SIAM J. MATRIX ANAL. APPL.
Vol. 43, No. 2, pp. 840–866

NEW RIEMANNIAN PRECONDITIONED ALGORITHMS FOR TENSOR COMPLETION VIA POLYADIC DECOMPOSITION*

SHUYU DONG†, BIN GAO†, YU GUAN†, AND FRANÇOIS GLINEUR‡

Abstract. We propose new Riemannian preconditioned algorithms for low-rank tensor completion via the polyadic decomposition of a tensor. These algorithms exploit a non-Euclidean metric on the product space of the factor matrices of the low-rank tensor in the polyadic decomposition form. This new metric is designed using an approximation of the diagonal blocks of the Hessian of the tensor completion cost function and thus has a preconditioning effect on these algorithms. We prove that the proposed Riemannian gradient descent algorithm globally converges to a stationary point of the tensor completion problem, with convergence rate estimates using the Łojasiewicz property. Numerical results on synthetic and real-world data suggest that the proposed algorithms are more efficient in memory and time compared to state-of-the-art algorithms. Moreover, the proposed algorithms display a greater tolerance for overestimated rank parameters in terms of the tensor recovery performance and thus enable a flexible choice of the rank parameter.

Key words. tensor completion, polyadic decomposition, CP decomposition, Riemannian optimization, preconditioned gradient

AMS subject classifications. 15A69, 90C26, 90C30, 90C52

DOI. 10.1137/21M1394734

1. Introduction.

Tensor completion refers to the task of recovering missing values of a multidimensional array and can be seen as a generalization of the matrix completion problem. Similar to the approximation of a matrix with low-rank models, the approximation of a tensor can be formulated by a *low-rank* tensor model. Starting from this idea, low-rank tensor completion consists in finding a low-rank approximation of a tensor based on a given subset of its entries. Applications of low-rank tensor completion can be found in many areas, e.g., signal processing for EEG (brain signals) data [35] and magnetic resonance imaging [6] and image and video inpainting [7, 31, 27].

Depending on different tensor decomposition forms, there are several ways to define the rank of a tensor. Low-rank tensor decompositions provide a useful tool for tensor representation and are widely used in tensor completion [4, 11, 24, 2, 38, 47, 26]. The low-rank tensor decomposition paradigm allows for extracting the most meaningful and informative latent structures of a tensor, which usually contain heterogeneous and multiaspect data. The canonical polyadic (CP) decomposition [21, 29, 26], the Tucker or the multilinear decomposition [48, 14, 15], and the tensor-train decomposition [37, 16, 39] are among the most fundamental tensor decomposition forms. Other variants include hierarchical tensor representations [13, 40, 41] and PARAFAC2 models [40].

*Received by the editors January 27, 2021; accepted for publication (in revised form) by J. Nie December 30, 2021; published electronically May 24, 2022.
https://doi.org/10.1137/21M1394734
Funding: This work was supported by the Fonds de la Recherche Scientifique – FNRS and the Fonds Wetenschappelijk Onderzoek – Vlaanderen under EOS project 30468160 (SeLMA – Structured low-rank matrix/tensor approximation: numerical optimization-based algorithms and applications). The first author was supported by the FNRS through a FRIA scholarship.

†ICTEAM, UCLouvain, Louvain-la-Neuve, Belgium (shuyu.dong@uclouvain.be, gaobin@lsec.cc.ac.cn, yu.guan@uclouvain.be).
‡ICTEAM and CORE, UCLouvain, Louvain-la-Neuve, Belgium (francois.glineur@uclouvain.be).

