Reducing swamp behavior for the canonical polyadic decomposition problem by rank-1 freezing

Charlotte Vermeylen[1*], Nico Vervliet[1,2] and Lieven De Lathauwer[1,2]

[1]KU Leuven, Department of Electrical Engineering (ESAT), Kasteelpark Arenberg 10 bus 2446, Leuven, B-3001, Belgium.
[2]KULAK, Subfaculty Science and Technology, E. Sabbelaan 53, Kortrijk, B-8500, Belgium.

*Corresponding author(s). E-mail(s): Charlotte.Vermeylen@kuleuven.be;
Contributing authors: Nico.Vervliet@kuleuven.be;
Lieven.DeLathauwer@kuleuven.be;

Abstract

A novel optimization framework is proposed for solving the low-rank tensor approximation problem using the canonical polyadic decomposition (CPD). This can be a difficult optimization problem for certain tensors, e.g., due to degeneracy, i.e., a tensor that can be approximated arbitrarily closely by an ill-conditioned tensor of lower rank. This is one of the phenomena that are encountered in regions of slow convergence, informally known as swamps. Numerical experiments with state-of-the-art optimization algorithms indicate that in a swamp often only a few rank-1 terms are modified while others stagnate. Often, the non-stagnant terms are problematic and form an ill-conditioned decomposition. To address this, we propose to temporarily freeze the stagnant terms. The lower number of terms has several benefits: it simplifies the problem by reducing the number of variables and reduces the cost per iteration significantly. Furthermore, in many cases the residual tensor can be compressed, and an algebraic (re)initialization can be carried out, even if this was not possible for the original tensor. A refinement step can further improve the accuracy if desired. We provide theoretical insights of why terms can stagnate. More specifically, we prove that terms that are close to the solution are not modified anymore in further optimization steps under certain assumptions. Extensive numerical experiments show that our framework greatly facilitates escaping from swamps. The resulting algorithm outperforms current state-of-the-art approaches on difficult-to-decompose tensors, both in accuracy and computation time, and has similar performance on easier problems.

Keywords: tensors, canonical polyadic decomposition, numerical optimization

1 Introduction

Tensors, or multi-way numerical arrays, play a crucial role in various fields, including machine learning, optimization, signal processing, and statistics [1–9]. Working with and analyzing these multi-way structures requires decomposing the tensor into terms that are easier to interpret and more compact to represent. Tensor decompositions are powerful tools that elevate the exponential increases in the number of elements and allow to recover the underlying 'components' of the data. An example of such a representation is the *(canonical) polyadic decomposition ((C)PD*, which writes a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ as a (minimal) sum of R rank-1 tensors [10]:

$$\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] := \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r, \tag{1}$$

where '$\otimes$' denotes the outer product, and $\mathbf{a}_r, \mathbf{b}_r$, and $\mathbf{c}_r$, $r = 1, \ldots, R$, are vectors in $\mathbb{R}^I$, $\mathbb{R}^J$, and $\mathbb{R}^K$, respectively. These vectors are collected column-wise in so-called *factor matrices* $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. The outer product of N nonzero vectors forms an Nth order rank-1 tensor. The *order* of a tensor thus equals the number of dimensions. The *(CP)-rank* of the tensor is the minimum R for (1) to hold. While the formulation for a third-order tensor is given for notational simplicity, both the decomposition and the proposed algorithm naturally extend to higher-order tensors.

Compared to other tensor decomposition methods, such as the tensor-train decomposition (TTD) [11], and the Tucker decomposition or the multilinear singular value decomposition (MLSVD) [12, 13], the CPD is unique under mild assumptions [14–16], which makes the CPD a reliable and interpretable model for many applications. Furthermore, it is compact, requiring the fewest parameters for a certain rank. The standard least-squares problem for finding a CPD or low-rank approximation is:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \underbrace{\frac{1}{2} \left\| \mathcal{T} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \right\|^2}_{=: f(\mathbf{x})}, \tag{2}$$

where $x := \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ and $\|\cdot\|$ denotes the Frobenius norm. When the cost f is zero, an exact decomposition of $\mathcal{T}$ is obtained.

In a survey paper from 2008, the alternating least-squares (ALS) method was considered the most commonly used approach for solving (2) [9] and is still very popular until today. This method simplifies the problem into a sequence of linear subproblems by iteratively updating one factor matrix at a time while keeping the others fixed. However, it is now well established that *all-at-once algorithms*—those that optimize (2) over all factor matrices simultaneously—significantly outperform ALS on more difficult-to-decompose tensors [17–20]. Furthermore, a standard implementation of ALS has no convergence guarantees and the asymptotic convergence is at most linear. On the other hand, all-at-once algorithms can achieve local quadratic convergence if a second-order method is used and global convergence can be achieved in combination with a line-search or trust-region strategy. A particularly effective class of these methods are Gauss–Newton-based algorithms, including damped variants such as the

Levenberg–Marquardt (LM) method [19, 20]. Efficient implementations of these algorithms for the CPD exist, which reduce the per-iteration computational cost from $\mathcal{O}\left(R^6\right)$ to $\mathcal{O}\left(R^4\right)$, for $I \approx J \approx K \approx R$ [20]. To reduce the computational complexity further, compression using the MLSVD [13, 21], TTD or hierarchical Tucker can be applied first [22]. Several CPD optimization algorithms are implemented in toolboxes such as Tensorlab [23], TensorLy [24], and Tensor Toolbox [25].

Some tensors are more difficult to decompose or approximate than others, leading to regions of very slow convergence, informally known as *swamps*. Swamps have been studied already since 1994 [26–29], primarily in the context of ALS. While ALS is attractive due to its simplicity, it seems to be particularly prone to swamps. Swamps can be classified into two types: transient and terminal [28]. Terminal swamps arise due to an ill-conditioned Hessian at the solution, leading to slow convergence toward the optimum. An ill-conditioned Hessian can occur, e.g., due to collinear factor vectors or rank-1 terms of different orders of magnitude. Because the solution is the cause of terminal swamps, the problem is intrinsically difficult. In this work, we focus instead on the reduction of transient swamps. *Transient swamps* are caused by essential singularities in the optimization landscape [30]. For rank-2 tensors, swamps can be analyzed using dynamical system theory because of the small number of parameters [28, 30]. However, for tensors of higher ranks, the behavior can be much more complex. An example swamp is shown in Figure 1 where a $5 \times 5 \times 5$ tensor of rank 7 with unbalanced norms of the rank-1 terms is decomposed using the LM method `nls_lm` [23]. As can be seen, the method needs more than 900 iterations to converge. On the right plot, the evolution of the norms of the seven rank-1 terms is shown together with the norms of the rank-1 terms at the solution in the last iteration. The algorithm recovers the unique rank-1 terms at the solution. The swamp can, in this case, be divided into two parts visible as the two plateaus with a slightly different cost function value: the first part consists of three terms that increase in norm. At the end of the first part, one additional rank-1 term stagnates because it has become close to a rank-1 term at the solution and the swamp continues with two terms of which the norms decrease towards those of the remaining terms in the solution. The other four terms are already close to a term in the solution after less than 100 iterations and their norm remains constant during the swamp. It can be verified that also no rotations (unit-modulus complex scaling) are performed. As the cost of standard second-order optimization algorithms scales at least quartic with the rank, a significant amount of computation resources are wasted. It thus seems natural to "freeze" the terms that are not modified by (temporarily) eliminating them from the optimization problem.

Another problem is that of *degeneracy*, where a tensor can be approximated arbitrary well with a tensor of lower rank [31, 32]. This is closely related to the concept of border-rank [1]. A well-known example is where two rank-1 terms grow to infinity and approximately become each other's opposite, such that the large terms cancel out. This is also one of the essential singularities that can be studied using dynamical systems [28]. This effect is not limited to situations with two terms. We frequently observed three or more terms that grow to infinity, as shown in Figure 2. The norms of the solutions do not match the norms of the approximation. Since the true rank-1 terms are unique in this case, it can be concluded that the optimum is not found.

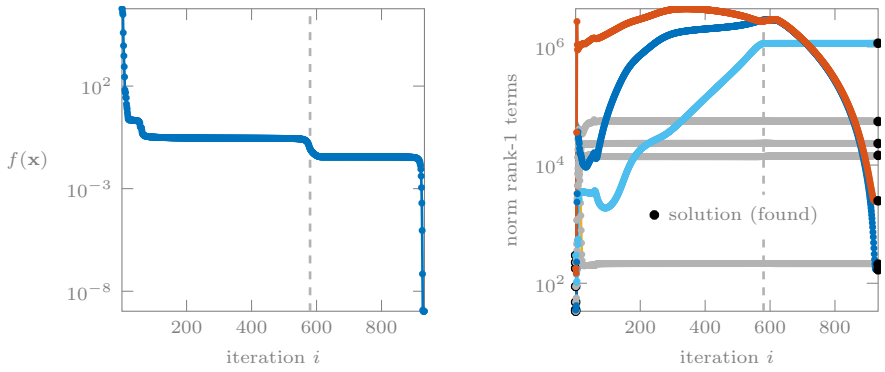

Fig. 1 *Only three of the seven rank-1 terms are changed significantly during the swamp.* Example of a swamp for a $5 \times 5 \times 5$ tensor of rank 7 with unbalanced norms of the rank-1 terms. The swamp has two plateaus of a different cost function value corresponding to different stagnant and non-stagnant terms. The solution is indicated with black dots in the last iteration and is recovered.

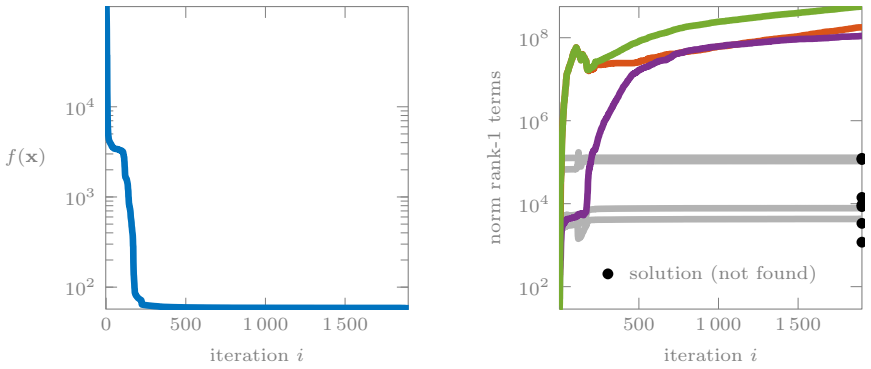

Fig. 2 *Example of a swamp with three degenerate terms for a $5 \times 5 \times 5$ tensor of rank 7 with unbalanced norms of the rank-1 terms.* While the maximal element in each of these three terms is approximately 10^3, the maximum element in their sum is only approximately 44. The norms of the rank-1 terms at the solution are shown in the right margin, but not all terms are recovered. The cost function stays above 10^1 while only little noise (with a signal-to-noise ratio (SNR) of 140 dB) was added to this tensor and the optimal cost function value should be around 10^{-8}.

Section 4 shows that the proposed algorithm can find a global optimum for this example, as the degenerate decomposition is situated in a zone where the algorithm passes, i.e., it is not part of the actual solution.

Several techniques to tackle swamps have been proposed. A first possibility is to use Riemannian optimization. More specifically, a Riemannian GN method with hot restarts (RGN-HR) method was introduced in [17]. This approach is designed for small, dense tensors. While it was not specifically designed to tackle swamps, numerical experiments show that it is particularly effective on model problems where we observed that swamps typically occur for other second-order methods as well. A possible limitation of the Riemannian optimization approach is that it cannot converge to solutions with non-unique rank-1 terms. This is because the solution must lie on

the manifold and a solution with non-unique rank-1 terms is a singular point of this manifold. The approximation of such tensors is ill-posed and their condition number is unbounded [17]. When the method converges towards an ill-conditioned decomposition, i.e., comes close to a singular point, a hot-restart strategy is employed, which involves selecting a new initialization. For non-convex problems like (2), the choice of the initialization can heavily influence the further convergence. RGN-HR outperforms other state-of-the-art methods in the 'expected time' to compute a solution when the rank-1 terms differ significantly in norm. A drawback is that the restart strategy requires the computation of the smallest eigenvalue of the GN Gramian, which is computationally expensive for large problems. As the proposed algorithm also performs well on these types of problems, both approaches are compared in Section 4.

Another approach to escape and prevent swamps is adding a regularization term to the least-squares cost function [28, 29]. The impact of regularization can be analyzed in combination with the ALS method when decomposing a rank-2 tensor using dynamical systems theory [28]. Regularization alters the optimization landscape, potentially eliminating the essential singularities that cause the transient swamps. However, this approach has drawbacks: it modifies the location of the optimum and introduces an additional regularization parameter that must be carefully tuned [29]. Moreover, when applied to second-order optimization methods, regularization eliminates quadratic convergence for exact decompositions, as the optimum is no longer at zero.

