X \in \mathbb{R}^{n_1 \times \dots \times n_d}$ is a tensor of order d , $p = \prod n_i$ and $s = \sum n_i$. For
 330 readability, we only report the dominant term for the cost of each assumption/line,
 331 and assume a large scale setting, so that $p \gg s$.

332 We make the following assumptions on the flops cost of the different kernels:

$$333 \quad \text{Flops LRA}(X, \varepsilon) = c_1 p \text{RANK}(X, \varepsilon)_1 + o(p), \quad (2.29a)$$

$$334 \quad \text{Flops DECOMPRESS}(F) = c_2 p \text{RANK}(F)_1 + o(p), \quad (2.29b)$$

$$335 \quad \text{Flops RECOMPRESS}(F, \varepsilon) = o(p). \quad (2.29c)$$

337 For tensors, the cost of LRA and DECOMPRESS depends on the order in which the
 338 dimensions are treated (see section 4). Here we assume they are treated in the natural
 339 order, without loss of generality since the dimensions can be arbitrarily reordered. In
 340 this case, the dominant cost of LRA and DECOMPRESS is proportional to the first
 341 coefficient of the rank vectors, denoted as $\text{RANK}(X, \varepsilon)_1$. Note that for matrices, the
 342 rank is a scalar so that $\text{RANK}(X, \varepsilon)_1 = \text{RANK}(X, \varepsilon)$.

343 Let us now measure the flops cost of the i th iteration of Algorithm 2.1.

- 344 • **Line 3:** $c_2 p \text{RANK}(X, \varepsilon_\ell^i)_1$ flops.
- 345 • **Line 4:** $2p$ flops (for the Frobenius norm).
- 346 • **Line 6:** p flops.
- 347 • **Line 7:** $c_1 p (\text{RANK}(X, \varepsilon_\ell^{i+1}/2)_1 + \text{RANK}(X, \varepsilon_\ell^i)_1)$ flops.

- 348 • [Line 8](#): $o(p)$ flops.
- 349 • [Line 9](#): $o(p)$ flops.

350 The dominant steps are the calls to LRA ([Line 7](#)), which is performed in low
 351 precision, and to DECOMPRESS ([Line 3](#)), which is performed in high precision. The
 352 computation of α and the scaling by α ([Lines 4](#) and [6](#)) could also be significant if the
 353 ranks are very small.

354 Summing the costs of these dominant steps across all n_{it} iterations, plus the cost
 355 of the initial LRA ([Line 1](#)), gives a total flops cost of

$$\begin{aligned}
 \text{Flops IR} &= c_1 p \text{RANK}(X, \varepsilon_\ell)_1 + p \sum_{i=1}^{n_{\text{it}}} \left(c_1 \text{RANK}(X, \varepsilon_\ell^{i+1}/2)_1 \right. \\
 &\quad \left. + c_1 \text{RANK}(X, \varepsilon_\ell^i)_1 + c_2 \text{RANK}(X, \varepsilon_\ell^i)_1 + 3 \right). \tag{2.30}
 \end{aligned}$$

357 Since some of these flops are performed in low precision and some in high precision, we
 358 must account for the different speeds of different arithmetics. To do so, we ponderate
 359 the low precision flops by a weight $\omega_\ell < 1$ which indicates the relative performance
 360 ratio between the low and the high precision. We obtain

$$\begin{aligned}
 \text{Time IR} &\propto \omega_\ell c_1 p \text{RANK}(X, \varepsilon_\ell)_1 + p \sum_{i=1}^{n_{\text{it}}} \left(\omega_\ell c_1 \text{RANK}(X, \varepsilon_\ell^{i+1}/2)_1 \right. \\
 &\quad \left. + \omega_\ell c_1 \text{RANK}(X, \varepsilon_\ell^i)_1 + c_2 \text{RANK}(X, \varepsilon_\ell^i)_1 + 3 \right), \tag{2.31}
 \end{aligned}$$

362 where the ‘‘Time’’ formula should be taken as a rough estimator of the time perfor-
 363 mance of the algorithm, although in practice the actual execution time depends on a
 364 number of other complex factors such as the arithmetic intensity of the operations,
 365 the data locality, and the parallelism.

366 This cost is to be compared with the cost of simply computing $\text{LRA}(X, \varepsilon)$ in the
 367 high precision u , given by

$$\text{Time Ref.} \propto c_1 p \text{RANK}(X, \varepsilon)_1. \tag{2.32}$$

369 This complexity analysis reveals two situations where [Algorithm 2.1](#) can outper-
 370 form the uniform precision approach.

371 *Numerical ranks $\text{RANK}(X, \varepsilon_\ell^i)$ rapidly decreasing as ε_ℓ increases.* The first situa-
 372 tion is when the numerical ranks of X at accuracy lower than the final target ε are
 373 much smaller than $\text{RANK}(X, \varepsilon)$. This can certainly be the case in some applications.
 374 For example, in the case of matrices, the numerical rank of X at any given accuracy
 375 is determined by its singular values. If the singular values decay rapidly, $\text{RANK}(X, \varepsilon_\ell)$
 376 will in general be significantly smaller than $\text{RANK}(X, \varepsilon)$. The most extreme exam-
 377 ple is a matrix with one large singular value and all the remaining $\text{RANK}(X, \varepsilon) - 1$
 378 singular values just above $\varepsilon \|X\|$. In this extreme case, all the iterations except the
 379 last will only need to compute on rank-1 matrices, so the overall cost of the method
 380 will be dominated by that of the last iteration, which is roughly $\omega_\ell c_1 p \text{RANK}(X, \varepsilon)$,
 381 and so is always smaller than [\(2.32\)](#). In a more realistic setting where the singular
 382 values decay exponentially, we may still expect [\(2.31\)](#) to be smaller than [\(2.32\)](#) even
 383 for reasonably traditional values of ω_ℓ .

384 This situation also extends to tensors, although we do not have such a simple
 385 characterization as one based on singular values. Essentially, if the underlying ma-
 386 trices used in the tensor representation exhibit rapidly decaying singular values, then
 387 the rank vectors $\text{RANK}(X, \varepsilon_\ell)$ will take much smaller values than $\text{RANK}(X, \varepsilon)$ when
 388 $\varepsilon_\ell \gg \varepsilon$.

389 *Very fast low precision arithmetic (or very slow high precision arithmetic).* The
 390 second situation is when the low precision arithmetic is much faster than the high
 391 precision one, that is, when $\omega_\ell \ll 1$. This is becoming increasingly common for
 392 low precisions on modern hardware, especially accelerators. For example, the fp16
 393 and bfloat16 arithmetics can be up to 16 times faster than fp32 arithmetic on recent
 394 NVIDIA GPUs. Similarly, fp16 arithmetic can be up to 8 times faster than fp32 on
 395 the AMD Instinct MI250X GPUs. In this situation, the time cost (2.31) can indeed
 396 be lower than (2.32), even in the worst case scenario where the ranks of all objects
 397 throughout the iterations attain their upper bound of $2 \text{RANK}(X, \varepsilon)$ for E . Indeed, in
 398 this case, (2.31) becomes

$$399 \quad p \text{RANK}(X, \varepsilon)_1 (\omega_\ell c_1 (2n_{\text{it}} + 1) + n_{\text{it}} c_2) + 3p. \quad (2.33)$$

400 Neglecting the $3p$ term, (2.33) is smaller than (2.32) if

$$401 \quad \omega_\ell (2n_{\text{it}} + 1) + n_{\text{it}} c_2 / c_1 \leq 1. \quad (2.34)$$

402 For most LRA algorithms of interest, the cost of compressing (LRA kernel) a full
 403 object is higher than the cost of decompressing it (DECOMPRESS kernel), that is,
 404 $c_1 > c_2$. Therefore (2.34) can certainly be satisfied for sufficiently small ω_ℓ .

405 Moreover, we now describe a case where the DECOMPRESS kernel can also be
 406 performed in low precision. Indeed, while Algorithm 2.1 requires the DECOMPRESS
 407 kernel Line 3 to be performed in high precision, this kernel has the particularity of
 408 taking as input the factors F , which are stored in low precision. Therefore, what we
 409 really need is to compute in high precision with low precision numbers; this happens
 410 to be an easier task than the more general problem of computing in high precision with
 411 high precision numbers. In fact, an increasing range of modern hardware provides the
 412 capability to perform this task inexpensively. For example, the NVIDIA GPUs are
 413 equipped with so-called tensor core units that can carry out matrix multiplication with
 414 half precision (16-bit) matrices at the accuracy of single precision (fp32) arithmetic
 415 but at the much higher speed of half precision arithmetic. Similar instructions are
 416 available on several other architectures, which have been analyzed by Blanchard et
 417 al. [4] under a common framework called block FMA.