Related work. For the tensor completion problem using the CP decomposition (CPD), Tomasi and Bro [47] proposed to used the Levenberg–Marquardt (modified Gauss–Newton) method for third-order tensors, in which the rank-R tensor candidate is represented by the vectorization of all three factor matrices of the CPD; Acar et al. [2] proposed to use a nonlinear conjugate gradient algorithm; Jain and Oh [23] proposed an alternating minimization algorithm, which uses a special initialization by the robust tensor power method [3] allowing for a guaranteed tensor recovery with a sample complexity lower bound. The Tucker decomposition, as a closely related decomposition format, is also widely used for tensor completion, with more or less different application purposes; see [12] about the relations and differences between the Tucker decomposition and the CPD. Kressner, Steinlechner, and Vandereycken [28] exploited the Riemannian geometry of the rank-constrained search space with the Tucker decomposition and proposed a Riemannian conjugate gradient (RCG) algorithm for the tensor completion problem. Kasai and Mishra [25] paid attention to the data fitting function of tensor completion and introduced a preconditioned metric on the quotient space of the rank-constrained search space with the Tucker decomposition; they used an RCG algorithm with respect to the proposed metric. Breiding and Vannieuwenhoven [10] proposed a Riemannian Gauss–Newton method for tensor approximation, which is based on the *Segre manifold* structure of tensors with a bounded rank. We refer to [43, 44, 45] for more recent work about CPD methods with fully observed or missing entries.

For an $m_1 \times \cdots \times m_k$ tensor that is only partially observed, low-rank tensor completion using CPD can be modeled by approximating the given partially observed tensor with a rank-R tensor candidate in the CPD form $\mathcal{T} = [\![U^{(1)}, \ldots, U^{(k)}]\!]$, where $U^{(i)}$ are $m_i \times R$ matrices with full column-rank and $[\![U^{(1)}, \ldots, U^{(k)}]\!]$ denotes the product of the CPD, which is the sum of the outer products of the respective columns of $U^{(i)}$. In such problem formulations, the CPD not only provides a powerful data representation but also has an advantageously low memory requirement—which scales as $O((m_1 + \cdots + m_k)R)$ for a given CP rank R—compared to other types of models (e.g., [31]) that involve otherwise a full dense tensor variable (requiring $O(m_1 m_2 \ldots m_k)$ memory). However, the fixed-rank CPD, as well as other tensor decomposition models with a fixed rank, requires the choice of an appropriate rank value. Since an optimal rank choice is usually unknown in practice, fixing the tensor rank in the CPD-based (as well as Tucker-based) approaches is not an ideal strategy. Unfortunately, the search or estimation of the optimal rank is also hard [30]. Therefore, the approach with a fixed CP rank limits the applicability of the completion model, and it is natural to study CPDs with a rank upper bound (e.g., [32]). For example, a rank-increasing approach is used (e.g., [50]) during the optimization process for the exploration of an optimal rank.

In this paper, we focus on the polyadic decomposition approach to tensor completion and consider the tensor candidate with a bounded CP rank. More precisely, the kth-order tensor variable is represented by k factor matrices in the polyadic decomposition form. Thus, we formulate the tensor completion problem as follows:

$$\underset{U:=(U^{(1)},\ldots,U^{(k)})\in\mathcal{M}}{\text{minimize}} \quad \frac{1}{2}\|\mathcal{P}_\Omega([\![U^{(1)}, \ldots, U^{(k)}]\!] - \mathcal{T}^\star)\|_{\mathrm{F}}^2 + \psi(U), \tag{1.1}$$

where $[\![U^{(1)}, \ldots, U^{(k)}]\!]$ denotes the tensor candidate in the form of polyadic decomposition, $\mathcal{T}^\star$ is the partially observed tensor, Ω is an index set indicating the observed entries of $\mathcal{T}^\star$, $\mathcal{P}_\Omega$ denotes the orthogonal projector such that the $(i_1, \ldots, i_k)$th entry

of $\mathcal{P}_\Omega(Z)$ is equal to $Z_{i_1,\dots,i_k}$ if $(i_1,\dots,i_k) \in \Omega$ and zero otherwise, and the search space $\mathcal{M}$ is a product space defined as

$$\mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R} \tag{1.2}$$

with a rank parameter R and where $\psi : \mathcal{M} \mapsto \mathbb{R}$ is a regularization function. Note that the search space of (1.1) includes points such that the actual CP rank of the product $[\![U^{(1)},\dots,U^{(k)}]\!]$ is smaller than the rank parameter R. Hence, the polyadic decomposition representation of the tensor candidate by $U \in \mathcal{M}$ is not necessarily *canonical*. In the context of low-rank tensor completion, the optimal rank of the tensor candidate is unknown, and thus, it is important to choose appropriate parameters of the low-rank model in question. One of the approaches (using low-rank tensor decomposition) in previous work is to explore and adjust the rank parameter R sequentially [50]. In the case of the model using the Tucker decomposition with a fixed Tucker rank [25], the natural way of choosing the Tucker rank is to explore among a range of (k-dimensional) trials.