The computation of a CPD consists of three parts. The first part is determining a good initialization. In the second part, the algorithm works its way to a neighborhood of the eventual optimum. The third part is the asymptotic convergence to that optimum. As mentioned previously in the introduction, second-order all-at-once algorithms have a superior asymptotic convergence rate compared to, e.g., ALS. Concerning the first part, the initialization can be done randomly. However, one may also use algebraic methods, such as the generalized eigenvalue decomposition (GEVD) [33–35] and, more recently, the generalized eigenspace decomposition (GESD) [36]. The GEVD makes use of one pencil of two slices of the tensor to compute a CPD. A truncated MLSVD can be computed to compress the tensor to two slices. If the multilinear rank in one of the modes is exactly two, an exact CPD of the original tensor is obtained. In other cases, only an approximate CPD is obtained. GESD algorithms extend this idea by combining multiple pencils to improve the accuracy. A downside of current algebraic initializations is that they cannot be used when the tensor rank of the CPD exceeds the tensor dimensions. Furthermore, the multilinear rank must be at least $R \times R \times K$ with $K \geq 2$ [36].

In this paper, we focus on accelerating the second part of obtaining a CPD: getting close to the optimum given some initialization. Especially for "difficult" problems, the second part can be relatively expensive as illustrated in Figure 1 and Figure 2. More specifically, the paper introduces a novel optimization framework that leverages state-of-the-art numerical optimization methods while exploiting their natural behavior in swamps. More specifically, as we observed that in many swamps only a few problematic ill-conditioned rank-1 terms are present, we propose to temporarily freeze all but the problematic terms in the optimization problem. As a consequence, the problem has simplified and thanks to the reduced rank, compression can be performed, accelerating

convergence further. Additionally, a new CPD approximation of the compressed tensor can be computed using an algebraic initialization, even when this was not possible for the original tensor due to the constraints on the multilinear rank.

The remainder of the paper is organized as follows. Section 2 introduces the preliminaries for the proposed algorithm. Section 3 provides the intuition behind the proposed optimization framework, detailing its key components before presenting the full algorithm, in Section 3.3. Section 4 presents numerical experiments demonstrating that, for challenging tensor decomposition problems, the proposed algorithm outperforms state-of-the-art methods in terms of iterations, accuracy, and computation time.

2 Preliminaries

After introducing the notation, the GN method is briefly discussed. The structure of the GN Gramian is at the basis of the intuition discussed in Section 3.

2.1 Notation

Scalars, vectors, matrices, and tensors are denoted by lowercase, bold, bold uppercase, and calligraphic letters, respectively, e.g., a, $\mathbf{a}$, $\mathbf{A}$, and $\mathcal{A}$. The ith column of $\mathbf{A}$ is denoted by $\mathbf{a}_i$ and the jth element of $\mathbf{a}$ by a_j. Sets are notated similarly as tensors but the letter $\mathcal{T}$ is reserved for tensors and $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{N}$ for sets. A (C)PD is parameterized as an ordered collection of factor matrices: $\mathbf{x} := \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ and $\mathbf{x}^{(1)} := \mathbf{A}$, $\mathbf{x}^{(2)} := \mathbf{B}$, and $\mathbf{x}^{(3)} := \mathbf{C}$. '$\otimes$' denotes the tensor or outer product, '$\otimes$' the Kronecker product, '$\odot$' the Khatri–Rao or column-wise Kronecker product, $*$ the Hadamard or element-wise product, and $\mathrm{vec}(\cdot)$ the vectorization operator. Vectorization of a tensor is performed such that $\mathrm{vec}(\mathcal{T})(i+(j-1)J+(k-1)JK) = \mathcal{T}(i,j,k)$ and the vectorization of a set of factor matrices is defined as: $\mathrm{vec}(\mathbf{x}) := \{\mathrm{vec}(\mathbf{A}), \mathrm{vec}(\mathbf{B}), \mathrm{vec}(\mathbf{C})\}$. $\mathcal{T}_{[1;2,3]}$ denotes the mode-1 unfolding or matricization of a tensor, i.e., $\mathcal{T}_{[1;2,3]}(i, j+(k-1)J) = \mathcal{T}(i,j,k)$, and $\mathcal{T}_{[2;1,3]}$ and $\mathcal{T}_{[3;1,2]}$ denote the mode-2 and mode-3 unfolding, respectively. $\mathbf{A}_{\mathcal{J}} := \mathbf{A}(:, \mathcal{J})$ denotes the matrix formed by a subset of the columns of $\mathbf{A}$ defined by the set $\mathcal{J} \subseteq \{1, \cdots, R\}$. $\mathbf{x}_{\mathcal{J}} := \big\{\mathbf{A}(:,\mathcal{J}), \mathbf{B}(:,\mathcal{J}), \mathbf{C}(:,\mathcal{J})\big\}$ denotes the CPD of a subset of rank-1 terms, $\mathbf{x}+\mathbf{z} := \big\{[\mathbf{A},\mathbf{U}], [\mathbf{B},\mathbf{V}], [\mathbf{C},\mathbf{W}]\big\}$ denotes the factor matrices of the sum of two (C)PDs, and $\mathbf{v} \cdot \mathbf{z} := \{\mathbf{v}^{(1)}\mathbf{z}^{(1)}, \mathbf{v}^{(2)}\mathbf{z}^{(2)}, \mathbf{v}^{(3)}\mathbf{z}^{(3)}\}$ the multiplication of two sets of matrices, where $\mathbf{v} := \{\mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \mathbf{V}^{(3)}\}$ and $\mathbf{z} := \{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$. Note that the sum of two CPDs is a PD but not necessarily a CPD. The mode-n rank of a tensor is the matrix rank of the mode-n unfolding of the tensor. $\mathbf{I}$ denotes the identity matrix of the appropriate size. While the notation and algorithm are presented for real-valued tensors of order three, the representation and algorithm can naturally be extended to complex numbers and higher-order tensors.

2.2 Gauss–Newton method

The Gauss–Newton (GN) method is a second-order optimization method which makes use of the so-called Gramian of the Jacobian or GN approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ of the Hessian

[37], where $\mathbf{J}$ is the Jacobian matrix, which equals for (2):

$$\mathbf{J}(\mathbf{x}) := \begin{bmatrix} (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I} & \mathbf{c}_1 \otimes \mathbf{I} \otimes \mathbf{a}_1 & \cdots & \mathbf{c}_R \otimes \mathbf{I} \otimes \mathbf{a}_R & \mathbf{I} \otimes (\mathbf{B} \odot \mathbf{A}) \end{bmatrix}. \tag{3}$$

The Jacobian matrix has dimensions $IJK \times (I+J+K)R$ and rank $\min(IJK, (I+J+K)R) - 2R$ for tensors with unique rank-1 terms. The null space of the Gramian has thus always at least dimension $2R$ because of the scaling invariance of the CPD. It can be shown that the vectors in the null space result in a scaling of the rank-1 terms of the current CPD. Using (3), the Gramian $\mathbf{J}^\top\mathbf{J}$ becomes:

$$\left[\begin{array}{ccc|ccc|ccc} \beta_{11}\gamma_{11}\mathbf{I} & \cdots & \beta_{1R}\gamma_{1R}\mathbf{I} & \gamma_{11}\mathbf{a}_1\mathbf{b}_1^\top & \cdots & \gamma_{1R}\mathbf{a}_R\mathbf{b}_1^\top & \beta_{11}\mathbf{a}_1\mathbf{c}_1^\top & \cdots & \beta_{1R}\mathbf{a}_R\mathbf{c}_1^\top \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \beta_{R1}\gamma_{R1}\mathbf{I} & \cdots & \beta_{RR}\gamma_{RR}\mathbf{I} & \gamma_{1R}\mathbf{a}_1\mathbf{b}_R^\top & \cdots & \gamma_{RR}\mathbf{a}_R\mathbf{b}_R^\top & \beta_{1R}\mathbf{a}_1\mathbf{c}_R^\top & \cdots & \beta_{RR}\mathbf{a}_R\mathbf{c}_R^\top \\ \hline \gamma_{11}\mathbf{b}_1\mathbf{a}_1^\top & \cdots & \gamma_{1R}\mathbf{b}_R\mathbf{a}_1^\top & \alpha_{11}\gamma_{11}\mathbf{I} & \cdots & \alpha_{1R}\gamma_{1R}\mathbf{I} & \alpha_{11}\mathbf{b}_1\mathbf{c}_1^\top & \cdots & \alpha_{1R}\mathbf{b}_R\mathbf{c}_1^\top \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{1R}\mathbf{b}_1\mathbf{a}_R^\top & \cdots & \gamma_{RR}\mathbf{b}_R\mathbf{a}_R^\top & \alpha_{R1}\gamma_{R1}\mathbf{I} & \cdots & \alpha_{RR}\gamma_{RR}\mathbf{I} & \alpha_{1R}\mathbf{b}_1\mathbf{c}_R^\top & \cdots & \alpha_{RR}\mathbf{b}_R\mathbf{c}_R^\top \\ \hline \beta_{11}\mathbf{c}_1\mathbf{a}_1^\top & \cdots & \beta_{1R}\mathbf{c}_R\mathbf{a}_1^\top & \alpha_{11}\mathbf{c}_1\mathbf{b}_1^\top & \cdots & \alpha_{1R}\mathbf{c}_R\mathbf{b}_1^\top & \alpha_{11}\beta_{11}\mathbf{I} & \cdots & \alpha_{1R}\beta_{1R}\mathbf{I} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \beta_{1R}\mathbf{c}_1\mathbf{a}_R^\top & \cdots & \beta_{RR}\mathbf{c}_R\mathbf{a}_R^\top & \alpha_{1R}\mathbf{c}_1\mathbf{b}_R^\top & \cdots & \alpha_{RR}\mathbf{c}_R\mathbf{b}_R^\top & \alpha_{R1}\beta_{R1}\mathbf{I} & \cdots & \alpha_{RR}\beta_{RR}\mathbf{I} \end{array}\right],$$

where $\alpha_{ij} := \langle \mathbf{a}_i, \mathbf{a}_j \rangle$, $\beta_{ij} := \langle \mathbf{b}_i, \mathbf{b}_j \rangle$, and $\gamma_{ij} := \langle \mathbf{c}_i, \mathbf{c}_j \rangle$. The gradient of (2) can be written as:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \operatorname{vec}(\mathbf{G}_\mathbf{A}) \\ \operatorname{vec}(\mathbf{G}_\mathbf{B}) \\ \operatorname{vec}(\mathbf{G}_\mathbf{C}) \end{bmatrix},$$

where

$$\begin{aligned} \mathbf{G}_\mathbf{A} &= (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})_{[1;2,3]} (\mathbf{C} \odot \mathbf{B}), \\ \mathbf{G}_\mathbf{B} &= (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})_{[2;1,3]} (\mathbf{C} \odot \mathbf{A}), \\ \mathbf{G}_\mathbf{C} &= (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})_{[3;1,2]} (\mathbf{B} \odot \mathbf{A}). \end{aligned} \tag{4}$$

The linear system that is solved for the step $\mathbf{p}$ is:

$$\mathbf{J}^\top\mathbf{J}\mathbf{p} = -\nabla f(\mathbf{x}).$$

This system is thus not of full rank because of the scaling invariance of the CPD and consequently, the step $\mathbf{p}$ is not unique. This can be resolved by projecting the step orthogonal to the null space. This is also necessary to obtain global convergence, i.e., convergence to a local optimum from every starting point, and asymptotic second-order convergence if a global optimum is attained [37]. In damped GN methods, such

as the LM method, the linear system is changed according to:

$$\left(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I}\right) \mathbf{p} = -\nabla f(\mathbf{x}),$$

where $\lambda > 0$ is a damping parameter that is updated dynamically [37]. Because of this parameter, the convergence towards a global optimum is superlinear instead of quadratic. The new iterate then becomes $\text{vec}(\mathbf{x}) + \alpha \mathbf{p}$, where α can be determined using a line-search scheme. Alternatively, a trust-region scheme can be used to determine the new iterate based on $\mathbf{p}$ [37].

2.2.1 Complexity

For a standard implementation of the GN method, the most expensive step in the optimization method is solving the square linear system for the step $\mathbf{p}$: $\mathbf{J}^\top \mathbf{J}\mathbf{p} = -\nabla f(\mathbf{x})$, which is a square system of size $(I+J+K)R \times (I+J+K)R$. If we assume that $I \approx J \approx K \approx R$, then the cost to solve this system is approximately $\mathcal{O}\left(R^6\right)$. For many variables, the (preconditioned) conjugate gradient (CG) algorithm can be used to solve the linear system [20]. The most computationally expensive step is in this case the evaluation of the cost function of approximately $\mathcal{O}\left(R^4\right)$ operations. Consequently, reducing the rank by reducing the number of rank-1 terms has a significant influence on the cost of the algorithm, which is exploited by the proposed algorithm in Section 3.3.