418 In our context, assuming the DECOMPRESS kernel can make use of these block
 419 FMA units changes the cost analysis quite significantly since all the dominant oper-
 420 ations are then performed at the speed of the low precision. Even under the same
 421 worst case scenario where the ranks of all objects throughout the iterations attain
 422 their upper bound, (2.34) now becomes

$$423 \quad \omega_\ell (2n_{\text{it}} + 1 + n_{\text{it}} c_2 / c_1) \leq 1. \quad (2.35)$$

424 Thus, if block FMA hardware can be exploited, even the ratio c_2 / c_1 need not be
 425 necessarily small for Algorithm 2.1 to be effective.

426 Clearly, the two situations discussed above are not exclusive, so in practice it is
 427 very possible that we have both smaller ranks and a fast low precision. To further
 428 assess under which condition Algorithm 2.1 can outperform the standard approach,
 429 we now specialize it to specific matrix or tensor algorithms in section 3 and section 4.

430 **3. Application to matrix low-rank approximation.** In this section we spe-
 431 cialize Algorithm 2.1 to the case where $X \in \mathbb{R}^{m \times n}$ is a matrix.

For any unitarily invariant norm, the optimal LRA algorithm for matrices is to
 compute their singular value decomposition (SVD) and truncate it to the desired

accuracy, a result known as Eckart–Young theorem [10]. Specifically, given the SVD $X = U\Sigma V^T$, the optimal rank- k approximation of X is

$$\arg \min_{\substack{M \in \mathbb{R}^{m \times n} \\ \text{RANK}(M)=k}} \|X - M\| = U_k \Sigma_k V_k^T,$$

432 where $U_k \Sigma_k V_k^T$ is the truncated SVD of X , formed by the first k singular vectors and
 433 values only.

434 While the truncated SVD provides the best accuracy, it is expensive due to the
 435 need to compute the full SVD before truncating it: it requires $O(mn^2)$ flops. For this
 436 reason, low-rank matrix approximations are often computed using slightly less accu-
 437 rate but cheaper alternatives. In this paper, we focus on two widely popular choices:
 438 the truncated QR decomposition with column pivoting (QRCP, subsection 3.1), and
 439 the randomized SVD (subsection 3.2). For each algorithm, we discuss their use as
 440 LRA kernel in our mixed precision iterative refinement approach (Algorithm 2.1): in
 441 particular, we check that the algorithm satisfies the assumptions (2.1)–(2.3) of the er-
 442 ror analysis and the assumptions (2.29a)–(2.29c) of the complexity analysis. We also
 443 explain how to perform the RECOMPRESS kernel based on the specific LRA choice.

3.1. Truncated QRCP decomposition. Our first choice of LRA algorithm is
 a truncated QRCP decomposition. QRCP decomposes the original matrix $X \in \mathbb{R}^{m \times n}$
 as

$$XP = QR$$

444 where $Q \in \mathbb{R}^{m \times n}$ is a matrix with orthonormal columns, $R \in \mathbb{R}^{n \times n}$ is an upper
 445 triangular matrix, and $P \in \mathbb{R}^{n \times n}$ is a permutation matrix. Thanks to pivoting, the
 446 QRCP decomposition can be used as a rank-revealing algorithm because the norm
 447 of the trailing submatrix at each step k of the QR factorization, $\|XP - Q_k R_k\|$, is
 448 monotonically decreasing. Therefore, we can stop the QR factorization as soon as
 449 this norm becomes smaller than the target tolerance ε_ℓ . We obtain

$$450 \quad X \approx Q_k V_k^T, \quad Q_k \in \mathbb{R}^{m \times k}, \quad V_k = PR_k^T \in \mathbb{R}^{n \times k}, \quad (3.1)$$

451 where $k = \text{RANK}(X, \varepsilon_\ell)$. To avoid the need to apply the permutation P each time we
 452 need to apply X , we form and store $V_k = PR_k^T$.

453 We can implement a suitable RECOMPRESS algorithm using this truncated QRCP
 454 decomposition. Several versions are possible; we describe in Algorithm 3.1 the one
 455 that we will use in this article. Given a (non-optimal) truncated QRCP decomposition
 456 $Q_{in} V_{in}^T$, Algorithm 3.1 recompresses it into an optimal LRA as follows. First, a thin
 457 QR factorization $\bar{Q}R$ of V_{in} is computed. Then, the product $Q_{in} R^T$ is formed and a
 458 truncated QRCP decomposition $Q_{out} \bar{V}^T$ is computed at the desired target accuracy
 459 ε . This yields the recompressed left factor Q_{out} ; the recompressed right factor V_{out} is
 460 obtained by forming $\bar{Q}\bar{V}$.

We now check that the assumptions (2.1)–(2.3) of the error analysis are satisfied
 for truncated QRCP and determine the value of the constants in the error bounds.
 For (2.1), we can analyze the truncated QRCP by separating the truncation and
 rounding errors. As explained above, we can control the size of the truncation error
 by stopping the QRCP decomposition once the approximation error falls below the
 desired threshold ε_ℓ , so that in exact arithmetic we obtain

$$Q_k R_k = XP + E, \quad \|E\| \leq \varepsilon_\ell \|X\|.$$

Algorithm 3.1 RECOMPRESS algorithm using truncated QRCP decomposition.

Input: a truncated QRCP decomposition $Q_{in}V_{in}^T$ of the form (3.1);
 ε , the target accuracy.

Output: a recompressed QRCP decomposition $Q_{out}V_{out}^T$.

- 1: Compute the thin QR factorization $\bar{Q}R = V_{in}$.
 - 2: Compute the truncated QRCP decomposition $Q_{out}\bar{V}^T = Q_{in}R^T$.
 - 3: $V_{out} \leftarrow \bar{Q}\bar{V}$
-

461 To account for the rounding errors, we use standard analysis of QR decomposition [16,
 462 p. 361], which shows that the computed QR factors satisfy the backward error bound

$$463 \quad \hat{Q}_k \hat{R}_k = XP + \Delta X + E, \quad \|\Delta X\| \leq \sqrt{k} \tilde{\gamma}_{mk} \|X\|, \quad (3.2)$$

464 where $\tilde{\gamma}_{mk} = cmku_\ell / (1 - cmku_\ell) \simeq cmku_\ell$, for a modest constant c independent of the
 465 dimensions of the problem. Note that this bound is valid even with column pivoting,
 466 since computing the QRCP decomposition $XP = QR$ of X is equivalent to computing
 467 the unpivoted QR decomposition of XP . Assumption (2.1) is thus satisfied with
 468 $b_1 \simeq cmk\sqrt{k}$. The DECOMPRESS kernel is a standard matrix multiplication between
 469 the low-rank factors Q and V hence assumption (2.2) is satisfied with $b_2 = k$ by [16,
 470 p. 71]. Finally, to bound the error introduced by the RECOMPRESS kernel we must
 471 analyze Algorithm 3.1. Since the algorithm simply consists of standard QR, QRCP,
 472 and matrix multiplication operations, this analysis is straightforward and relies on
 473 standard error bounds from the literature. Although we omit it here for the sake of
 474 conciseness, we have performed this analysis and found that (2.3) is satisfied with
 475 $b_3 \simeq cn(k^{3/2} + \tilde{k}^{3/2}) + k$, where k is the rank of $Q_{in}V_{in}^T$ before recompression and \tilde{k}
 476 is the rank of $Q_{out}V_{out}^T$ after recompression.

477 Finally, we discuss the cost of Algorithm 2.1 when using truncated QRCP de-
 478 composition. The truncated QRCP decomposition (3.1) can be computed in $4mnk +$
 479 $o(mnk)$ flops [12], so that assumption (2.29a) is satisfied with $c_1 = 4$. The DECOMPRESS
 480 kernel is a matrix multiplication which requires $2mnk$ flops, so that assumption (2.29b)
 481 is satisfied with $c_2 = 2$. For the RECOMPRESS kernel, Algorithm 3.1 requires $(6m +$
 482 $2n)k^2 + o(mk^2 + nk^2)$ flops, so that assumption (2.29c) is satisfied since $(m + n)k^2 =$
 483 $o(mn)$.

484 **3.2. Randomized SVD decomposition.** Our second choice of LRA algorithm
 485 is the randomized SVD decomposition. Several variants have been proposed; in this
 486 article we use the one described in Algorithm 3.2, which was proposed by Mar-
 487 tinsson and Voronin [22].