In this work, we are interested in choosing a large enough rank parameter (while maintaining it in the range of values that are much lower than the tensor dimensions) for the model (1.1). To alleviate issues where R is overestimated compared to the rank of the desired solution, we design algorithms that tolerate rank-deficient factor matrices (or *almost* rank-deficient matrices, with close-to-zero tailing singular values). The proposed algorithms are shown in numerical results to be able to reach optimal solutions to (1.1), given a large enough value of R, and thus avoid lengthy sequential rank explorations.

Contributions. We design a *preconditioned* metric on the search space $\mathcal{M}$ of the polyadic decomposition model (1.1) and propose Riemannian gradient descent (RGD) and RCG algorithms to solve the tensor completion problem.

We prove that the sequence of iterates generated by the RGD algorithm converges to a critical point of the objective function and provide estimates of the convergence rate using the Łojasiewicz property.

We test on synthetic data for recovering a partially observed tensor with and without additive noise. The numerical results show that our algorithms outperform the several existing algorithms. On the real-world dataset (MovieLens 1M), we find that the proposed algorithms are also faster than the other algorithms under various rank choices. Moreover, the tensor recovery performance of our algorithms is not sensitive to the choice of the rank parameter, in contrast to algorithms based on a fixed-rank model.

Organization. We give the definitions and notation for the tensor operations and state the main problem in section 2. The proposed algorithms and convergence analysis are presented in sections 3–4. Numerical results are reported in section 5. We conclude the paper in section 6.

2. Preliminaries and problem statement. In this section, we introduce the definitions and notation involved in the tensor operations and give a concrete problem formulation of (1.1).

The term *tensor* refers to a multidimensional array. The dimensionality of a tensor is described as its order. A kth-order tensor is a k-way array, also known as a k-mode tensor. We use the term *mode* to describe operations on a specific dimension (e.g., mode-k matricization).

For a strictly positive integer n, we denote the index set $\{1,\dots,n\}$ as $[\![n]\!]$. The set of n-dimensional real-valued vectors is denoted by $\mathbb{R}^n$. For $\ell \in [\![n]\!]$, we denote

by $\mathbf{e}_\ell$ the (n-dimensional) vector that indicates 1 in the ℓth entry: $[\mathbf{e}_\ell]_\ell = 1$ and $[\mathbf{e}_\ell]_j = 0$ for all $j \neq \ell$. The set of kth-order real-valued tensors is denoted by $\mathbb{R}^{m_1 \times \cdots \times m_k}$. The ith row and the jth column of a matrix A are denoted by $A_{i,:}$ and $A_{:,j}$, respectively. An entry of a real-valued kth-order tensor $\mathcal{Z} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$ is accessed via a k-dimensional index $[\ell_j]_{j=1,\ldots,k}$ with $\ell_j \in [\![m_j]\!]$ and is denoted as $\mathcal{Z}_{\ell_1,\ldots,\ell_k}$. The inner product of two tensors $\mathcal{Z}^{(1)}, \mathcal{Z}^{(2)} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$ is defined as follows: $\langle \mathcal{Z}^{(1)}, \mathcal{Z}^{(2)} \rangle = \sum_{i_1=1}^{m_1} \cdots \sum_{i_k=1}^{m_k} \mathcal{Z}^{(1)}_{i_1,\ldots,i_k} \mathcal{Z}^{(2)}_{i_1,\ldots,i_k}$. The Frobenius norm of a tensor $\mathcal{Z}$ is defined as $\|\mathcal{Z}\|_{\mathrm{F}} = \sqrt{\langle \mathcal{Z}, \mathcal{Z} \rangle}$.