3 Methodology

Before presenting the algorithm and the implementation details in Section 3.3 and Section 3.3.5, respectively, some intuition behind the algorithm is given in Section 3.1. Lastly, Section 3.4 gives an explanation of why some terms can stay constant during the optimization of (2) using the GN method.

3.1 Intuition

After analyzing difficult CPDs that show swamplike behavior using the ALS method as described in several papers, e.g., [26, 28–30], and performing many additional experiments that cause swamplike behavior for GN-type methods as well, we observed that in many cases only a few rank-1 terms cause problematic behavior. The remaining terms stagnate, i.e., they are not modified by the GN step during the swamp. As a matter of fact, in many cases the stagnant terms have already (almost) converged to rank-1 terms in the solution. The following observation is encountered in many swamps.

Observation 1. *Some rank-1 terms can converge to the solution before other terms: Let* $\mathbf{x} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ *be the current approximation, let* $\mathbf{x}_\mathcal{I}$*, with* $\mathcal{I} \subseteq \{1, \ldots, R\}$*, be the* $F = |\mathcal{I}|$ *terms that have converged and assume* $\mathcal{T}$ *admits an exact CPD, i.e.,*

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^* \otimes \mathbf{b}_r^* \otimes \mathbf{c}_r^*,$$

then $\mathbf{a}_i^* \approx \mathbf{a}_i$, $\mathbf{b}_i^* \approx \mathbf{b}_i$, *and* $\mathbf{c}_i^* \approx \mathbf{c}_i$, *for* $i \in \mathcal{I}$, *after resolving scaling and permutation ambiguities.*

Section 3.4 proofs that under the assumption that the stagnant terms have converged, they are indeed not changed by the GN step.

3.2 Rank-1 freezing

Given the observation that some terms do not change, we propose a metric to detect which terms are stagnant. This metric is based on the change in the overall rank-1 terms rather than the step size, to account for the scaling indeterminacy. This distance d_j between rank-1 term j in the current point $\mathbf{x}$ and the new point $\mathbf{x}^{\text{new}}$ is

$$d_j := \left\| \mathbf{x}_j^{(3)} \otimes \mathbf{x}_j^{(2)} \otimes \mathbf{x}_j^{(1)} - \mathbf{x}_{\text{new},j}^{(3)} \otimes \mathbf{x}_{\text{new},j}^{(2)} \otimes \mathbf{x}_{\text{new},j}^{(1)} \right\|^2,$$

and the normalized change n_j is given by:

$$n_j(\mathbf{x}, \mathbf{x}_{\text{new}}) := \frac{d_j}{\sum_{r=1}^{R} d_i}. \tag{5}$$

If this metric falls below a given tolerance ε, the term is frozen:

$$\mathcal{I}_{\text{freeze}} \leftarrow \{j \in \{1, \ldots, R\} \mid n_j(\mathbf{x}, \mathbf{x}_{\text{new}}) < \varepsilon\}.$$

Note that d_j can be computed efficiently without forming the full rank-1 tensors:

$$\begin{aligned} d_j = &\left\|\mathbf{x}_j^{(1)}\right\|^2 \left\|\mathbf{x}_j^{(2)}\right\|^2 \left\|\mathbf{x}_j^{(3)}\right\|^2 + \left\|\mathbf{x}_{\text{new},j}^{(1)}\right\|^2 \left\|\mathbf{x}_{\text{new},j}^{(2)}\right\|^2 \left\|\mathbf{x}_{\text{new},j}^{(3)}\right\|^2 \\ &- 2\langle \mathbf{x}_j^{(1)}, \mathbf{x}_{\text{new},j}^{(1)} \rangle \langle \mathbf{x}_j^{(2)}, \mathbf{x}_{\text{new},j}^{(2)} \rangle \langle \mathbf{x}_j^{(3)}, \mathbf{x}_{\text{new},j}^{(3)} \rangle. \end{aligned}$$

Thus, all n_j, $j = 1, \ldots, R$, can be computed in $\mathcal{O}\left(R(I + J + K)\right)$ operations or $\mathcal{O}\left(R^2\right)$ when $I \approx J \approx K \approx R$. This is consequently not a computational expensive step in a standard optimization framework to solve (2). Another, more stable approach to compute n_j efficiently in the same complexity is forming the trivial rank-2 PD of $\mathbf{x} - \mathbf{x}^{\text{new}}$: $\Delta\mathbf{x} := \left\{ \begin{bmatrix} \mathbf{x}^{(1)} & -\mathbf{x}_{\text{new}}^{(1)} \end{bmatrix}, \begin{bmatrix} \mathbf{x}^{(2)} & \mathbf{x}_{\text{new}}^{(2)} \end{bmatrix}, \begin{bmatrix} \mathbf{x}^{(3)} & \mathbf{x}_{\text{new}}^{(3)} \end{bmatrix} \right\}$, computing the $\mathbf{R}$ in the thin QR decompositions of the factor matrices: $\mathbf{Q}^{(i)}\mathbf{R}^{(i)} = \Delta\mathbf{x}^{(i)}$, and taking the norm of the CPD formed by the $\mathbf{R}$ matrices. Because the Frobenius norm is invariant under orthogonal transformation, we have $d_j = \left|\left| [\![\mathbf{R}^{(1)}, \mathbf{R}^{(2)}, \mathbf{R}^{(3)}]\!] \right|\right|^2$.

3.3 Algorithm framework

The proposed freezing framework presented in Algorithm freezing starts by running a regular second-order optimization method such as GN, until some terms, indexed by $\mathcal{I}_{\text{freeze}}$, do not change significantly anymore, i.e., $n_j < \varepsilon$. These terms are frozen and a CPD of the residual tensor is computed starting from the active terms. As there are fewer terms, compression (Line 4) can be performed, reducing the computational

cost, and an algebraic (re)initialization (Line 7) can be used. During the refinement of this (re)initialization on Line 10, Algorithm 2 is used. When Algorithm 2 stops with an empty set of indices $\mathcal{I}_{\text{freeze}}$, another stopping criterion of the inner algorithm is met. In this case, all terms are unfrozen and regular optimization on the full tensor $\mathcal{T}$ continues. The cycle between two full tensor optimizations is called a freezing cycle. The number of these cycles is counted using the variable k and can be controlled with the parameter $k_{\max}$. The main steps, i.e., the compression, algebraic (re)initialization, and (un)freezing steps, are explained in more detail in the next subsections.

The initialization of the active and frozen parameters on Line 1 involves:

$$\mathcal{T}_{\text{active}} \leftarrow \mathcal{T}, \quad \mathbf{x}_{\text{active}} \leftarrow \mathbf{x}, \quad \mathbf{x}_{\text{frozen}} \leftarrow \{\}, \quad \mathbf{v}_{\text{active}} \leftarrow \{\mathbf{I}_I, \mathbf{I}_J, \mathbf{I}_K\}, \quad k \leftarrow 1, \tag{6}$$

where $\mathbf{x}_{\text{active}}$ holds the CPD factors of the active terms that are updated in the present iteration, while $\mathbf{x}_{\text{frozen}}$ contains the frozen terms. $\mathcal{T}_{\text{active}}$ is the original tensor minus the frozen terms. A maximum value can be defined for this parameter to control the computational effort. Lastly, $\mathbf{v}_{\text{active}}$ stores the factors to decompress $\mathbf{x}_{\text{active}}$ to the original dimensions.

The inner Algorithm 2 consists of the optimization step on Line 2, which is the placeholder for the actual optimization algorithm, and the criterion to fix terms described in Section 3.2. In practice, any numerical optimization algorithm can be used on Line 2. The algorithms currently used and other implementation details are discussed in Section 3.3.5.

3.3.1 Compression of active tensor

When the frozen terms are subtracted from the current active tensor $\mathcal{T}_{\text{active}}$ of size $I_{\text{active}} \times J_{\text{active}} \times K_{\text{active}}$, a CPD with fewer terms is computed. When the rank is sufficiently small, an MLSVD can be computed of $\mathcal{T}_{\text{active}}$ with a core tensor that contains fewer elements. The notation of the MLSVD is:

$$\mathcal{T}_{\text{active}} = \mathcal{S} \cdot_1 \mathbf{V}^{(1)} \cdot_2 \mathbf{V}^{(2)} \cdot_3 \mathbf{V}^{(3)},$$

where $\mathbf{V}^{(i)}$, $i = 1, 2, 3$, are orthogonal factor matrices of dimensions $I_{\text{active}} \times I_{\mathcal{S}}$, $J_{\text{active}} \times J_{\mathcal{S}}$, and $K_{\text{active}} \times K_{\mathcal{S}}$, respectively. Since the mode-n ranks, $n = 1, 2, 3$, are always smaller than or equal to the CP-rank, an MLSVD can be computed with core size $I_{\mathcal{S}} \times J_{\mathcal{S}} \times K_{\mathcal{S}}$, where $I_{\mathcal{S}} := \min\left(R_{\text{active}} + q, I_{\text{active}}\right)$, where $q \geq 0$, and similarly for $J_{\mathcal{S}}$ and $K_{\mathcal{S}}$. The parameter q can be chosen to relax the compression rank. The following compression ratio can be defined:

$$\gamma := \frac{I_{\mathcal{S}} J_{\mathcal{S}} K_{\mathcal{S}}}{I_{\text{active}} J_{\text{active}} K_{\text{active}}}, \tag{7}$$

which is the ratio of the number of elements in the core tensor $\mathcal{S}$ versus the number of elements in the active tensor $\mathcal{T}_{\text{active}}$. Compression can be performed if the compression ratio γ is below a given tolerance. If compression is performed, the active tensor is set equal to the core tensor, the orthogonal factor matrices are saved in the set $\mathbf{v}_{\text{active}}$,

Algorithm freezing CPD with rank-1 freezing

Input: $\mathcal{T}$, options for Algorithm 2, initial guess $\mathbf{x}$, tolerance ε on n_j (5), maximum number of full cycles $k_{\max}$

$\mathcal{T}_{\text{active}}, \mathbf{x}_{\text{active}}, \mathbf{x}_{\text{frozen}}, \mathbf{v}_{\text{active}}, k \leftarrow$ initialize active and frozen parameters (6)
while maximum number of cumulative iterations of Algorithm 2 not reached
  if criterion compression satisfied
    $\mathcal{T}_{\text{active}}, \mathbf{v} \leftarrow$ perform MLSVD compression of $\mathcal{T}_{\text{active}}$ (Section 3.3.1)
    $\mathbf{x}_{\text{active}}, \mathbf{v}_{\text{active}} \leftarrow$ update active point $\mathbf{x}_{\text{active}}$ and factors $\mathbf{v}$ using (8)
    if algebraic (re)initialization applicable
      $\mathbf{x}_{\text{active}} \leftarrow$ algebraic initialization of $\mathcal{T}_{\text{active}}$ (Section 3.3.2)
      $\mathcal{I}_{\text{freeze}} \leftarrow \emptyset$
      if refinement chosen
        $\mathbf{x}_{\text{active}}, \mathcal{I}_{\text{freeze}} \leftarrow$ Algorithm 2 $(\mathcal{T}_{\text{active}}, \mathbf{x}_{\text{active}}, \text{options}, \varepsilon)$
    else
      # optimization on compressed tensor
      $\mathbf{x}_{\text{active}}, \mathcal{I}_{\text{freeze}} \leftarrow$ Algorithm 2 $(\mathcal{T}_{\text{active}}, \mathbf{x}_{\text{active}}, \text{options}, \varepsilon)$
  else
    # optimization on uncompressed tensor
    $\mathbf{x}_{\text{active}}, \mathcal{I}_{\text{freeze}} \leftarrow$ Algorithm 2 $(\mathcal{T}_{\text{active}}, \mathbf{x}, \text{options}, \varepsilon)$
  if $\mathcal{I}_{\text{freeze}} = \emptyset$
    $\mathcal{T}_{\text{active}}, \mathbf{x}_{\text{active}}, \mathbf{x}_{\text{frozen}}, \mathbf{v}_{\text{active}} \leftarrow$ unfreeze all terms (10)
    if converged (11)
      Output: $\mathbf{x} \leftarrow \mathbf{x}_{\text{active}}$
    $k \leftarrow k + 1$
    if $k > k_{\max}$
      $\varepsilon \leftarrow -\infty$ *# stop freezing*
  else
    $\mathcal{T}_{\text{active}}, \mathbf{x}_{\text{active}}, \mathbf{x}_{\text{frozen}}, \mathbf{v}_{\text{active}} \leftarrow$ freeze terms $(\mathcal{I}_{\text{freeze}})$ (9)

Output: $\mathbf{x}$

Algorithm 2 Inner algorithm in Algorithm freezing

Input: $\mathcal{T}_{\text{active}}$, options, starting point $\mathbf{x}_{\text{active}}$, tolerance ε on n_j (5)

while not converged (11) & $\mathcal{I}_{\text{freeze}} = \emptyset$
  $\mathbf{x}_{\text{new}} \leftarrow$ optimization step $(\mathcal{T}_{\text{active}}, \mathbf{x}, \text{options}, \varepsilon)$
  $\mathcal{I}_{\text{freeze}} \leftarrow \{j \in \{1, \ldots, R_{\text{active}}\} \mid n_j(\mathbf{x}, \mathbf{x}_{\text{new}}) < \varepsilon\}$ (5)
  $\mathbf{x} \leftarrow \mathbf{x}_{\text{new}}$

Output: $\mathbf{x}$, $\mathcal{I}_{\text{freeze}}$

and the new active decomposition is changed according to:

$$\mathcal{T}_{\text{active}} \leftarrow \mathcal{S}, \qquad \mathbf{v}_{\text{active}} \leftarrow \mathbf{v}_{\text{active}} \cdot \mathbf{v}, \qquad \mathbf{x}_{\text{active}} \leftarrow \mathbf{v}^{\top} \cdot \mathbf{x}_{\text{active}}, \tag{8}$$

where the transpose is taken cell-wise. Note that the multiplications are between two sets of matrices, as defined in Section 2.1. Decompression is only performed at the

end of each freezing cycle as discussed in the unfreezing part in Section 3.3.3, to save computational cost.