488 The algorithm consists of two phases. The first phase (Lines 3 to 11) iteratively
 489 builds a QB decomposition of the matrix such that $\|X - QB\| \leq \varepsilon_\ell \|X\|$, where
 490 $Q \in \mathbb{R}^{m \times k}$ has orthonormal columns and where the dimension k is hopefully close
 491 to $\text{RANK}(X, \varepsilon_\ell)$. To do so, a random Gaussian matrix $\Omega \in \mathbb{R}^{n \times b}$ is drawn for a
 492 given block size b and used to sample the matrix $Y = X\Omega$ (Line 5). Then Y
 493 is added to the basis Q while preserving the orthonormality: this can for example be
 494 accomplished using the Gram-Schmidt algorithm (Line 6). This process is repeated
 495 until the columns of Q are a sufficiently good approximation of the column space of
 496 X , that is, until $\|X - QQ^T X\|$ is small. In order to efficiently compute this quantity,
 497 the matrix $B = Q^T X$ is formed block by block (Line 7). Then, in the second phase
 498 (Lines 12 and 13), a truncated SVD of X can easily be obtained by computing the

Algorithm 3.2 Randomized SVD decomposition.

Input: $X \in \mathbb{R}^{m \times n}$, a target accuracy ε_ℓ , and a block size b .

Output: a truncated SVD $U\Sigma V^T$ decomposition of X .

- 1: Initialize Q and B to empty matrices.
 - 2: $n_X = \|X\|$
 - 3: **repeat**
 - 4: Draw a random Gaussian matrix $\Omega \in \mathbb{R}^{n \times b}$.
 - 5: $Y = X\Omega$
 - 6: $Q_b = \mathbf{qr}(Y - Q(Q^T Y))$
 - 7: $B_b = Q_b^T X$
 - 8: $Q \leftarrow [Q \ Q_b]$
 - 9: $B \leftarrow \begin{bmatrix} B \\ B_b \end{bmatrix}$
 - 10: $X \leftarrow X - Q_b B_b$
 - 11: **until** $\|X\| \leq \varepsilon_\ell n_X$
 - 12: Compute the truncated SVD decomposition $B \approx \bar{U}\Sigma V^T$ at accuracy ε_ℓ .
 - 13: $U = Q\bar{U}$
-

499 truncated SVD of the lower dimensional matrix $B \in \mathbb{R}^{k \times n}$: if $B = \bar{U}\Sigma V^T$, then
500 $X = U\Sigma V^T$ with $U = Q\bar{U}$.

501 We chose to use this specific randomized SVD variant because it presents several
502 advantages. It is blocked, which allows for an efficient implementation on modern
503 hardware. Moreover it provides a cheap yet reliable error estimation. Thanks to
504 blocking and error estimation, [Algorithm 3.2](#) can adaptively reveal the numerical
505 rank of the matrix at the requested accuracy ε_ℓ ; no a priori knowledge on the rank is
506 thus necessary. Finally, we have experimentally observed [Algorithm 3.2](#) to be more
507 accurate than other randomized SVD variants that we have tested.

508 In order to perform the RECOMPRESS operation using randomized SVD, [Algo-](#)
509 [rithm 3.2](#) needs to be slightly adapted. The algorithm takes as input a (non-optimal)
510 low-rank matrix $X = U\Sigma V^T$ and seeks to recompress it, without forming the full
511 X . [Lines 5](#) and [7](#) of [Algorithm 3.2](#) are matrix products and can thus exploit the low-
512 rank structure of X . [Line 10](#) is a subtraction between two low-rank matrices, which
513 requires no operations (the low-rank factors can simply be concatenated). The only
514 difficulty lies on [Line 11](#), which requires to compute $\|X\|$ to control the error and stop
515 the algorithm. In order to compute $\|X\|$ without forming X , we orthonormalize one
516 of the two low-rank factors and compute the norm of the other one.

517 We now check if the assumptions [\(2.1\)](#)–[\(2.3\)](#) of the error analysis are satisfied for
518 the randomized SVD. To do so, we rely on the error analysis of Connolly, Higham,
519 and Pranesh [\[7\]](#), which determines error bounds for [Algorithm 3.2](#) in floating-point
520 arithmetic. By [\[7, Corollary 2.4\]](#), assumption [\(2.1\)](#) is satisfied with $b_1 \simeq \sqrt{nmk}$.
521 Assumption [\(2.2\)](#) is satisfied with $b_2 = k + 1$ since decompressing $U\Sigma V^T$ can be
522 achieved with a matrix–matrix product and a scaling. Finally, the analysis of [\[7\]](#) does
523 not directly apply to the RECOMPRESS version of the algorithm discussed above, but
524 we expect its numerical behavior to be similar to the original algorithm.

525 Finally, we discuss the cost of [Algorithm 3.2](#). If X is a full $m \times n$ matrix, [Al-](#)
526 [gorithm 3.2](#) costs $6mnk + o(mnk)$ flops [\[22, Eqn. \(25\)\]](#), so that assumption [\(2.29a\)](#)
527 is satisfied with $c_1 = 6$. It is easy to extend [\[22, Eqn. \(25\)\]](#) to the case where
528 $X = U\Sigma V^T$ is represented under low-rank form to check that the RECOMPRESS vari-

529 ant of [Algorithm 3.2](#) described above has a cost in $O((m+n)k^2) = o(mn)$ flops, so
 530 that assumption [\(2.29c\)](#) is indeed satisfied. Finally, assumption [\(2.29b\)](#) is satisfied
 531 with $c_2 = 2$ since the DECOMPRESS kernel is simply a matrix–matrix product.

532 **4. Application to tensor low-rank approximation.** In this section, we ex-
 533 plore the application of our method to three different tensor decompositions, namely
 534 Tucker [\[29\]](#), TT [\[27\]](#), and HT [\[13\]](#). All three decompositions provide direct methods
 535 for the LRA and RECOMPRESS kernels that can guarantee a prescribed accuracy ε , as
 536 well as fast DECOMPRESS routines, rendering them amenable to [Algorithm 2.1](#). We
 537 first give a brief description of these decompositions as well as their corresponding
 538 LRA, RECOMPRESS, and DECOMPRESS kernels, then provide an error and complexity
 539 analysis with respect to the assumptions [\(2.1\)–\(2.3\)](#) and [\(2.29a\)–\(2.29c\)](#).

540 **4.1. Tucker, TT, and HT decompositions.** The Tucker decomposition rep-
 541 represents a tensor $X \in \mathbb{R}^{n_1 \times \dots \times n_d}$ with a core tensor $G \in \mathbb{R}^{r_1 \times \dots \times r_d}$ and with factor ma-
 542 trices $U = \{U^{(i)}\}$ with orthonormal columns, where $U^{(i)} \in \mathbb{R}^{n_i \times r_i}$ for $i \in \{1, \dots, d\}$.
 543 Each element of a Tucker tensor is given by

$$544 \quad X_{i_1, \dots, i_d} = \sum_{\alpha_1, \dots, \alpha_d}^{r_1, \dots, r_d} G_{\alpha_1, \dots, \alpha_d} U_{i_1, \alpha_1}^{(1)} \dots U_{i_d, \alpha_d}^{(d)}. \quad (4.1)$$

545 The TT format consists of a sequence of 3rd-order tensors $G^{(i)}$ linked by the first
 546 and third dimensions that correspond to ranks. Each element of a TT tensor is given
 547 by

$$548 \quad X_{i_1, \dots, i_d} = \sum_{\alpha_0, \dots, \alpha_d}^{r_0, \dots, r_d} G_{\alpha_0, i_1, \alpha_1}^{(1)} \dots G_{\alpha_{d-1}, i_d, \alpha_d}^{(d)}. \quad (4.2)$$

549 The HT decomposition represents X as a binary tree of tensors whose leaves are
 550 matrices $U^{(i)} \in \mathbb{R}^{n_i \times r_i}$ and the rest are 3rd order tensors. Each element of an HT
 551 tensor is given by

$$552 \quad X_{i_1, \dots, i_d} = \sum_{\alpha_0, \dots, \alpha_{2d-2}}^{r_0, \dots, r_{2d-2}} G_{\alpha_0, \alpha_1, \alpha_2}^{(1)} \dots G_{\alpha_{d-2}, \alpha_{2d-3}, \alpha_{2d-2}}^{(d-1)} U_{i_1, \alpha_{d-1}}^{(1)} \dots U_{i_d, \alpha_{2d-2}}^{(d)}. \quad (4.3)$$

553 We illustrate these decompositions in [Figure 4.1](#) using a tensor network dia-
 554 gram [\[5\]](#) where each circle represents a tensor and each edge corresponds to a di-
 555 mension. For each decomposition, the corresponding LRA method yields a network
 556 of core tensors (or just “cores”), whose contraction/multiplication along the inner
 557 dimensions (of size r_i , $i = 1 : \bar{d}$) yields the full tensor. The outer dimensions (of size
 558 n_i , $i = 1 : d$) correspond to dimensions of the full tensor X .

559 The HOSVD [\[29\]](#), TTSVD [\[27\]](#), and HTSVD [\[13\]](#) algorithms can be used as LRA
 560 kernels to compute the Tucker, TT, and HT decompositions of a given full tensor X ,
 561 respectively. All three are direct methods that can achieve any prescribed accuracy
 562 ε in exact arithmetic. To the best of our knowledge, the effect of rounding errors on
 563 these methods in finite precision arithmetic has not been analyzed in the literature.
 564 While these methods are therefore not known to be stable, empirical experiments
 565 suggest that they still reliably achieve an accuracy of order ε when run in a finite
 566 precision with a unit roundoff u sufficiently smaller than ε . Formally proving this
 567 result is an open problem that deserves a dedicated study.