3.3.2 Algebraic (re)initialization

Despite the compression of the active tensor and the reduced number of terms, it is possible that the algorithm still needs a significant amount of iterations to converge to a solution for $\mathcal{T}_{\text{active}}$. Thus, to save iterations and consequently speed up the convergence on the compressed tensors, an algebraic (re)initialization can be computed, e.g, using a GEVD or GESD algorithm [33, 36]. Such algorithms provide in general a good (re)initialization of numerical CPD algorithms if the required conditions on the mode-n ranks are met. Thanks to freezing, the active rank is reduced and these algorithms become applicable even in cases when they were not on the full tensor.

The refinement step on Line 10 can be activated to improve this (re)initialization on the active tensor. The only difference with the optimization step on Line 13 is this (re)initialization. During the refinement, more terms can be fixed. Otherwise, the algorithm automatically continues to optimize all terms at once on Line 16. If needed, more terms can be frozen again in the next freezing cycle.

3.3.3 Freezing and unfreezing: details

As discussed previously, if Algorithm 2 exits and $\mathcal{I}_{\text{freeze}}$ is not empty, the terms defined by this set are temporarily excluded from the optimization problem in the following iteration(s). The resulting CPD factors are denoted by called $\mathbf{x}_{\text{active}}$ and the new tensor is called $\mathcal{T}_{\text{active}}$:

$$\begin{aligned} &\mathcal{T}_{\text{active}} \leftarrow \mathcal{T}_{\text{active}} - [\![\mathbf{x}_{\mathcal{I}_{\text{freeze}}}]\!], && \mathcal{J}_{\text{active}} \leftarrow \{1, \cdots, R_{\text{active}}\} \setminus \mathcal{I}_{\text{freeze}}, \\ &R_{\text{active}} \leftarrow |\mathcal{J}_{\text{active}}|, && \mathbf{x}_{\text{active}} \leftarrow \mathbf{x}_{\mathcal{J}_{\text{active}}}, \\ &\mathbf{x}_{\text{frozen}} \leftarrow \mathbf{x}_{\text{frozen}} + \mathbf{v}_{\text{active}} \cdot \mathbf{x}_{\mathcal{I}_{\text{freeze}}}, \end{aligned} \tag{9}$$

where $\mathbf{v}_{\text{active}}$ contains the factors of the MLSVD when compression is applied, as discussed in Section 3.3.1, and $\mathbf{x}_{\text{frozen}}$ contains all currently frozen terms. This process is continued until no more terms can be frozen and until the condition for optimality discussed in Section 3.3.5 is satisfied. When this happens, all frozen and current active terms are merged and the method continues on the original tensor $\mathcal{T}$:

$$\begin{aligned} &\mathcal{T}_{\text{active}} \leftarrow \mathcal{T}, && \mathbf{x}_{\text{active}} \leftarrow \mathbf{x}_{\text{frozen}} + \mathbf{v}_{\text{active}} \cdot \mathbf{x}_{\text{active}}, && R_{\text{active}} \leftarrow R, \\ &\mathbf{x}_{\text{frozen}} \leftarrow \{\}, && \mathbf{v}_{\text{active}} \leftarrow \{\mathbf{I}_I, \mathbf{I}_J, \mathbf{I}_K\}. \end{aligned} \tag{10}$$

This is one freezing cycle. The number of freezing cycles can be controlled with the parameter $k_{\max}$.

3.3.4 Complexity and convergence

The most expensive step in Algorithm freezing is the optimization step on Line 2 of Algorithm 2 of at least $\mathcal{O}\left(I_{\text{active}} J_{\text{active}} K_{\text{active}} R_{\text{active}}\right)$ operations per iteration, or

$\mathcal{O}\left(R_{\text{active}}^4\right)$ when $I_{\text{active}} \approx J_{\text{active}} \approx K_{\text{active}} \approx R_{\text{active}}$. The computation of the MLSVD also requires $\mathcal{O}\left(R_{\text{active}}^4\right)$ operations [13, 38], the computation of the algebraic initialization $\mathcal{O}\left(R_{\text{active}}^3\right)$ operations, and as explained in Section 3.2, the measure n_j requires only $\mathcal{O}\left(R_{\text{active}}^2\right)$ operations.

The same convergence properties as the chosen optimization algorithm are obtained. This is a consequence of imposing a maximum on the number of full freezing cycles $k_{\max}$ and because the algorithm always stops with the inner optimization Algorithm 2 on the full tensor (using the chosen optimization algorithm).

3.3.5 Implementation

To illustrate the performance of Algorithm freezing, a Matlab implementation is made using Tensorlab [23]. As any algorithm can be used, we have tested both GN (`nls_gndl`) and LM (`nls_lm`) and both the exact (`largescale=false`) and inexact version (`largescale=true`). The inexact version solves the linear system for the step using a preconditioned CG method. The lower the number of CG iterations, the lower the cost, but the less accurate the system will be solved. Instead of the number of iterations, also the tolerance on the residual can be tuned (default is 10^{-6}).

Compression is performed with $q = 0$ equal to zero and if the compression ratio γ defined in (7) is strictly smaller than 1, thus if the number of elements of the core tensor will be strictly smaller than the number of elements of the active tensor. The `cpd_gevd` and `cpd_gesd` [36] functions are used to compute an algebraic (re)initialization if applicable, as discussed in Section 3.3.2.

Algorithm 2 does successive steps until one of the following stopping criteria is met:

$$\frac{f(\mathbf{x}) - f(\mathbf{x}_{\text{new}})}{f(\mathbf{x})} < \text{options.TolRelF} \quad \text{or} \quad \frac{\|\mathbf{p}\|}{\|\mathbf{x}\|} < \text{options.TolX}, \tag{11}$$

or until a maximum number of iterations $i_{\max}$ is reached or the condition to fix terms, see Section 3.2, is met. During optimization on compressed tensors, $i_{\max} = 100$, and otherwise $i_{\max} = 10^4$ for the experiments in Section 4. The stopping criteria in (11) are also used on Line 1 in Algorithm 2.

Another parameter that is included is a minimum number of iterations before terms can be frozen. This is because in the first few iterations, the terms change significantly. As this change mostly consists of scaling the rank-1 terms, the metric n_j fluctuates strongly. It is thus advised to wait a few iterations before deciding on freezing terms. We experimentally found 10 to be a good value.

3.4 Theoretical insights

We prove that under the assumption that the stagnant terms have converged as in Observation 1, they are indeed not changed by the GN step. The following proposition shows that the gradient is zero for terms that have converged, under certain assumptions. These assumptions may appear strong, but they can be slightly relaxed and can therefore be realistic in practice, as explained at the end of this subsection.

Proposition 1. *Let $\mathcal{J} := \{1, \ldots, R\} \backslash \mathcal{I}$ and let the terms in $\mathbf{x}_{\mathcal{I}}$ be converged before the terms in $\mathbf{x}_{\mathcal{J}}$, as in Observation 1. Assume that the factor vectors in $\mathbf{x}_{\mathcal{I}}$ are orthogonal*

to the ones in $\mathbf{x}_{\mathcal{J}}$*:*

$$\mathbf{A}_{\mathcal{I}}^{\top}\mathbf{A}_{\mathcal{J}} = \mathbf{0}, \qquad \mathbf{B}_{\mathcal{I}}^{\top}\mathbf{B}_{\mathcal{J}} = \mathbf{0}, \qquad \mathbf{C}_{\mathcal{I}}^{\top}\mathbf{C}_{\mathcal{J}} = \mathbf{0}, \tag{12}$$

and that the factor vectors of the swamping terms lie in the subspaces defined by $\mathbf{A}_{\mathcal{J}}$*,* $\mathbf{B}_{\mathcal{J}}$*, and* $\mathbf{C}_{\mathcal{J}}$*:*

$$\mathbf{a}_j^* = \mathbf{A}_{\mathcal{J}}\mathbf{d}_j, \qquad \mathbf{b}_j^* = \mathbf{B}_{\mathcal{J}}\mathbf{e}_j, \qquad \mathbf{c}_j^* = \mathbf{C}_{\mathcal{J}}\mathbf{f}_j, \tag{13}$$

for all $j \in \mathcal{J}$ *and some nonzero* $\mathbf{d}_j, \mathbf{e}_j, \mathbf{f}_j \in \mathbb{R}^{R-F}$*. Then* $\mathbf{G}_{\mathbf{A}_{\mathcal{I}}}$*,* $\mathbf{G}_{\mathbf{B}_{\mathcal{I}}}$*, and* $\mathbf{G}_{\mathbf{C}_{\mathcal{I}}}$ *in* (4) *are zero.*

Proof. The R columns in $\mathbf{G}_{\mathbf{A}}$ in (4) correspond to the R derivatives of f to $\mathbf{a}_r$, $r = 1, \ldots, R$. Thus, using Observation 1, the columns defined by $\mathcal{I}$ can be rewritten as

$$\begin{aligned}
\mathbf{G}_{\mathbf{A}_{\mathcal{I}}} &= \left(\sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r - \sum_{r=1}^{R} \mathbf{a}_r^* \otimes \mathbf{b}_r^* \otimes \mathbf{c}_r^* \right)_{[1;2,3]} (\mathbf{C}_{\mathcal{I}} \odot \mathbf{B}_{\mathcal{I}}) \\
&= \left(\sum_{j \in \mathcal{J}} \mathbf{a}_j \left(\mathbf{c}_j \otimes \mathbf{b}_j\right)^{\top} - \mathbf{a}_j^* \left(\mathbf{c}_j^* \otimes \mathbf{b}_j^*\right)^{\top} \right) (\mathbf{C}_{\mathcal{I}} \odot \mathbf{B}_{\mathcal{I}}) \\
&= \left(\sum_{j \in \mathcal{J}} \mathbf{a}_j \left(\mathbf{C}_{\mathcal{I}}^{\top}\mathbf{c}_j \otimes \mathbf{B}_{\mathcal{I}}^{\top}\mathbf{b}_j\right)^{\top} - \mathbf{a}_j^* \left(\mathbf{C}_{\mathcal{I}}^{\top}\mathbf{c}_j^* \otimes \mathbf{B}_{\mathcal{I}}^{\top}\mathbf{b}_j^*\right)^{\top} \right) \\
&\overset{(13)}{=} \left(\sum_{j \in \mathcal{J}} \mathbf{a}_j \left(\mathbf{C}_{\mathcal{I}}^{\top}\mathbf{c}_j \otimes \mathbf{B}_{\mathcal{I}}^{\top}\mathbf{b}_j\right)^{\top} - \mathbf{a}_j^* \left(\mathbf{C}_{\mathcal{I}}^{\top}\mathbf{C}_{\mathcal{J}}\mathbf{f}_j \otimes \mathbf{B}_{\mathcal{I}}^{\top}\mathbf{B}_{\mathcal{J}}\mathbf{e}_j\right)^{\top} \right) \overset{(12)}{=} \mathbf{0},
\end{aligned}$$

and similarly for $\mathbf{G}_{\mathbf{B}_{\mathcal{I}}}$ and $\mathbf{G}_{\mathbf{C}_{\mathcal{I}}}$. □

Theorem 2. *If the terms in* $\mathbf{x}_{\mathcal{I}}$ *converge before the terms in* $\mathbf{x}_{\mathcal{J}}$ *as in Observation 1 and if the assumptions in Proposition 1 are true, the step of the GN iteration corresponding to the converged terms is zero:* $\begin{bmatrix}\mathbf{p}_{\mathbf{A}_{\mathcal{I}}}^{\top} & \mathbf{p}_{\mathbf{B}_{\mathcal{I}}}^{\top} & \mathbf{p}_{\mathbf{C}_{\mathcal{I}}}^{\top}\end{bmatrix}^{\top} = \mathbf{0}$*, or at most results in a scaling of the corresponding factor matrix columns.*

Proof. If the step $\mathbf{p}$ and Jacobian $\mathbf{J}$ are divided in three parts: $\mathbf{p}^{\top} := \begin{bmatrix}\mathbf{p}_{\mathbf{A}}^{\top} & \mathbf{p}_{\mathbf{B}}^{\top} & \mathbf{p}_{\mathbf{C}}^{\top}\end{bmatrix}$ and $\mathbf{J} := \begin{bmatrix}\mathbf{J}_{\mathbf{A}} & \mathbf{J}_{\mathbf{B}} & \mathbf{J}_{\mathbf{C}}\end{bmatrix}$, the linear system for the step can be rewritten as:

$$\begin{bmatrix}
\mathbf{J}_{\mathbf{A}}^{\top}\mathbf{J}_{\mathbf{A}} & \mathbf{J}_{\mathbf{A}}^{\top}\mathbf{J}_{\mathbf{B}} & \mathbf{J}_{\mathbf{A}}^{\top}\mathbf{J}_{\mathbf{C}} \\
\mathbf{J}_{\mathbf{B}}^{\top}\mathbf{J}_{\mathbf{A}} & \mathbf{J}_{\mathbf{B}}^{\top}\mathbf{J}_{\mathbf{B}} & \mathbf{J}_{\mathbf{B}}^{\top}\mathbf{J}_{\mathbf{C}} \\
\mathbf{J}_{\mathbf{C}}^{\top}\mathbf{J}_{\mathbf{A}} & \mathbf{J}_{\mathbf{C}}^{\top}\mathbf{J}_{\mathbf{B}} & \mathbf{J}_{\mathbf{C}}^{\top}\mathbf{J}_{\mathbf{C}}
\end{bmatrix}
\begin{bmatrix}\mathbf{p}_{\mathbf{A}} \\ \mathbf{p}_{\mathbf{B}} \\ \mathbf{p}_{\mathbf{C}}\end{bmatrix} =
\begin{bmatrix}\operatorname{vec}(\mathbf{G}_{\mathbf{A}}) \\ \operatorname{vec}(\mathbf{G}_{\mathbf{B}}) \\ \operatorname{vec}(\mathbf{G}_{\mathbf{C}})\end{bmatrix}.$$

If (12) and (13) hold, and because any of the nine blocks contains at least one of the inner products $\langle \mathbf{a}_i, \mathbf{a}_j \rangle$, $\langle \mathbf{c}_i, \mathbf{c}_j \rangle$, and $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ on position (i, j), the elements at (i, j) in each block will be zero for all $i \in \mathcal{I}$ and $j \in \mathcal{J}$, e.g, $\mathbf{J}_{\mathbf{A}_{\mathcal{I}}}^{\top}\mathbf{J}_{\mathbf{A}_{\mathcal{J}}} = \mathbf{0}$, and similarly for the other combinations of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. From Proposition 1 follows that the components

in the gradient corresponding to the converged terms defined by $\mathcal{I}$ are zero. If the factor vectors are reordered into two groups, first the ones defined by the set $\mathcal{I}$ and secondly the ones defined by $\mathcal{J}$, the following reordered system is obtained:

$$\left[\begin{array}{cc|cc|cc} \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{\mathbf{A}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{A}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{A}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{J}} \\ \hline \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{\mathbf{B}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{B}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{B}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{J}} \\ \hline \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{0} & \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{\mathbf{C}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{C}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{J}} & \mathbf{0} & \mathbf{J}_{\mathbf{C}_\mathcal{J}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{J}} \end{array}\right] \begin{bmatrix} \mathbf{p}_{\mathbf{A}_\mathcal{I}} \\ \mathbf{p}_{\mathbf{A}_\mathcal{J}} \\ \mathbf{p}_{\mathbf{B}_\mathcal{I}} \\ \mathbf{p}_{\mathbf{B}_\mathcal{J}} \\ \mathbf{p}_{\mathbf{C}_\mathcal{I}} \\ \mathbf{p}_{\mathbf{C}_\mathcal{J}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \operatorname{vec}(\mathbf{G}_{\mathbf{A}_\mathcal{J}}) \\ \mathbf{0} \\ \operatorname{vec}(\mathbf{G}_{\mathbf{B}_\mathcal{J}}) \\ \mathbf{0} \\ \operatorname{vec}(\mathbf{G}_{\mathbf{C}_\mathcal{I}}) \end{bmatrix},$$

as illustrated in Figure 3. Thus, the following linear system is obtained for the step corresponding to the rank-1 terms defined by $\mathcal{I}$:

$$\begin{bmatrix} \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{J}_{\mathbf{A}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} \\ \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{J}_{\mathbf{B}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} \\ \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{A}_\mathcal{I}} & \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{B}_\mathcal{I}} & \mathbf{J}_{\mathbf{C}_\mathcal{I}}^\top \mathbf{J}_{\mathbf{C}_\mathcal{I}} \end{bmatrix} \begin{bmatrix} \mathbf{p}_{\mathbf{A}_\mathcal{I}} \\ \mathbf{p}_{\mathbf{B}_\mathcal{I}} \\ \mathbf{p}_{\mathbf{C}_\mathcal{I}} \end{bmatrix} = \mathbf{0}.$$

Because the full right hand side for the set $\mathcal{I}$ is zero, the solution must lie in the null space of this matrix. It can be easily shown that the null space vectors can only scale the factor vectors in $\mathbf{x}_\mathcal{I}$.

□

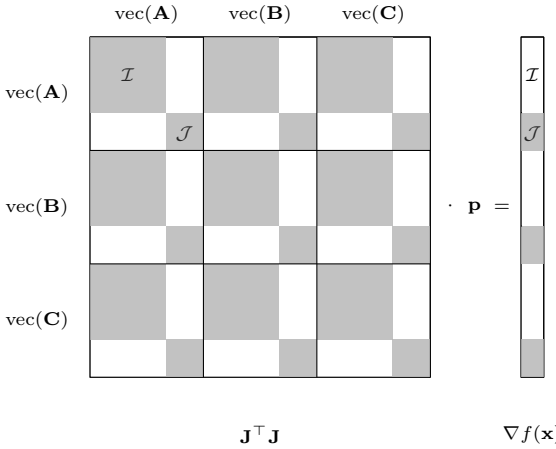

Fig. 3 The Gramian of (2) has a specific structure when the conditions in (12) hold. The gray areas indicate nonzeros and the white ones zeros. The factor vectors are permuted to separate $\mathbf{x}_\mathcal{I}$ and $\mathbf{x}_\mathcal{J}$.

Although Theorem 2 says that the step can still scale the converged factor vectors in $\mathbf{x}_\mathcal{I}$, these directions are usually filtered out in the numerical implementation by ensuring that the step is orthogonal to the null space of the Gramian. This is actually a necessity for second-order convergence. This is done, e.g., by using the pseudo-inverse

or a (modified) CG method for solving the linear system or by using a damped GN method such as the LM method, which makes the matrix of the linear system full rank.

Although the orthogonality conditions in (12) may appear strong, they can be realistic in practice. First, if not zero, the inner products in (12) are small. We often observe differences in step size entries of orders of magnitude. Secondly, many applications involve random processes that lead to factor vectors of which the entries are generated using some independent, zero mean distribution. For such vectors, the expected value of the inner product is zero. Lastly, if the inner product is not small, the terms are correlated and thus both likely part of the problematic terms (defined by $\mathcal{J}$).

4 Numerical experiments

To illustrate the application and advantages of Algorithm freezing, a synthetic model similar to the one in [17] is used to generate difficult-to-decompose tensors. First, synthetic factor matrices are generated randomly from a normal distribution (using `randn` in Matlab), and afterwards the first $m \in \{1, \ldots, R\}$ columns are changed according to:

$$\mathbf{A}(:, 1:m) \leftarrow \mathbf{A}(:, 1:m) \cdot \mathbf{R}_c \cdot \operatorname{diag}\left(10^{\frac{s}{3m}}, 10^{\frac{2s}{3m}}, \cdots, 10^{\frac{sn}{3m}}\right), \tag{14}$$

where $\mathbf{R}_c$ is the upper-triangular factor in the Cholesky factorization $\mathbf{R}_c^\top \mathbf{R}_c = c\mathbf{1}\mathbf{1}^\top + (1-c)\mathbf{I}$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. The main parameters that are varied over the experiments are the collinearity parameter c, the scaling parameter s, the number of modified columns m, and the rank R. Increasing one or more of these parameters increases the difficulty of the problem and increases the occurrence of swamps. Afterwards, the tensor $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is perturbed by Gaussian noise $\mathcal{N}$ such that a certain signal-to-noise ratio (SNR) η in dB is obtained:

$$\mathcal{T}_\eta = \mathcal{T} + \mathcal{N}, \qquad \eta = 20 \cdot \log_{10}\left(\frac{\|\mathcal{T}\|}{\|\mathcal{N}\|}\right).$$

The following criterion is used to verify successful recovery of the solution [17]:

$$\frac{\|\mathcal{T}_\eta - [\![\mathbf{x}]\!]\|}{\|\mathcal{T}_\eta\|} < 1.1 \cdot 10^{-\frac{\eta}{20}} \;\;\&\;\; \operatorname{cond}(\mathbf{x}) \leq 50 \cdot \operatorname{cond}(\mathbf{x}^*) \;\;\&\;\; \max(\mathbf{e}) < 10^{-1}, \tag{15}$$

where the condition number is defined in [17], and the relative factor matrix error is:

$$\mathbf{e}(i) = \frac{\|\mathbf{x}^{*,(i)} - \mathbf{x}^{(i)} \cdot \mathbf{P} \cdot \mathbf{D}^{(i)}\|}{\|\mathbf{x}^{*,(i)}\|},$$

which is computed using `cpderr`, for $i = 1, 2, 3$, and where $\mathbf{x}^* = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ contains the factor matrices at the solution, $\mathbf{x}$ contains the factor matrices of the approximation, and $\mathbf{P}$ and $\mathbf{D}^{(i)}$ are a permutation and diagonal scaling matrices to match the factors in the solution as well as possible. The first criterion is a function of the relative cost function and the noise level, the second criterion says that the solution cannot have

a much larger condition number than the ground truth, and the last says that each factor matrix has to be found with sufficient accuracy. Unless mentioned otherwise, the parameters from Algorithm freezing are set to:

$$\varepsilon := 10^{-4}, \quad \texttt{options.TolRelF} := 10^{-10}, \quad k_{\max} := 50, \quad \text{and} \quad \texttt{options.TolX} := 10^{-10}.$$

4.1 Revisiting example swamps

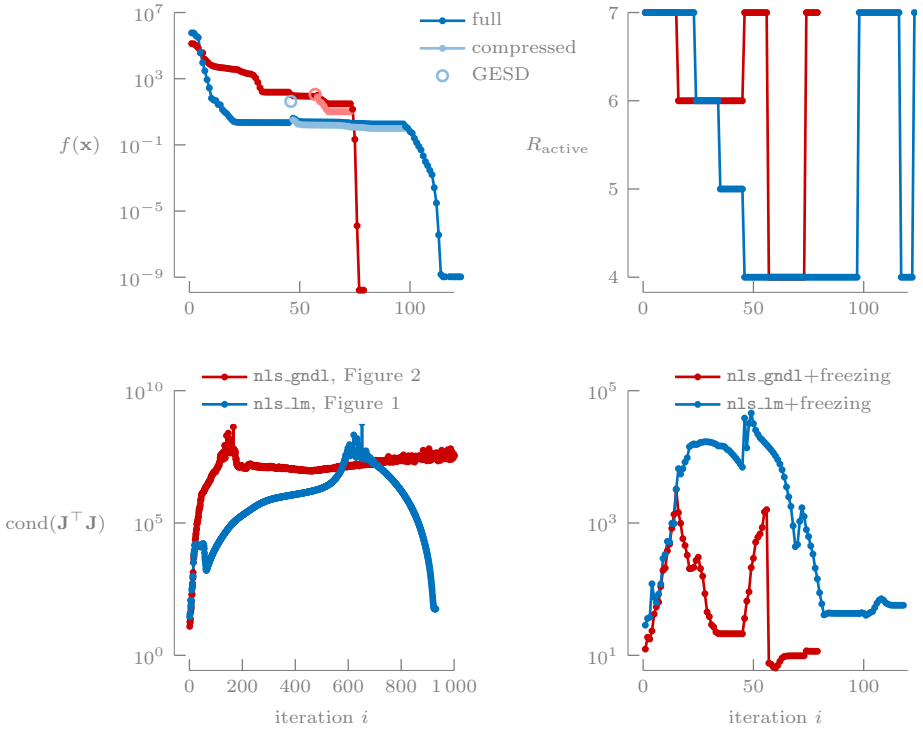

Fig. 4 *Using Algorithm freezing,* `nls_lm` *needs much fewer iterations to recover the solution for the tensor and starting point in Figure 1 (blue curves). Furthermore, the required time is reduced from approximately* 6 s *to* 1 s. `nls_gndl` *with Algorithm freezing is able to recover the solution for the tensor and starting point in Figure 2 (red curves). No degenerate tensor is formed in this case during convergence.* The upper left plot shows the evolution of the cost function value $f(\mathbf{x})$ on both the full and active tensor. During compression, the cost function value on the active tensor (light blue/red) is always lower than or equal to the cost on the full tensor (dark blue/red). The upper right plot shows the number of active terms R_{active} as a function of the iterations. Compression is performed when the active rank R_{active} has dropped to 4 in this example. The lower two plots show the evolution of the condition number [17] without and with freezing (left and right bottom plot, respectively). Note the scale difference of the y-axis. After compression, the condition number clearly decreases (bottom right plot). The condition number at the solution is lower than 100, whereas without freezing the condition number increases above 10^5 (bottom left plot).