568 We provide a high level pseudo-code that encapsulates these methods in [Algo-](#)
 569 [rithm 4.1](#). In essence, all three algorithms perform \bar{d} (defined in [\(1.1\)](#)) successive SVDs

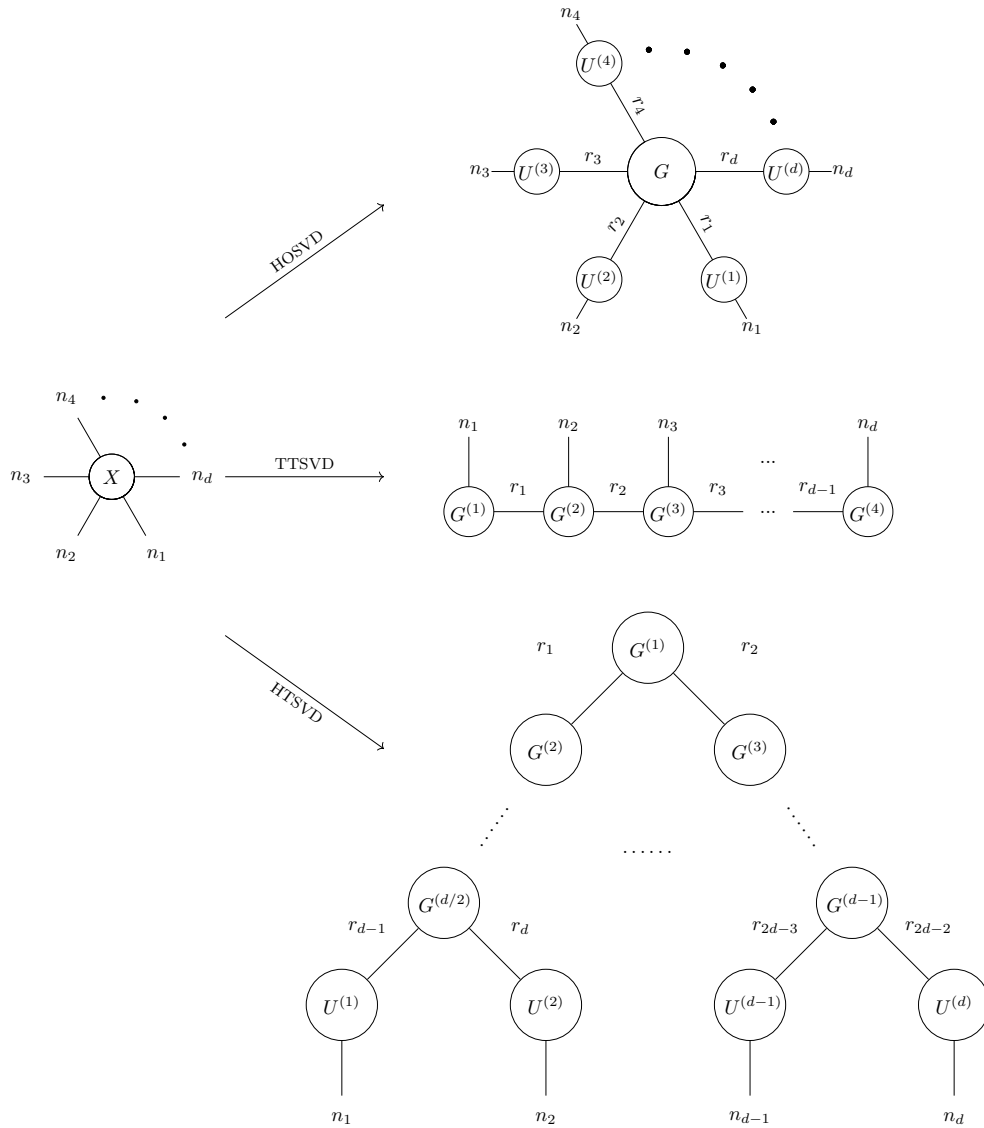


Fig. 4.1: A d -order tensor (left) and its representation in Tucker, TT, and HT decompositions (right) using tensor network diagram.

570 on matricizations of X to isolate each core of the decomposition and determine its
 571 corresponding rank, as indicated on [Line 4](#). Then, the left orthogonal component U
 572 of the SVD is reshaped and added to the decomposition F_X , and the non-orthogonal
 573 component is reshaped into a tensor to be further factored in the subsequent iterations.
 574 At the end, the decomposed tensor F_X consisting of $\bar{d} + 1$ tensors is returned.

575 As the LRA kernel isolates the cores through a series of \bar{d} SVDs on X , the
 576 DECOMPRESS kernel performs successive tensor contractions on neighboring cores of
 577 F_X through the inner dimensions that link them to obtain the full tensor X . In [Algo-](#)
 578 [rithm 4.2](#), we provide a generic DECOMPRESS kernel that performs these contractions

Algorithm 4.1 LRA algorithm for a tensor.**Input:** a tensor X and a target accuracy ε .**Output:** F_X : a low-rank tensor decomposition of X .

```
1:  $\bar{\varepsilon} \leftarrow \varepsilon/\bar{d}$ 
2:  $F_X \leftarrow []$ 
3: for  $i = 1$  to  $\bar{d}$  do
4:    $[U, Y] \leftarrow \text{SVD}(\text{MATRICIZE}(X), \bar{\varepsilon})$ 
5:    $F_X \leftarrow [F_X, \text{RESHAPE}(U)]$ 
6:    $X \leftarrow \text{RESHAPE}(Y)$ 
7: end for
8:  $F_X \leftarrow [F_X, \text{RESHAPE}(X)]$ 
```

Algorithm 4.2 DECOMPRESS algorithm for a tensor decomposition.**Input:** a low-rank tensor decomposition F_X .**Output:** the full tensor X corresponding to F_X .

```
1:  $X \leftarrow F_X[1]$ 
2: for  $i = \bar{d}$  to 1 in reverse order do
3:    $X \leftarrow X \times_{\{r_i\}} F_X[i + 1]$ 
4: end for
```

579 in the reverse order of SVDs in [Algorithm 4.1](#) for simplicity. This is done on [Line 3](#)
580 where $\times_{\{r_i\}}$ represents the contraction/multiplication across the inner dimension of
581 size r_i . In practice, these contractions could be performed in an arbitrary order among
582 the \bar{d} inner dimensions as tensor contraction is associative.

583 Finally, the RECOMPRESS kernel of the TT decomposition consists of an orthog-
584 onalization and a rank pruning phase [27]. The goal of the orthogonalization phase is
585 to “sweep” all non-orthogonal components in F_X into a single (the first or the last)
586 core. To do this, a QR factorization is performed on a matricization of the core at
587 the opposite end. The orthogonal component Q is reshaped to replace the treated
588 core, and the non-orthogonal component R is passed over to the next core through
589 a tensor–matrix contraction. This process is repeated \bar{d} times until all cores except
590 one become orthogonal, which finalizes the orthogonalization phase. Next, the rank
591 pruning phase performs an SVD on a matricization of the non-orthogonal core to
592 reduce the size of an inner dimension. The orthogonal part U obtained from the SVD
593 is kept as the pruned core whereas the non-orthogonal part ΣV^T is integrated into
594 the next core to be truncated through tensor-matrix contraction, thereby still keep-
595 ing all cores except one orthogonal. The process is similarly repeated on all \bar{d} inner
596 dimensions in the reverse order of orthogonalization, which finalizes the rank pruning
597 phase. The standard RECOMPRESS kernel in the HT format [13] similarly sweeps all
598 non-orthogonal components into a single core (the root of the tree) with a sequence
599 of QR factorizations followed by matrix multiplies through a bottom-up traversal of
600 the tree. However, in the rank pruning phase, it processes inner cores level-by-level
601 starting from the root, and performs the rank truncation on the matricization of the
602 tensor sub-network involving all tensors in the path from the root down to the inner
603 core that is being truncated. Though this variant remains efficient (thanks to the
604 re-use of partial tensor contractions from the previous level through memorization)
605 and provides more coarse-grain parallelism (with respect to TTSVD for instance, as
606 each node in a tree level can be truncated independently), it requires to form a Gram

607 matrix to remain efficient when performing rank truncation (otherwise the size of the
608 matricized tensor explodes, yet its Gramian stays compact). We observed that this
609 method greatly suffers from numerical instability due to this inherent limitation, par-
610 ticularly when soliciting a high precision in RECOMPRESS, as it squares the condition
611 number of the matricized tensor. We instead devised an alternative RECOMPRESS
612 kernel, reminiscent of that of the TT decomposition, for HT and Tucker decomposi-
613 tions. This variant similarly involves an orthogonalization and rank pruning through
614 \bar{d} QRs and SVDs followed by matrix multiplies, albeit with no Gram matrix form-
615 ing, and proved to be much more robust numerically yet still efficient. We defer the
616 detailed presentation of these algorithms and their numerical and cost analysis to a
617 future work. Due to these intricacies, we do not provide a high level RECOMPRESS
618 kernel encapsulating all three decompositions; however, for the purposes of this pa-
619 per, we only use the fact that all three decompositions provide a RECOMPRESS kernel
620 performing \bar{d} QR and SVD computations followed by matrix multiplies on its cores,
621 which suffices for our analysis in the next section.