First, the results of Algorithm freezing are given for the example swamps shown in Figure 1 and Figure 2, which were obtained with `nls_lm` and `nls_gndl` with

`largescale` = `false`, respectively. The inner algorithm on Line 2 in Algorithm 2 is in both cases the same as the algorithm being compared to. The scaling parameter $s = 2$, the collinearity $c = 0$, the SNR $\eta = 140\,\text{dB}$, and the number of modified columns $m = 7$. We recall from the introduction that the size of both tensors is $5 \times 5 \times 5$ and the rank 7. Furthermore, `cpd_gesd` is chosen as algebraic (re)initialization [36][1] and refinement is activated. The results are shown in Figure 4. It can be seen that in both experiments, the solution is recovered, whereas using `nls_gndl` on the second example, the solution was not found before. The number of iterations in the first example is also significantly reduced from approximately 10^3 to 10^2 and the computation time from approximately 6 s to 1 s (not shown on the figures). When the active rank drops to 4, the rank has become smaller than the tensor dimensions. Consequently, compression can be performed and a GESD (re)initialization can be computed. Although the GESD step (slightly) increases the overall cost function, the cost function and condition number [17] (lower two plots) decrease significantly afterwards. The light blue and light red lines in Figure 4 indicate the cost function on the active tensor during compression. This cost function is always smaller than or equal to the cost function on the corresponding decompressed tensor because the residual or active tensor is in most cases not exactly of low-rank and thus some 'information' (can also be noise) is lost after compression. Thus when decompressing, the cost function will increase again slightly. Note that the full cost function during compression is only computed in these examples for illustrative purposes.

4.2 Different difficult-to-decompose tensors

The improvements of Algorithm freezing on (14) for different parameters are shown in Figure 5 and Figure 6. We noticed that in many cases the refinement step is not necessary and is therefore turned off in these experiments. In Figure 5, the average behavior of the iterations and computation time with and without freezing is shown on 100 experiments per parameter set. Also the number of successful cases based on (15) is indicated on the figures. In the complementary Figure 6, the behavior per single experiment can be seen. The size of the tensor is set to $15 \times 15 \times 15$ and the rank to 20. In the last set, the size is increased to $100 \times 100 \times 100$ by multiplying each mode with a random orthogonal matrix of size 100×15. The maximum number of iterations is set to 10^3. The SNR value ranges from 140 dB in the first four sets to 10 dB in the last. In all but two sets the `nls_gndl` method with `largescale` = `true` is used. The different improvements are detailed in the next paragraphs.

Algorithm freezing is robust in the maximum number of CG iterations. In the first set in Figure 5, the maximum number of CG iterations is set equal to the rank (20), which is slightly higher than the default value. Without freezing, in more than 75 experiments, the maximum number of iterations is reached and only 20 experiments are successful. On the other hand, with freezing, all cases are found and the distribution of the iterations and computation time is much lower. In the next set, the CG iterations are increased to 100. Increasing the CG iterations increases the computation time per iteration but typically reduces the overall number of iterations. Significantly more solutions (83) are found without freezing, and, with freezing, still all solutions

[1] https://www.tensorlabplus.net/papers/evert2022gesd.html

are found. The number of iterations with freezing stays approximately the same. It can thus be concluded that increasing the number of CG iterations has a significant influence on the performance of `nls_gndl` without freezing but does not have a significant influence on the performance with freezing in this example.

Algorithm freezing performs well for high scaling parameters s. For sets two to four, the CG iterations are increased to 100 and the scaling and collinearity parameter are gradually increased from 3 to 4 and 0 to 0.25, respectively. Note that the scaling parameter s is in the exponent in (14) which means that the difference in norm of the rank-1 terms in the first set is approximately 10^3 and in the last 10^4. In general, increasing the parameters s, c, m, and R, makes (2) more difficult to solve. This can be checked by looking at the condition number of the Jacobian matrix of the cost function (2) [39]. Figure 5 and Figure 6 also show that the experiments get increasingly difficult for `nls_gndl` without freezing from sets two to four: the number of iterations increases significantly and fewer experiments are successful. However, with freezing, all solutions are still found without increasing the number of iterations or computation time significantly.

Exact implementations to compute the GN step are not always reliable for ill-conditioned CPDs. When the number of CG iterations is set to ∞ (`largescale` = `false`), the pseudo-inverse, i.e., `pinv`, is used to solve the linear system. This setting is used in the sixth set in Figure 5. When the solution is increasingly ill-conditioned, the result of a numerical implementation of the pseudo-inverse, e.g., `pinv` in Matlab, becomes inaccurate and some directions that are close to the null-space are filtered out in the search direction. These directions are usually crucial to get out of a swamp. Consequently, exact GN methods are not always reliable for ill-conditioned CPDs and more red lines are visible in Figure 6.

Algorithm freezing performs equally well in combination with other optimization methods such as the LM method. Sets five to seven have an SNR $\eta = 40\,\text{dB}$. If we want to recover all terms with at least one correct digit, the noise should not be higher than the norm of the smallest term. Consequently, the scaling should be decreased in this case ($s = 2$) compared to the 140 dB experiments ($s = 3$, $s = 4$). All parameters are fixed over these three sets except for the optimization methods, which are `nls_gndl` with (`CGMaxIter` = 20) and with `largescale` = `false` (`CGMaxIter` = ∞), and `nls_lm`, respectively. It can be seen that the `largescale` version is slower in number of iterations but significantly faster in computation time compared to the other two methods. However, `nls_lm` finds the most solutions (without freezing). With freezing, the performance is again significantly improved for all three sets.

Algorithm freezing performs well for larger tensors and high noise levels. In the last set of Figure 5, η is further decreased to 10 dB and the size of the tensor is increased to $100 \times 100 \times 100$; see first paragraph. The number of CG iterations is decreased to 15 (the default value). As a consequence, the computation time stays relatively low. With freezing, 17 more solutions are found in less time and fewer iterations.

Algorithm freezing is more accurate and finds many more solutions according to the measure in (15). For all eight sets of experiments, significantly more solutions are found according to the criterion defined in (15).

Similar conclusions can be drawn from Figure 6, where the improvement for each individual experiment is visualized. The light blue lines indicate an improvement in iterations or computation time with a factor 2 or more and the red lines a worsening of a factor 2 or more. The gray lines are everything in between. The unsuccessful cases are indicated with a black cross. As can be seen, only when using `nls_gndl` with `largescale` = `false` and thus when using `pinv`, a significant amount of red lines are visible. This can, however, be explained by the wrong use of the `largescale` = `false` version in this case and the unpredictability of `pinv` for ill-conditioned linear systems.

Lastly, Figure 7 and Figure 8 present an alternative visualization for two sets of experiments. Each experiment can be individually compared in these figures. Two different noise levels are chosen: 40 and 10 dB, respectively. For Figure 8, the size of the tensor is increased to $100 \times 100 \times 100$ as explained at the beginning of this subsection. The following conclusions can be made.

Algorithm freezing obtains more accurate results in less computation time. In the upper two plots of Figure 7 and Figure 8, `nls_gndl` is compared with and without freezing. As can be seen, on almost all cases, Algorithm freezing is faster both in terms of iterations and computation time. The speed-up in computation time is even more significant because of the lower cost per iteration thanks to freezing. In the middle plots, it can be seen that both the standard cost function and the maximum value of the error `e` on the factor matrices, computed using `cpderr`, are almost always lowest for Algorithm freezing. The number of successful experiments with Algorithm freezing is significantly higher (+38 and +17, respectively).

Algorithm freezing outperforms other state-of-the-art methods to decompose difficult tensors, such as RGN-HR [17], in computation time. In the bottom two plots of Figure 7, `nls_gndl` with freezing is compared with RGN-HR [17]. Algorithm freezing is faster on all experiments. Note that the RGN-HR method cannot be used directly (without compression) anymore for the set of $100 \times 100 \times 100$ tensor because of the large number of variables as the computation of the condition number becomes too expensive in this case.

5 Conclusion

Swamps, or regions of slow convergence, occur frequently when computing a low-rank tensor approximation of more difficult-to-decompose tensors. We illustrate that, although far less often than for ALS, swamps also occur for second-order methods. For example, transient swamps occur more frequently when the rank-1 terms have different orders of magnitude or when the factor matrices have close to collinear columns. We observed experimentally that, during such a swamp, often only a few problematic or ill-conditioned rank-1 terms cause the slow convergence while the other ones stagnate. This ill-conditioning is not intrinsic to the solution and this region of the optimization landscape is merely a passage toward the solution. This observation formed the starting point for a fairly effective strategy to get through swamps. Namely, we have designed an optimization framework that freezes all but the problematic rank-1 terms temporarily in the optimization problem to reduce the cost per iteration and simplify the problem. As a consequence of the lower number of terms, compression, e.g., using

an MLSVD, can be performed such that a smaller tensor needs to be dealt with and smaller factor matrices need to be estimated. Moreover, an algebraic (re)initialization can be computed for the compressed residual tensor which may help speed up the convergence for the remaining (compressed) terms. A measure is defined to detect which terms stagnate, filtering out trivial scaling indeterminacies while remaining computationally efficient. In several numerical experiments using state-of-the-art second-order optimization methods, it is illustrated that the proposed framework is up to an order of magnitude faster for difficult-to-decompose tensors and is approximately as fast on easy-to-decompose tensors, both in the low and high SNR regime.

Declarations

Funding/acknowledgment

This work was funded by: (1) the Flemish Government: AI Research Program. Lieven De Lathauwer and Charlotte Vermeylen are affiliated to Leuven.AI - KU Leuven institute for AI, B-3000, Leuven, Belgium. (2) KU Leuven Internal Funds: iBOF/23/064, C14/22/096. (3) Nico Vervliet holds a research grant from the Research Foundation Flanders (FWO) (12ZM223N).

Code availability

The code of the proposed algorithm and to regenerate all results and figures will be made available on https://www.tensorlabplus.net/.

Authors' contributions

C.V. wrote the main manuscript, developed the algorithm, and performed the experiments. N.V. helped with the development of the algorithm, the experiments, and the start-up of the project, and reviewed the manuscript. L.D.L. proposed the original idea, helped with the development of the algorithm, and reviewed the manuscript.

Ethical Approval/Availability of supporting data/Competing interests

Not applicable.

Appendix A Additional experiment

A similar visualization as in Figure 7 and Figure 8 is shown in Figure A1 for a high SNR value of 140 dB. The scaling parameter s is also increased from 2 to 4. This is a different visualization of the data in the fourth column in Figure 5 and Figure 6. Similar conclusions as for Figure 7 and Figure 8 apply. Using Algorithm freezing, `nls_gndl` is on almost all experiments faster and on some experiments even an order of magnitude faster or more. Furthermore, 36 more solutions are found according to the measure in (15). It can be seen in the lower two plots that there is a significant difference in cost function between the successful and unsuccessful experiments because of the high SNR. There is also a clear gap between the maximum error on the factor matrices. Algorithm freezing is faster than RGN [17] for all experiments and in some cases even an order of magnitude faster.