622 **4.2. Error and complexity analysis.** For the error analysis, as previously
623 mentioned the stability of tensor computations in finite precision arithmetic is an
624 open problem that deserves a dedicated study. However, we may mention that the
625 RESHAPE and MATRICIZE operations do not introduce any error as they merely in-
626 volve data shuffling, nor do they have any impact on the Frobenius norm of the
627 successive matricizations. Moreover, the LRA kernel is based on a chain of SVDs, the
628 DECOMPRESS kernel is based on a chain of matrix multiplies, and the RECOMPRESS
629 kernel on a chain of QRs, SVDs, and matrix multiplies. All of these basic linear
630 algebra building blocks have stable implementations and so we can reasonably ex-
631 pect the numerical behavior of the overall tensor computation to satisfy assumptions
632 (2.1)–(2.3).

633 For the complexity analysis of the LRA step, we assume that the first SVD re-
634 duces the size of the tensor significantly, that is, to $o(p)$, which renders the cost of
635 subsequent SVDs negligible. In this case, we can use the same constant $c_1 = 6$ as
636 in the case of randomized SVD in subsection 3.2. With the same assumption, the
637 cost of DECOMPRESS in Algorithm 4.2 will also be dominated by the last contraction,
638 which is essentially a matrix multiplication on permuted tensors across the first inner
639 dimension. Thus, we can similarly use the constant $c_2 = 2$ as in subsection 3.2, since
640 the cost of the previous contractions is in $o(p)$ in this case as well. Even without this
641 assumption, we can find constants $c_1 = 6t$ and $c_2 = 2t$ with the same $t > 1$, since
642 intermediate tensor sizes in each SVD and contraction steps in Algorithm 4.1 and Al-
643 gorithm 4.2 stay the same. This keeps the cost ratio of these two steps constant across
644 all \bar{d} dimensions yielding the same t for c_1 and c_2 ; we skip further details for brevity.
645 Finally, the cost of RECOMPRESS remains in $o(p)$ as it involves QRs and SVDs on the
646 matricizations of 2nd or 3rd order cores in the decomposition, whose size is in $o(p)$
647 by the low-rank assumption. Therefore, we satisfy the assumptions (2.29a)–(2.29c) of
648 the complexity analysis in the tensor case.

649 5. Numerical experiments.

650 **5.1. Experimental setting.** We now test our iterative refinement approach
651 experimentally. We developed a MATLAB code that implements Algorithm 2.1 and
652 can use as LRA kernel any of the matrix and tensor LRA algorithms discussed in the
653 previous two sections: QRCP, randomized SVD, HOSVD, TTSVD, and HTSVD.

654 For the matrix algorithms (QRCP and randomized SVD), we use our own imple-

655 mentation. For the tensor algorithms (HOSVD, TTSSVD, and HTSVD), we rely on the
 656 implementations from the libraries described in [3], [26], and [21], respectively, with
 657 some adjustments. We use MATLAB version R2019a throughout the experiments.

658 In the experiments, the high precision u is set to double precision (fp64 arithmetic,
 659 with unit roundoff $u = 2^{-53} \approx 1 \times 10^{-16}$) and the use of various low precisions u_ℓ is
 660 simulated with the chop library of Higham and Pranesh [18]: single precision (fp32
 661 arithmetic, with unit roundoff $u_\ell = 2^{-24} \approx 6 \times 10^{-8}$) and half precision (fp16 and
 662 bfloat16 arithmetics, with unit roundoff $u_\ell = 2^{-11} \approx 5 \times 10^{-4}$ and $u_\ell = 2^{-8} \approx 4 \times 10^{-3}$,
 663 respectively).

664 For the low-rank truncation thresholds, we set $\varepsilon = 10^{-13}$ and $\varepsilon_\ell = \theta_\ell u_\ell$, where
 665 $\theta_\ell \leq 1$ is a scaling factor necessary to control the effect of rounding errors on the
 666 ability of the LRA to detect the correct numerical rank. We analyze in detail the role
 667 of this parameter θ_ℓ in subsection 5.3; based on the conclusions of this analysis, we
 668 have set $\theta_\ell = 2^{-1}$ for all LRA kernels except randomized SVD and QRCP, for which
 669 we have used $\theta_\ell = 2^{-2}$ and $\theta_\ell = 2^{-3}$, respectively.

670 To test the algorithms, we use randomly generated matrices and tensors with
 671 various singular value distributions. More precisely, we compare three types of distri-
 672 butions for the singular values σ_i :

$$673 \quad \sigma_i = \max(\widehat{\sigma}_i, 10^{-16}), \quad \widehat{\sigma}_i = \begin{cases} 1/i & \text{(linear)} \\ i^{-10} & \text{(power)} \\ e^{-i} & \text{(exponential)} \end{cases}. \quad (5.1)$$

674 These three distributions are illustrated in Figure 5.1. We generate the matrices as
 675 $Q_1 \Sigma Q_2$ where Q_1, Q_2 are random orthogonal matrices and Σ is a diagonal matrix
 676 with the specified singular values as coefficients. For the tensor experiments involving
 677 Tucker and TT decompositions, we generate the low-rank tensor in the Tucker format
 678 with $Q^{(1)}, \dots, Q^{(d)}$ random orthonormal matrices and a d th-order core tensor G whose
 679 coefficient G_{i_1, \dots, i_d} is given by $\sigma_{\max(i_1, \dots, i_d)}$ in (5.1). For the experiments involving the
 680 HT format, we generate the low-rank tensor in the HT format whose leaf nodes are
 681 matrices with orthogonal columns, and each element $G_{i,j,k}^{(t)}$ of its internal cores is set
 682 to $\sigma_{\max(i,j,k)}$.

683 **5.2. Experimental results.** In the first experiment, we analyze the behavior
 684 of Algorithm 2.1 for the matrix case, which is provided in Figure 5.2. We consider
 685 three types of matrices based on their singular value distribution (linear, power, and
 686 exponential) and two LRA kernels (QRCP and randomized SVD). In each case, we
 687 plot the relative error

$$688 \quad \eta_i = \frac{\|X - F_i\|}{\|X\|} \quad (5.2)$$

689 where F_i is the computed low-rank factor of X after i refinement steps (F_0 is thus a
 690 standard LRA of X in precision u_ℓ). The number next to each marker indicates the
 691 rank of F_i after recompression.

692 Figure 5.2 shows that for all three matrices and for both QRCP and randomized
 693 SVD, Algorithm 2.1 behaves as expected: the error η_i is roughly equal to $\varepsilon_\ell^{i+1} =$
 694 $(\theta u_\ell)^{i+1}$ after i refinement steps. Thus, using single precision as the low precision u_ℓ ,
 695 we can achieve an accuracy close to double precision with only one refinement step.
 696 Naturally, since for QRCP $\theta = 2^{-3}$ is relatively small, $(u_\ell/\theta)^2$ is noticeably larger
 697 than the accuracy achieved by a standard double precision LRA; the refined factors
 698 are thus not completely as accurate as if computed directly in double precision. This

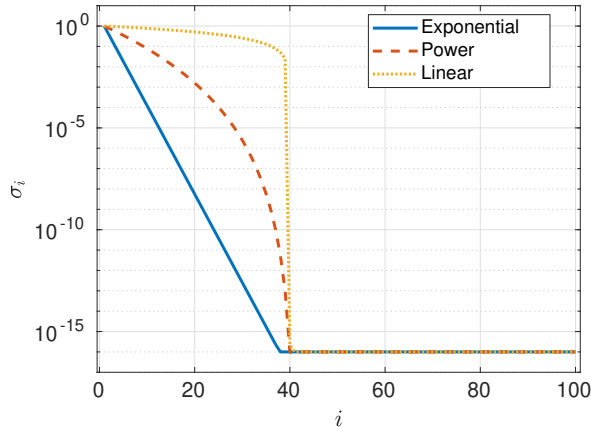


Fig. 5.1: Three types of singular value distributions used in the experiments.

699 gap can be filled by performing a second refinement step, although this would likely
700 be unnecessary in most practical scenarios.

701 Similar results are obtained using half precision as the low precision format (fp16
702 or bfloat16 arithmetics). An error close to the single precision accuracy can be
703 achieved in just one or a few steps. Moreover, we can even reach double precision
704 accuracy if needed, which illustrates an attractive property of [Algorithm 2.1](#): it can
705 reach an accuracy essentially as high as desired while performing most of its operations
706 in a precision as low as desired.

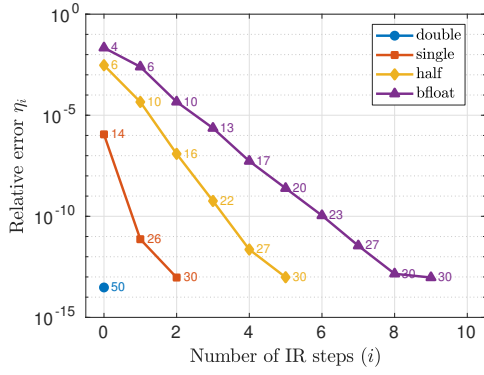
707 Finally, we discuss the rank behavior of F_i across IR steps. We see that it is
708 roughly equal to $\text{RANK}(X, \eta_i)$, the numerical rank of X at accuracy η_i . Thus, for ma-
709 trices with rapidly decaying singular values such as the exponential case in [Figure 5.1](#),
710 these ranks tend to be much smaller at the early steps in low precision. Reflecting on
711 the cost of the algorithm, we can expect the use of low precision plus refinement to
712 be particularly cost efficient for such matrices.

713 [Figure 5.3](#) shows similar plots for the tensor decompositions (TTSVD, HOSVD,
714 and HTSVD). The results for tensors follow the same trend, and lead to the same
715 conclusion as for matrices. The main difference is that the rank of F_i (text labels) is
716 now a vector instead.

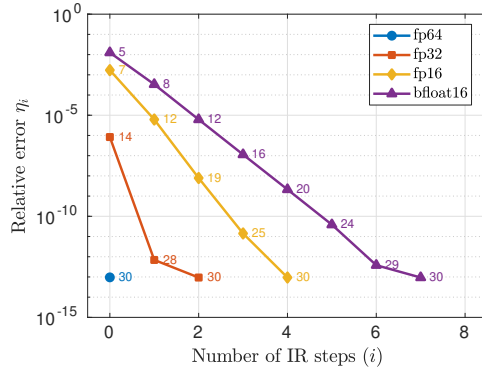
717 Overall, our experiments confirm that [Algorithm 2.1](#) is able to rapidly converge
718 to a high accuracy, while using low precision for the LRA kernel. This observation is
719 valid for all types of matrices and tensors in our test set, and all five LRA kernels that
720 we tested, which shows that [Algorithm 2.1](#) is robust and can work in a wide variety
721 of settings.

722 **5.3. Role of θ_ℓ .** There is a tradeoff in choosing the scaling factor θ_ℓ : the larger
723 it is, the faster the convergence (since the error is reduced by a factor roughly equal to
724 $\theta_\ell u_\ell$ at each refinement step), but if it is too large, the rank will no longer be correctly
725 detected and this will lead to a significant rank growth.

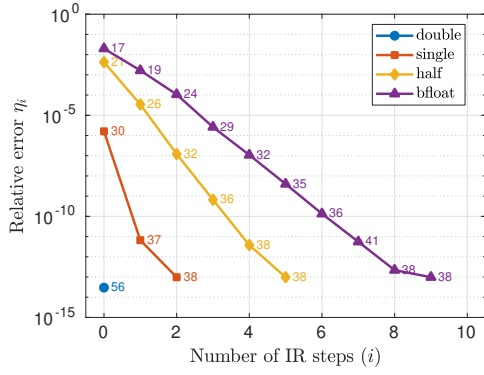
726 We illustrate this in [Table 5.1](#) and [Table 5.2](#), where we compare the convergence
727 of [Algorithm 2.1](#) for different values of θ_ℓ . [Table 5.1](#) is for the TTSVD kernel and [Ta-](#)
728 [ble 5.2](#) is for the QRCP kernel; fp16 arithmetic is used as the low precision u_ℓ in both
729 cases. The tables show that the method converges faster as θ_ℓ increases, as expected:



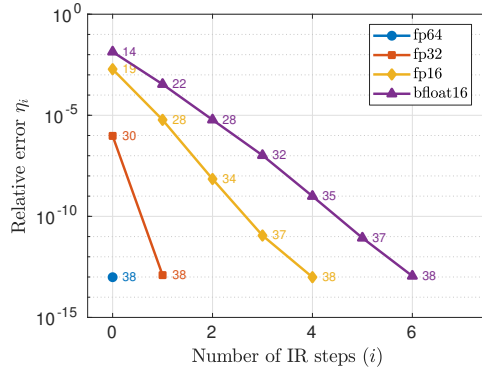
(a) QRCP (exponential).



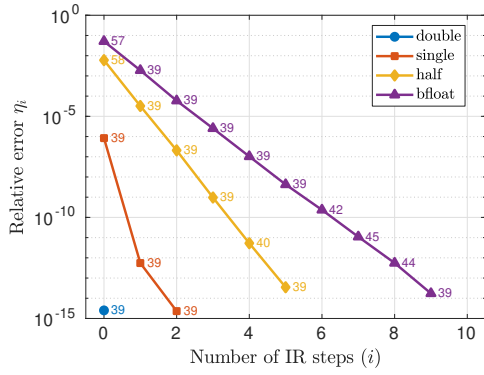
(b) Randomized SVD (exponential).



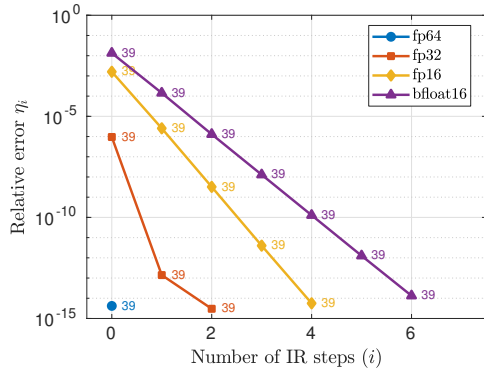
(c) QRCP (power).



(d) Randomized SVD (power).

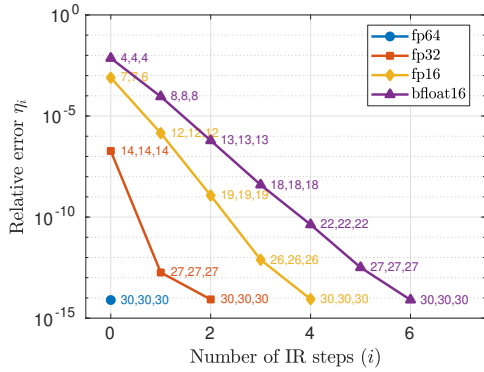


(e) QRCP (linear).

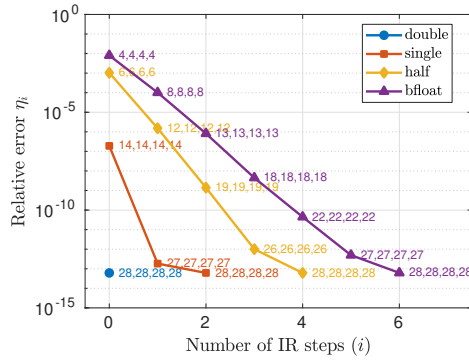


(f) Randomized SVD (linear).

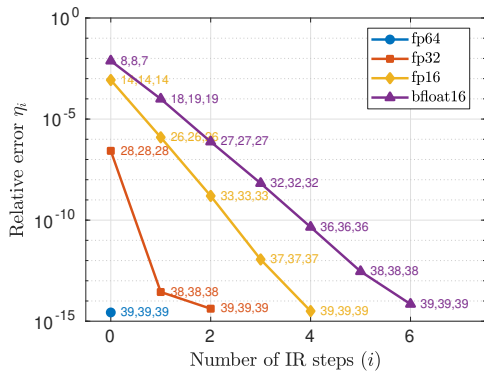
Fig. 5.2: Convergence of Algorithm 2.1 for three types of matrices (with different singular value distributions, see Figure 5.1) and for two different LRA kernels (QRCP or randomized SVD).



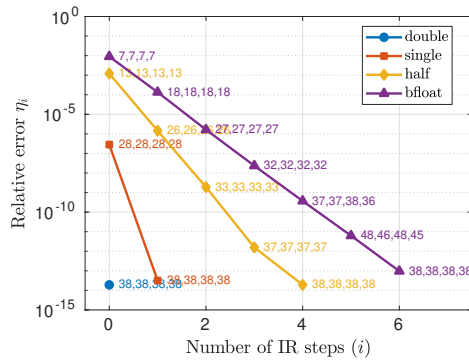
(a) TTSVD (exponential).