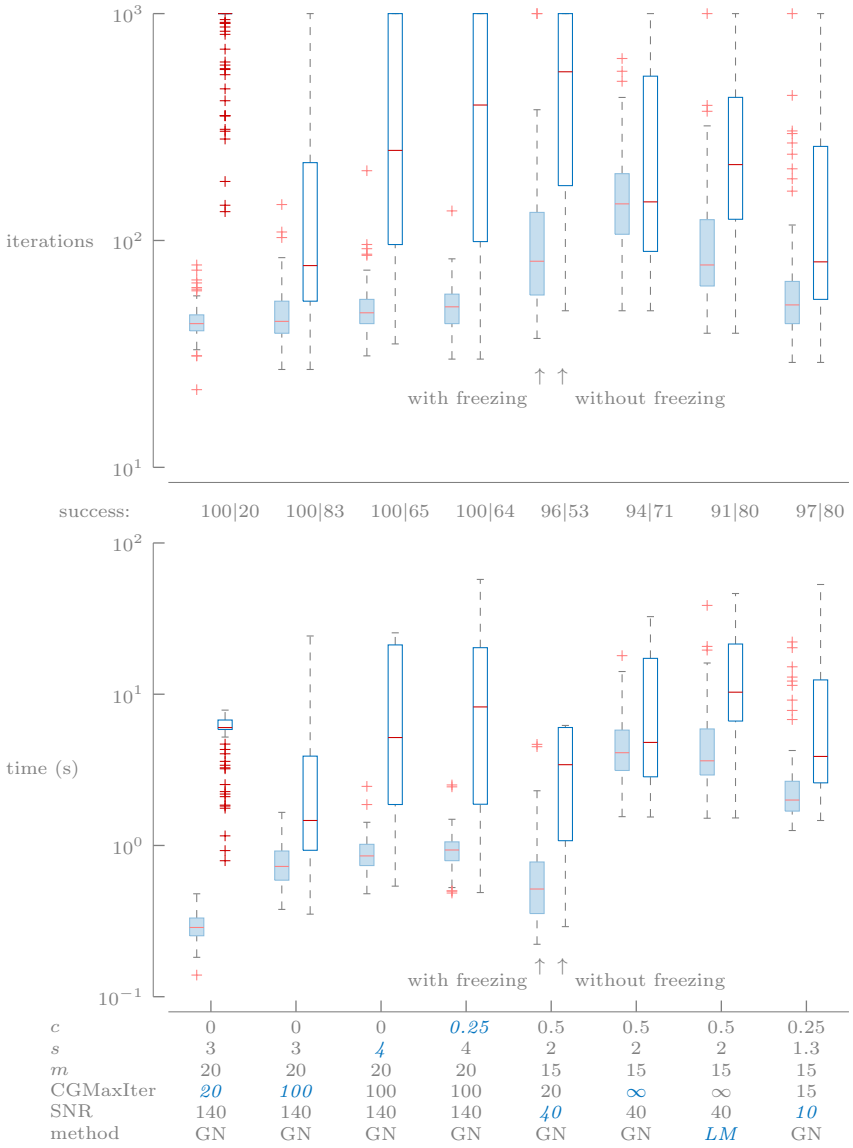

Fig. 5 *Algorithm freezing is robust and performs well for different parameters such as the number of CG iterations, collinearity c and scaling parameter s, using different inner optimization methods, and under different noise levels.* box plots of the iterations and computation time for eight sets of parameters of the model problem in (14) and for different methods. Each plot represents 100 experiments. Outliers are indicated with red crosses. The blue boxes indicate the 25 and 75 percentiles, the middle red line the median value, and the dashed lines the upper and lower adjacent. The parameter settings are indicated at the bottom. The most important parameter changes from left to right are indicated in blue italic. For every choice of parameters, the results with and without freezing are plotted on the left and right, respectively.

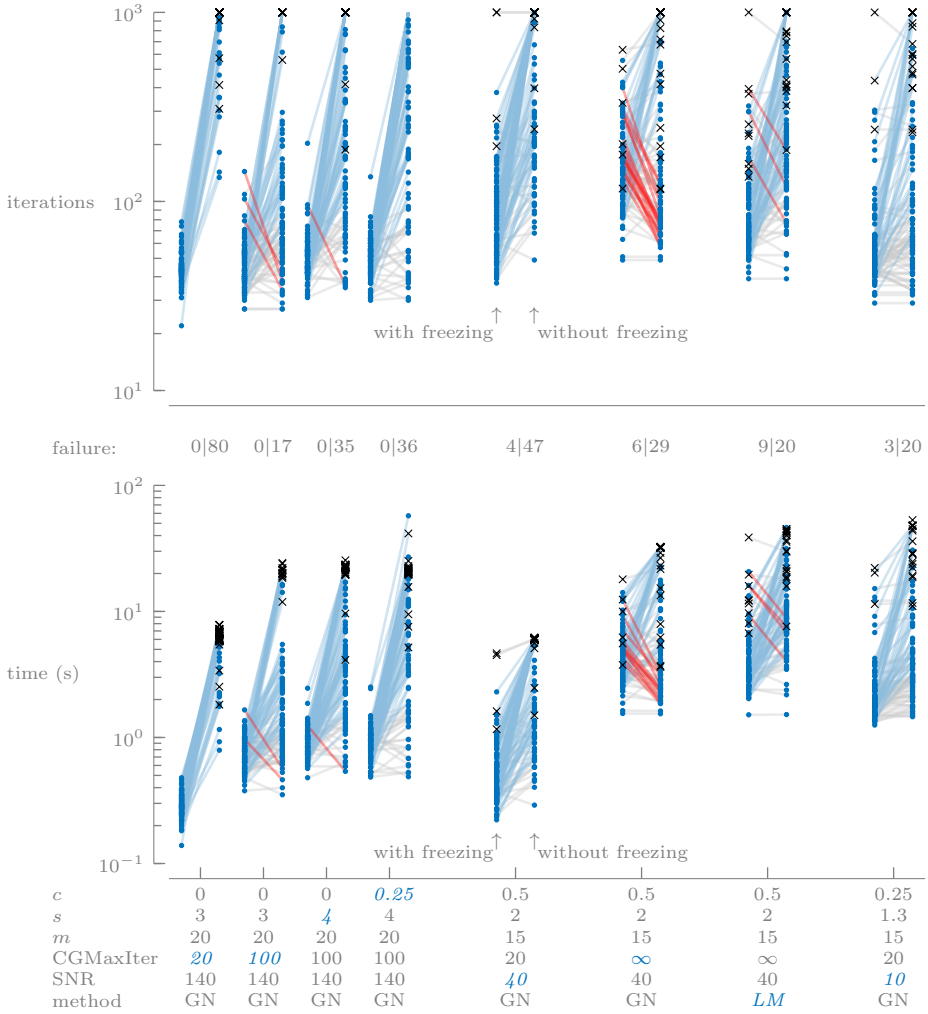

Fig. 6 *For various parameter settings of* (14)*, the freezing framework improves the number of iterations and computation time for a significant amount of individual cases with a factor 2 or more. Furthermore, the number of successful cases has significantly increased.* In each column, the `nls_gndl` or `nls_lm` method with and without the proposed freezing framework are compared for 100 experiments. The blue lines indicate an improvement in computation time of a factor 2 or more, the red lines a worsening in computation time with a factor 2 or more, and the gray lines all other cases. The crosses/dots indicate a un-/successful experiment, respectively, for the method at hand. The number of failed experiments (×) per set is indicated on the figure. The most important parameter changes from left to right are indicated in blue italic.

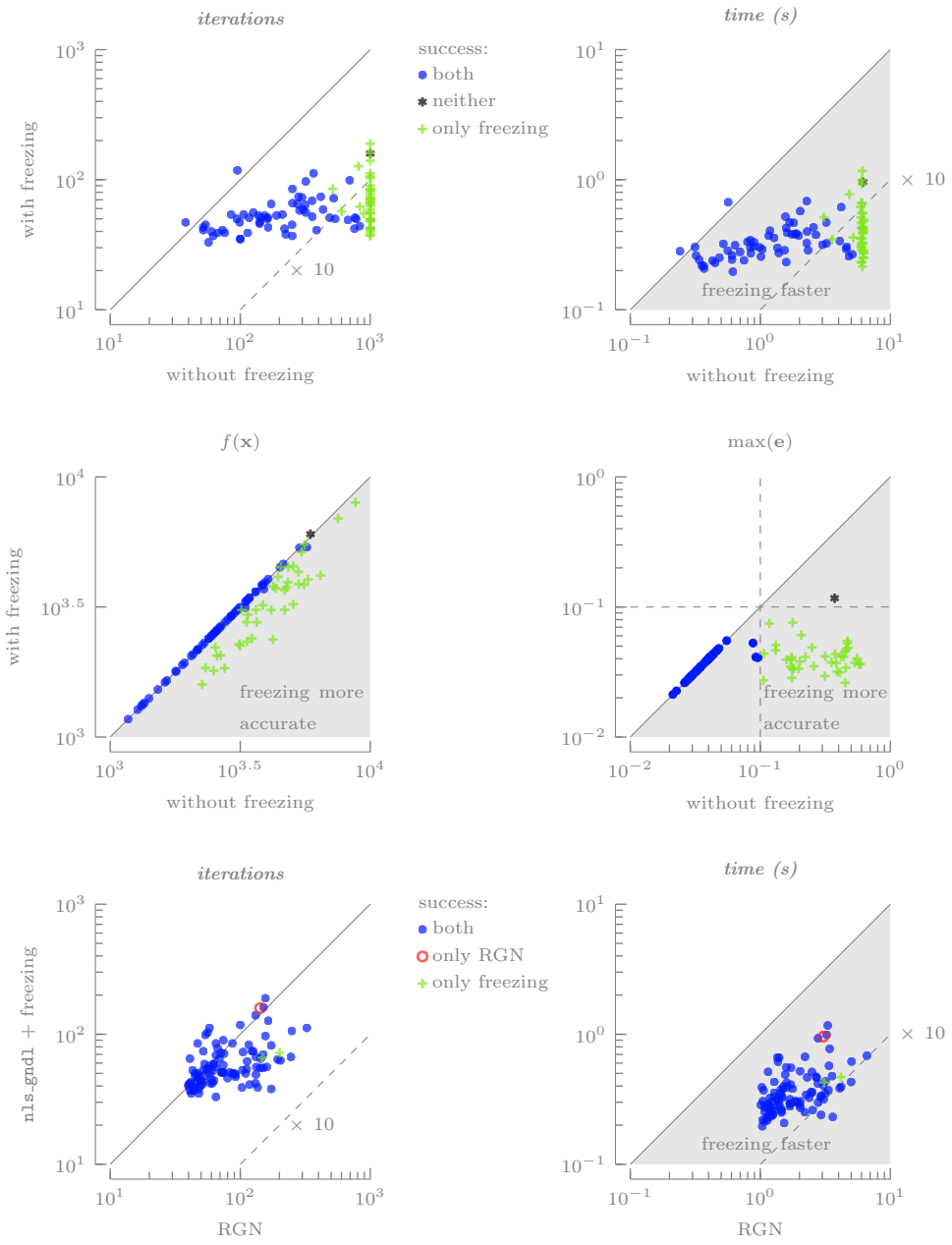

Fig. 7 *Illustration of the speed-up of* `nls_gndl` *using Algorithm freezing on 100 difficult-to-decompose tensors in the upper two plots and the improvements in accuracy in the middle two plots. The lower two plots show that* `nls_gndl` *with freezing is faster than RGN-HR [17] on all experiments.* The dashed lines in the middle-right plot indicate the threshold of 10^{-1} in (15). The parameters in (14) are: $c = 0.25$, $s = 2$, $\eta = 40$, and `CGMaxIter` $= 20$.

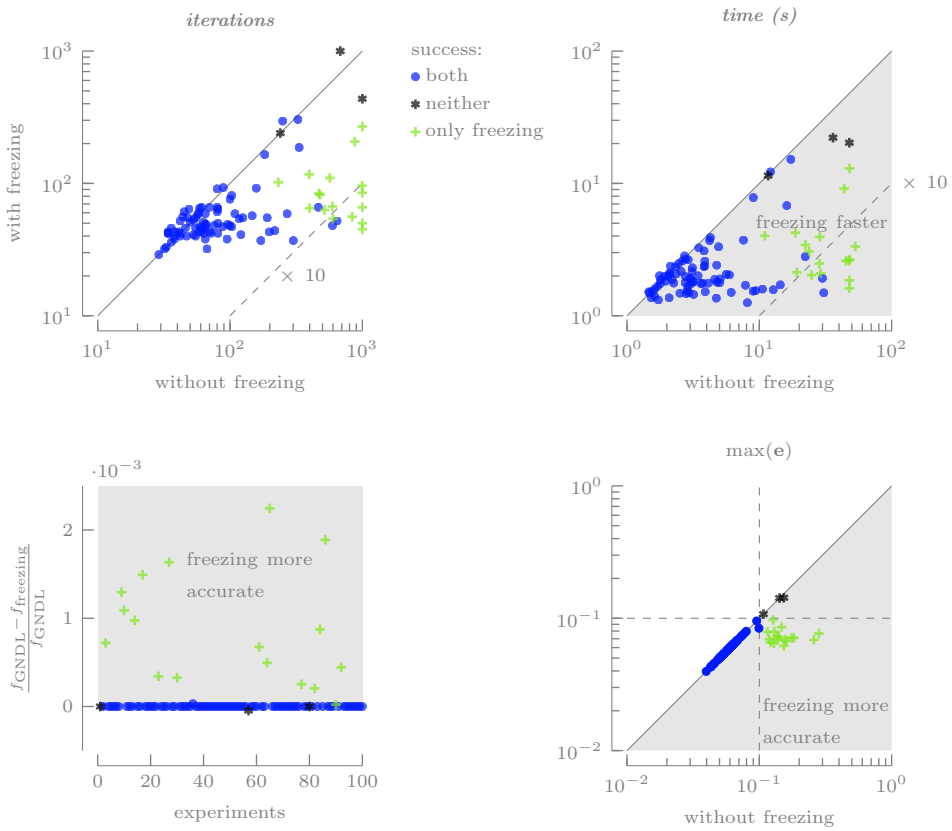

Fig. 8 *Algorithm freezing improves* `nls_gndl` *on almost all 100 difficult-to-decompose tensors both in terms of computation time (upper two plots) and in accuracy (lower two plots). Larger tensors and a low SNR value are used for these experiments.* The dashed lines in the lower-right plot indicate the threshold of 10^{-1} in (15). The parameters in (14) are: $c = 0.25$, $s = 1.3$, $\eta = 10$, and `CGMaxIter` $= 15$.