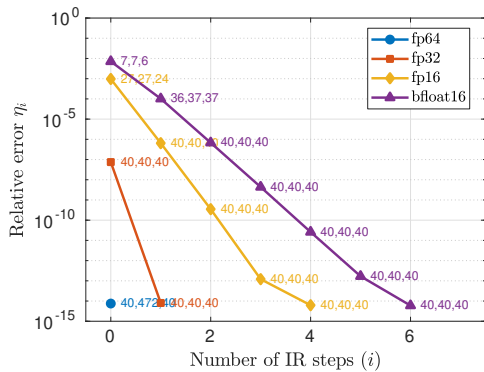
(b) HOSVD (exponential).



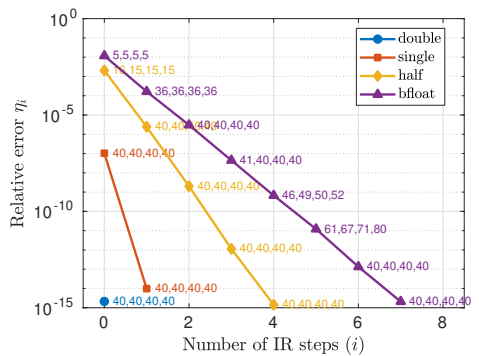
(c) TTSVD (power).



(d) HOSVD (power).



(e) TTSVD (linear).



(f) HOSVD (linear).

Fig. 5.3: Convergence of Algorithm 2.1 for three types of tensors (depending on the singular value distribution, see Figure 5.1) and with TTSVD or HOSVD as LRA kernel.

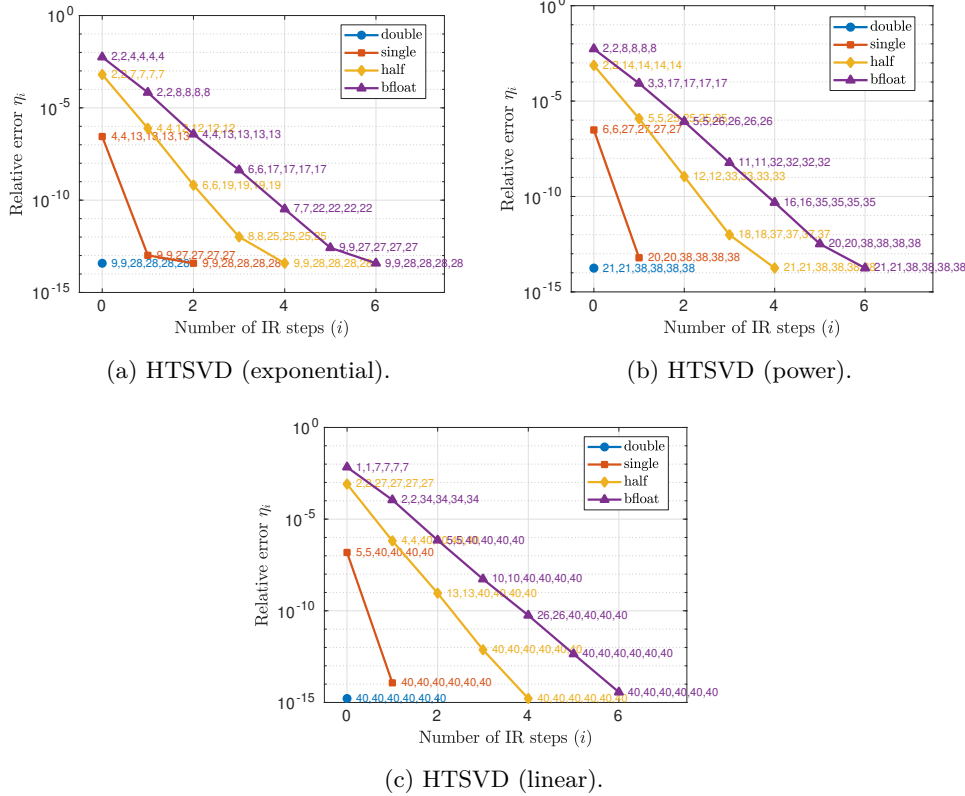


Fig. 5.4: Convergence of Algorithm 2.1 for three types of tensors (depending on the singular value distribution, see Figure 5.1) and with HTSVD as LRA kernel.

730 the error η_i is smaller for larger values of θ_ℓ . For TTSVD (Table 5.1), this faster convergence is achieved without compromising the correct rank detection, which remains
 731 contained throughout the iterations and for all values of $\theta_\ell \leq 1$. Thus, in this case,
 732 a large value of θ_ℓ is recommended. The situation is different for QRCP (Table 5.2),
 733 for which a too large value of θ_ℓ (here $\theta_\ell \geq 2^{-2}$) prevents the rank to be correctly
 734 detected, thus leads to a rank explosion.

736 We therefore conclude that the optimal choice of θ_ℓ depends on the LRA algo-
 737 rithm, and more specifically, on its sensitivity to rounding errors when detecting the
 738 numerical rank. Empirically, we have observed QRCP to be the most sensitive of the
 739 LRA kernels, and to a lesser extent the randomized SVD kernel; the other kernels be-
 740 haved well even for large θ_ℓ . For this reason, in our experiments we have set $\theta_\ell = 2^{-1}$
 741 for all kernels except QRCP and randomized SVD, for which we have set $\theta_\ell = 2^{-3}$
 742 and $\theta_\ell = 2^{-2}$, respectively. This setting allows to keep the ranks contained except for
 743 a few sporadic cases when using half precision.

744 **6. Conclusion.** We have presented a new mixed precision iterative refinement
 745 algorithm for computing low-rank matrix and tensor approximations. The algorithm
 746 first computes a low-rank approximation in low precision, and then computes another
 747 low-rank approximation of the error term, also in low precision, to refine the accuracy

Table 5.1: Relative error η_i and $\text{RANK}(F_i)$ at different steps i and for different values of θ_ℓ , for TTSVD (using fp16 as u_ℓ and exponential distribution of singular values).

$i \setminus \theta_\ell$	Relative error η_i					RANK(F_i)				
	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
0	6e-04	8e-04	9e-04	2e-03	5e-03	23,39,16	07,07,06	06,06,06	05,05,05	04,04,04
1	3e-07	1e-06	4e-06	1e-05	9e-05	14,14,15	12,12,12	11,11,11	10,10,10	08,08,08
2	2e-10	1e-09	8e-09	7e-08	6e-07	21,21,21	19,19,19	17,17,17	15,15,15	13,13,13
3	9e-14	8e-13	2e-11	4e-10	4e-09	29,29,29	26,26,26	23,23,23	20,20,20	18,18,18
4		6e-14	6e-14	1e-12	4e-11		28,28,28	28,28,28	26,26,26	22,22,22
5				6e-14	3e-13				28,28,28	27,27,27
6					6e-14					28,28,28

Table 5.2: Relative error η_i and $\text{RANK}(F_i)$ at different steps i and for different values of θ , for QRCP decomposition (using fp16 as u_ℓ and exponential distribution of singular values).

$i \setminus \theta_\ell$	Relative error η_i					RANK(F_i)				
	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
0	2e-03	2e-03	2e-03	2e-03	3e-03	13	9	9	7	6
1	2e-05	1e-05	1e-05	5e-05	1e-04	25	18	13	10	9
2	5e-08	8e-08	7e-08	1e-07	1e-06	72	43	20	16	14
3	2e-10	2e-10	4e-10	8e-10	8e-09	95	86	36	21	19
4	2e-12	2e-12	3e-12	2e-12	4e-11	27	27	29	27	24
5	7e-13	7e-13	7e-13	7e-13	1e-12	28	28	28	28	27
6					7e-13					28

748 of the approximation. The process can be repeated to further refine the accuracy,
749 and we ensure the rank of the approximation remains bounded by using inexpensive
750 recompression operations. We have performed the error analysis of this algorithm,
751 which proves that the low precision determines the convergence speed, whereas the
752 attainable accuracy only depends on the high precision. Therefore, any desired level
753 of accuracy can be attained, even though most of the operations are performed in
754 low precision. This makes the algorithm computationally attractive, and we have
755 performed a complexity analysis to determine specific conditions under which we can
756 expect it to be cheaper than simply computing a low-rank approximation directly in
757 high precision. We have applied our algorithm to various matrix and tensor low-rank
758 approximations algorithms, and performed MATLAB experiments that confirm its
759 robustness and convergence in a wide range of settings.

760 This paper lays the theoretical and algorithmic foundations for a mixed precision
761 iterative refinement framework for matrix and tensor low-rank approximations. In
762 future work, we will develop high performance implementations of these algorithms
763 to tackle the low-rank approximation of large scale matrices and tensors on modern
764 parallel architectures, especially those equipped with GPU accelerators, which provide
765 extremely fast low precision arithmetic.

766 **7. Acknowledgements.** This work was performed thanks to generous grants
767 from ANR (JCJC SELESTE, ANR-20-CE46-008-01) and Paris Ile-de-France Region
768 (DIM RFSI RC-TENSOR No 2021-05).

- 770 [1] E. AGULLO, O. COULAUD, L. GIRAUD, M. IANNACITO, G. MARAIT, AND N. SCHENKELS, *The*
771 *backward stable variants of GMRES in variable accuracy*, (2022), [https://inria.hal.science/](https://inria.hal.science/hal-03776837)
772 [hal-03776837](https://inria.hal.science/hal-03776837).
- 773 [2] P. AMESTOY, O. BOITEAU, A. BUTTARI, M. GEREST, F. JÉZÉQUEL, J.-Y. L'EXCELLENT, AND
774 T. MARY, *Mixed Precision Low Rank Approximations and their Application to Block Low*
775 *Rank LU Factorization*, IMA J. Numer. Anal., (2022), [https://doi.org/10.1093/imanum/](https://doi.org/10.1093/imanum/drac037)
776 [drac037](https://doi.org/10.1093/imanum/drac037).
- 777 [3] B. W. BADER AND T. G. KOLDA, *Algorithm 862: MATLAB tensor classes for fast algorithm*
778 *prototyping*, ACM Trans. Math. Software, 32 (2006), pp. 635–653, [https://doi.org/10.1145/](https://doi.org/10.1145/1186785.1186794)
779 [1186785.1186794](https://doi.org/10.1145/1186785.1186794).
- 780 [4] P. BLANCHARD, N. J. HIGHAM, F. LOPEZ, T. MARY, AND S. PRANESH, *Mixed precision block*
781 *fused multiply-add: Error analysis and application to GPU tensor cores*, SIAM J. Sci.
782 *Comput.*, 42 (2020), pp. C124–C141, <https://doi.org/10.1137/19M1289546>.
- 783 [5] A. CICHOCKI, N. LEE, I. OSELEDETS, A.-H. PHAN, Q. ZHAO, AND D. P. MANDIC, *Tensor*
784 *networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor*
785 *decompositions*, Foundations and Trends® in Machine Learning, 9 (2016), pp. 249–429,
786 <https://doi.org/http://dx.doi.org/10.1561/22000000059>.
- 787 [6] P. COMON, *Tensors: a brief introduction*, IEEE Signal Processing Magazine, 31 (2014),
788 pp. 44–53, <https://doi.org/10.1109/MSP.2014.2298533>.
- 789 [7] M. P. CONNOLLY, N. J. HIGHAM, AND S. PRANESH, *Randomized low rank matrix approximation:*
790 *Rounding error analysis and a mixed precision algorithm*, MIMS EPrint 2022.5, Manchester
791 Institute for Mathematical Sciences, The University of Manchester, UK, July 2022, [http:](http://eprints.maths.manchester.ac.uk/2884/)
792 [//eprints.maths.manchester.ac.uk/2884/](http://eprints.maths.manchester.ac.uk/2884/). Revised March 2023.
- 793 [8] O. COULAUD, L. GIRAUD, AND M. IANNACITO, *A robust gmres algorithm in tensor train format*,
794 arXiv preprint arXiv:2210.14533, (2022), <https://doi.org/10.48550/arXiv.2210.14533>.
- 795 [9] L. DE LATHAUWER, B. DE MOOR, AND J. VANDEWALLE, *A multilinear singular value decom-*
796 *position*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278, [https://doi.org/10.1137/](https://doi.org/10.1137/S0895479896305696)
797 [S0895479896305696](https://doi.org/10.1137/S0895479896305696).
- 798 [10] C. ECKART AND G. YOUNG, *The approximation of one matrix by another of lower rank*,
799 *Psychometrika*, 1 (1936), pp. 211–218, <https://doi.org/10.1007/BF02288367>.
- 800 [11] M. FASI, N. J. HIGHAM, F. LOPEZ, T. MARY, AND M. MIKAITIS, *Matrix multiplication in*
801 *multiword arithmetic: Error analysis and application to GPU tensor cores*, SIAM J. Sci.
802 *Comput.*, 45 (2023), pp. C1–C19, <https://doi.org/10.1137/21m1465032>.
- 803 [12] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, JHU press, 2013, [https://doi.org/](https://doi.org/10.1137/1028073)
804 [10.1137/1028073](https://doi.org/10.1137/1028073).
- 805 [13] L. GRASEDYCK, *Hierarchical singular value decomposition of tensors*, SIAM J. Matrix Anal.
806 *Appl.*, 31 (2010), pp. 2029–2054, <https://doi.org/10.1137/090764189>.
- 807 [14] L. GRASEDYCK, D. KRESSNER, AND C. TOBLER, *A literature survey of low-rank tensor approx-*
808 *imation techniques*, GAMM-Mitteilungen, 36 (2013), pp. 53–78, [https://doi.org/10.1002/](https://doi.org/10.1002/gamm.201310004)
809 [gamm.201310004](https://doi.org/10.1002/gamm.201310004).
- 810 [15] W. HACKBUSCH AND S. KÜHN, *A new scheme for the tensor representation*, Journal
811 of Fourier analysis and applications, 15 (2009), pp. 706–722, [https://doi.org/10.1007/](https://doi.org/10.1007/s00041-009-9094-9)
812 [s00041-009-9094-9](https://doi.org/10.1007/s00041-009-9094-9).
- 813 [16] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and
814 Applied Mathematics, Philadelphia, PA, USA, second ed., 2002, [https://doi.org/10.1137/](https://doi.org/10.1137/1.9780898718027)
815 [1.9780898718027](https://doi.org/10.1137/1.9780898718027).
- 816 [17] N. J. HIGHAM AND T. MARY, *Mixed precision algorithms in numerical linear algebra*, Acta
817 Numerica, 31 (2022), pp. 347–414, <https://doi.org/10.1017/s0962492922000022>.
- 818 [18] N. J. HIGHAM AND S. PRANESH, *Simulating low precision floating-point arithmetic*, SIAM J.
819 *Sci. Comput.*, 41 (2019), pp. C585–C602, <https://doi.org/10.1137/19M1251308>.
- 820 [19] N. KISHORE KUMAR AND J. SCHNEIDER, *Literature survey on low rank approximation of ma-*
821 *trices*, Linear and Multilinear Algebra, 65 (2017), pp. 2212–2244, [https://doi.org/10.1080/](https://doi.org/10.1080/03081087.2016.1267104)
822 [03081087.2016.1267104](https://doi.org/10.1080/03081087.2016.1267104).
- 823 [20] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM Rev., 51
824 (2009), pp. 455–500, <https://doi.org/10.1137/07070111X>.
- 825 [21] D. KRESSNER AND C. TOBLER, *htucker—a MATLAB toolbox for tensors in hierarchical tucker*
826 *format*, Mathicse, EPF Lausanne, (2012).
- 827 [22] P.-G. MARTINSSON AND S. VORONIN, *A randomized blocked algorithm for efficiently computing*
828 *rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507,
829 <https://doi.org/10.1137/15M1026080>.

- 830 [23] R. OOI, T. IWASHITA, T. FUKAYA, A. IDA, AND R. YOKOTA, *Effect of mixed precision computing*
831 *on \mathcal{H} -matrix vector multiplication in BEM analysis*, in Proceedings of the International
832 Conference on High Performance Computing in Asia-Pacific Region, 2020, pp. 92–101,
833 <https://doi.org/10.1145/3368474.3368479>.
- 834 [24] H. OOTOMO AND R. YOKOTA, *Recovering single precision accuracy from tensor cores while*
835 *surpassing the FP32 theoretical peak performance*, Int. J. High Perform. Comput. Appl.,
836 36 (2022), pp. 475–491, <https://doi.org/10.1177/10943420221090256>.
- 837 [25] H. OOTOMO AND R. YOKOTA, *Mixed-precision random projection for RandNLA on tensor*
838 *cores*, 2023, <https://arxiv.org/abs/2304.04612>.
- 839 [26] I. OSELEDETS, *TT-Toolbox v2.2.2*, 2023, <https://github.com/oseledets/TT-Toolbox>.
- 840 [27] I. V. OSELEDETS, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317,
841 <https://doi.org/10.1137/090752286>.
- 842 [28] L. R. TUCKER, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31
843 (1966), pp. 279–311, <https://doi.org/10.1007/BF02289464>.
- 844 [29] N. VANNIEUWENHOVEN, R. VANDEBRIL, AND K. MEERBERGEN, *A new truncation strategy*
845 *for the higher-order singular value decomposition*, SIAM J. Sci. Comput., 34 (2012),
846 pp. A1027–A1052, <https://doi.org/10.1137/110836067>.
- 847 [30] Z. YANG, J. SHAN, AND Z. ZHANG, *Hardware-efficient mixed-precision CP tensor decomposition*,
848 arXiv preprint arXiv:2209.04003, (2022), <https://doi.org/10.48550/arXiv.2209.04003>.