References

[1] Landsberg, J.M.: Tensors: Geometry and Applications vol. 128. American Mathematical Soc., Rhode Island (2011)

[2] Khoromskaia, V., Khoromskij, B.N.: Tensor Numerical Methods in Quantum Chemistry. De Gruyter, Berlin, Boston (2018). https://doi.org/10.1515/9783110365832 . https://doi.org/10.1515/9783110365832

[3] Hackbusch, W.: Tensor Spaces and Numerical Tensor Calculus vol. 42. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-030-35554-8

[4] Cichocki, A., Mandic, D., Phan, H.A., Caiafa, C., Zhou, G., Zhao, Q., De Lathauwer, L.: Tensor decompositions for signal processing applications: From two-way to multiway component analysis. IEEE Signal Processing Magazine **32**(2), 145–163 (2015) https://doi.org/10.1109/MSP.2013.2297439

[5] Sidiropoulos, N.D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E.E., Faloutsos, C.: Tensor decomposition for signal processing and machine learning. IEEE Transactions on Signal Processing **65**(13), 3551–3582 (2017) https://doi.org/10.1109/TSP.2017.2690524

[6] Lim, L.-H.: Tensors in computations. Acta Numerica **30**, 555–764 (2021) https://doi.org/10.1017/S0962492921000076

[7] Bi, X., Tang, X., Yuan, Y., Zhang, Y., Qu, A.: Tensors in statistics. Annual Review of Statistics and Its Application **8**(1), 345–368 (2021) https://doi.org/10.1146/annurev-statistics-042720-020816

[8] Franc, A.: Tensor Ranks for the Pedestrian for Dimension Reduction and Disentangling Interactions (2022). https://arxiv.org/abs/2201.07473

[9] Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Review **51**(3), 455–500 (2009) https://doi.org/10.1137/07070111X

[10] Harshman, R.A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-model factor analysis, vol. 16, pp. 1–84 (1970)

[11] Oseledets, I.V.: Tensor-train decompositions. Methods and Algorithms for Scientific Computing **33**(5), 2295–2317 (2011) https://doi.org/10.1137/090752286

[12] Tucker, L.R.: The extension of factor analysis to three-dimensional matrices. In: Gulliksen, H., Frederiksen, N. (eds.) Contributions to Mathematical Psychology., pp. 110–127. Holt, Rinehart and Winston, New York (1964)

[13] De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value

decomposition. SIAM Journal on Matrix Analysis and Applications **21**(4), 1253–1278 (2000) https://doi.org/10.1137/S0895479896305696

[14] Domanov, I., De Lathauwer, L.: On the uniqueness of the canonical polyadic decomposition of third-order tensors—Part I: Basic results and uniqueness of one factor matrix. SIAM Journal on Matrix Analysis and Applications **34**(3), 855–875 (2013) https://doi.org/10.1137/120877234

[15] Domanov, I., De Lathauwer, L.: On the uniqueness of the canonical polyadic decomposition of third-order tensors—Part II: Uniqueness of the overall decomposition. SIAM Journal on Matrix Analysis and Applications **34**(3), 876–903 (2013) https://doi.org/10.1137/120877258

[16] Kruskal, J.B.: Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. Linear algebra and its applications **18**(2), 95–138 (1977) https://doi.org/10.1016/0024-3795(77)90069-6

[17] Breiding, P., Vannieuwenhoven, N.: A Riemannian trust region method for the canonical tensor rank approximation problem. SIAM Journal on Optimization **28**(3), 2435–2465 (2018) https://doi.org/10.1137/17M114618X

[18] Acar, E., Dunlavy, D.M., Kolda, T.G.: A scalable optimization approach for fitting canonical tensor decompositions. Journal of chemometrics **25**(2), 67–86 (2011) https://doi.org/10.1002/cem.1335

[19] Phan, A.-H., Tichavský, P., Cichocki, A.: Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC. SIAM Journal on Matrix Analysis and Applications **34**(1), 126–147 (2013) https://doi.org/10.1137/100808034

[20] Sorber, L., Van Barel, M., De Lathauwer, L.: Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization. SIAM Journal on Optimization **23**(2), 695–720 (2013) https://doi.org/10.1137/120868323

[21] Bro, R., Andersson, C.A.: Improving the speed of multiway algorithms: Part II: Compression. Chemometrics and Intelligent Laboratory Systems **42**(1), 105–113 (1998) https://doi.org/10.1016/S0169-7439(98)00011-2

[22] Vervliet, N., Debals, O., De Lathauwer, L.: Exploiting efficient representations in large-scale tensor decompositions. SIAM Journal on Scientific Computing **41**(2), 789–815 (2019) https://doi.org/10.1137/17M1152371

[23] Vervliet, N., Debals, O., Sorber, L., Van Barel, M., De Lathauwer, L.: Tensorlab 3.0. Available online. https://www.tensorlab.net

[24] Kossaifi, J., Panagakis, Y., Anandkumar, A., Pantic, M.: TensorLy: Tensor

learning in Python. Journal of Machine Learning Research **20**(26) (2019)

[25] Bader, B.W., Kolda, T.G., et al.: Tensor Toolbox for MATLAB, Version 3.6. Available online. www.tensortoolbox.org

[26] Mitchell, B.C., Burdick, D.S.: Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies. Journal of Chemometrics **8**(2), 155–168 (1994) https://doi.org/10.1002/cem.1180080207

[27] Gong, X., Mohlenkamp, M.J., Young, T.R.: The optimization landscape for fitting a rank-2 tensor with a rank-1 tensor. SIAM Journal on Applied Dynamical Systems **17**(2), 1432–1477 (2018) https://doi.org/10.1137/17M112213X https://doi.org/10.1137/17M112213X

[28] Mohlenkamp, M.J.: The dynamics of swamps in the canonical tensor approximation problem. SIAM Journal on Applied Dynamical Systems **18**(3), 1293–1333 (2019) https://doi.org/10.1137/18M1181389

[29] Navasca, C., De Lathauwer, L., Kindermann, S.: Swamp reducing technique for tensor decomposition. In: 2008 16th European Signal Processing Conference, pp. 1–5 (2008)

[30] Gong, X., Mohlenkamp, M.J., Young, T.R.: The optimization landscape for fitting a rank-2 tensor with a rank-1 tensor. SIAM Journal on Applied Dynamical Systems **17**(2), 1432–1477 (2018) https://doi.org/10.1137/17M112213X

[31] Krijnen, W.P., Dijkstra, T.K., Stegeman, A.: On the non-existence of optimal solutions and the occurrence of "degeneracy" in the CANDECOMP/-PARAFAC model. Psychometrika **73**(3), 431–439 (2008) https://doi.org/10.1007/s11336-008-9056-1

[32] Paatero, P.: Construction and analysis of degenerate PARAFAC models. Journal of Chemometrics **14**(3), 285–299

[33] Domanov, I., De Lathauwer, L.: Canonical polyadic decomposition of third-order tensors: Relaxed uniqueness conditions and algebraic algorithm. Linear Algebra and its Applications **513**, 342–375 (2017) https://doi.org/10.1016/j.laa.2016.10.019

[34] Sanchez, E., Kowalski, B.R.: Tensorial resolution: A direct trilinear decomposition. Journal of Chemometrics **4**(1), 29–45 (1990) https://doi.org/10.1002/cem.1180040105

[35] Leurgans, S.E., Ross, R.T., Abel, R.B.: A decomposition for three-way arrays. SIAM Journal on Matrix Analysis and Applications **14**(4), 1064–1083 (1993)

[36] Evert, E., Vandecappelle, M., De Lathauwer, L.: A recursive eigenspace computation for the canonical polyadic decomposition. SIAM Journal on Matrix Analysis and Applications **43**(1), 274–300 (2022) https://doi.org/10.1137/21M1423026

[37] Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York (2006). https://doi.org/10.1007/978-0-387-40065-5

[38] Vannieuwenhoven, N., Vandebril, R., Meerbergen, K.: A new truncation strategy for the higher-order singular value decomposition. SIAM Journal on Scientific Computing **34**(2), 1027–1052 (2012) https://doi.org/10.1137/110836067

[39] Breiding, P., Vannieuwenhoven, N.: The condition number of join decompositions. SIAM Journal on Matrix Analysis and Applications **39**(1), 287–309 (2018) https://doi.org/10.1137/17M1142880

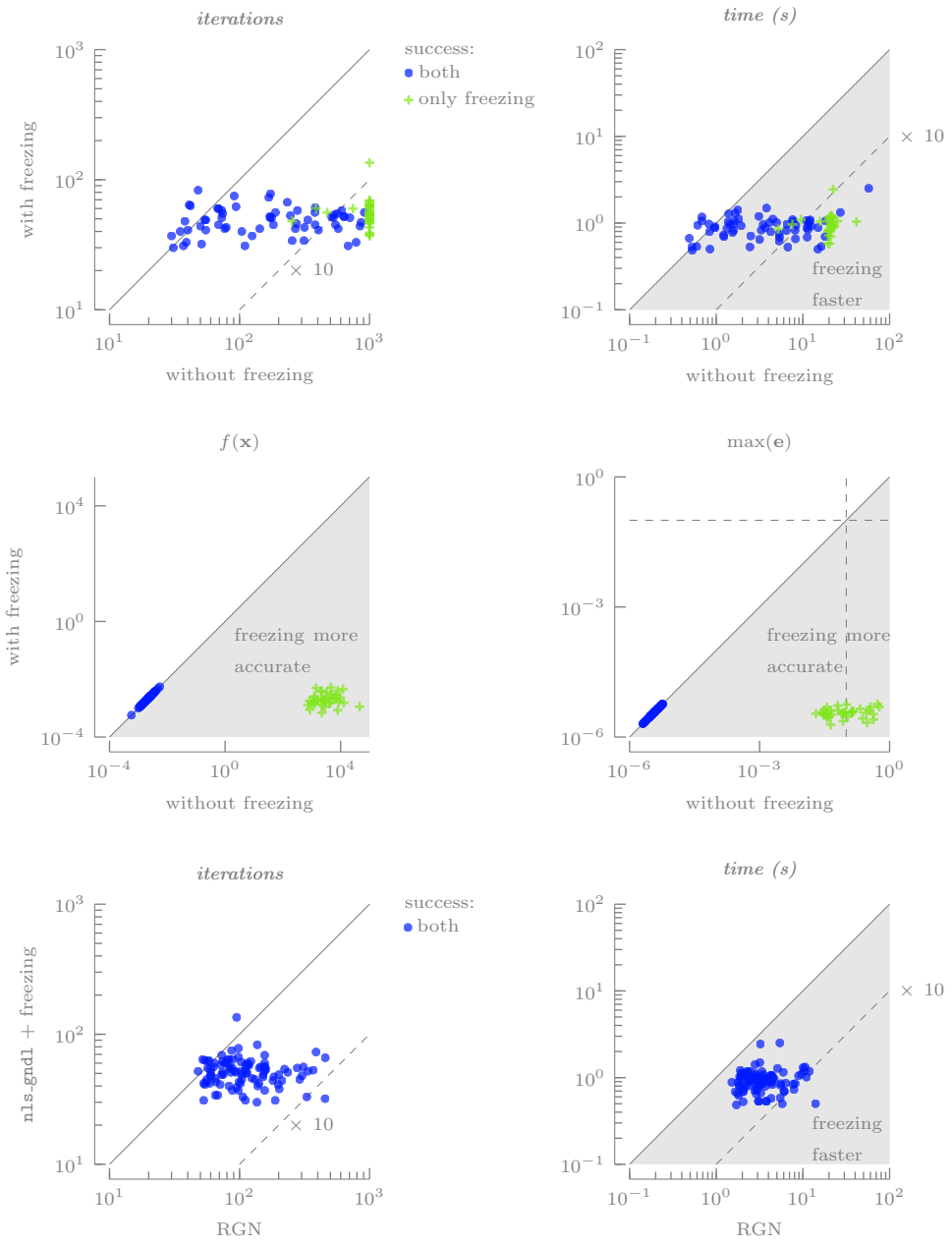

Fig. A1 *Illustration of the speed-up of* `nls_gndl` *using Algorithm freezing on 100 difficult-to-decompose tensors in the upper two plots and the improvements in accuracy in the middle two plots. The lower two plots show that* `nls_gndl` *is faster than RGN-HR [17] on all experiments.* The dashed lines in the middle-right plot indicate the threshold of 10^{-1} in (15). The parameters in (14) are: $c = 0.25$, $s = 4$, $\eta = 140$, and `CGMaxIter` $= 100$.