The following definitions are involved in the tensor computations. The Kronecker product of vectors $\mathbf{u} = [u_\ell] \in \mathbb{R}^{m_1}$ and $\mathbf{v} = [v_\ell] \in \mathbb{R}^{m_2}$ results in a vector $\mathbf{u} \otimes \mathbf{v} = [u_1 \mathbf{v}^T, u_2 \mathbf{v}^T, \ldots, u_{m_1} \mathbf{v}^T]^T \in \mathbb{R}^{m_1 m_2}$. The Khatri–Rao product of two matrices with the same number of columns $U \in \mathbb{R}^{m_1 \times R}$ and $V \in \mathbb{R}^{m_2 \times R}$ is defined and denoted as $U \odot V = [U_{:,1} \otimes V_{:,1}, \ldots, U_{:,R} \otimes V_{:,R}] \in \mathbb{R}^{m_1 m_2 \times R}$. The Hadamard product of two matrices A and B of the same dimensions, denoted by $A \star B$, is a matrix of the same dimensions by entrywise multiplications: $[A \star B]_{ij} = A_{ij} B_{ij}$. The mode-$\ell$ product of a given tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \cdots \times r_k}$ with a matrix $U \in \mathbb{R}^{m \times r_\ell}$, denoted as $\mathcal{G} \times_\ell U$, is a tensor of size $r_1 \times \cdots \times r_{\ell-1} \times m \times r_{\ell+1} \cdots \times r_k$, which has entries

$$[\mathcal{G} \times_\ell U]_{i_1,\ldots,i_{\ell-1},j,i_{\ell+1},\ldots,i_k} = \sum_{p=1}^{r_\ell} U_{j,p} \mathcal{G}_{i_1,\ldots,i_{\ell-1},p,i_{\ell+1},\ldots,i_k}$$

for $j \in [\![m]\!]$, $i_1 \in [\![r_1]\!]$, and so on. The mode-i matricization $\mathcal{Z}_{(i)}$ of a tensor $\mathcal{Z} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$, also called the unfolding of $\mathcal{Z}$ along the ith mode, is a matrix of size $m_i \times (\prod_{j \neq i} m_j)$ such that the tensor element $z_{\ell_1,\ldots,\ell_k}$ in $\mathcal{Z}$ is identified with the matrix element $[\mathcal{Z}_{(i)}]_{\ell_i, r_i}$ in $\mathcal{Z}_{(i)}$, where $r_i = 1 + \sum_{n=1, n\neq i}^{k} (\ell_n - 1) I_n$ with $\quad I_n = \prod_{j=1, j\neq i}^{n-1} m_j$. Using the (ℓ_i, r_i)-indexing, the mode-i matricization of the k-dimensional indices in an index set $\Omega \subset [\![m_1]\!] \times \cdots \times [\![m_k]\!]$ is defined in the same way as in the tensor matricization, and the mode-i matricization of Ω is an index set $\Omega_{(i)}$ such that $(\ell_i, r_i) \in \Omega_{(i)}$ if and only if $(\ell_1, \ldots, \ell_k) \in \Omega$.

DEFINITION 2.1 (Tucker decomposition). *The Tucker decomposition of a tensor* [48, 14, 15] *is defined with a series of mode-ℓ products between a core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \cdots \times r_k}$ and the (orthogonal) factor matrices $U^{(i)} \in \mathbb{R}^{m_i \times r_i}$ such that*

$$\mathcal{Z} = \mathcal{G} \times_1 U^{(1)} \times_2 \cdots \times_k U^{(k)}.$$

A tensor $\mathcal{Z}$ admitting a Tucker decomposition form with $\mathcal{G}$ and $U^{(1)}, \ldots, U^{(k)}$ can be unfolded along its ith mode into the following matricization:

$$\mathcal{Z}_{(i)} = U^{(i)} \mathcal{G}_{(i)} \left(U^{(i-1)} \otimes \cdots \otimes U^{(1)} \otimes U^{(k)} \otimes \cdots \otimes U^{(i+1)} \right)^T.$$

The tensor Tucker rank or multilinear rank [48] is defined as

$$\mathrm{rank}_{\mathrm{tc}}(\mathcal{Z}) = (\mathrm{rank}(\mathcal{Z}_{(1)}), \ldots, \mathrm{rank}(\mathcal{Z}_{(k)})),$$

where $\mathrm{rank}(\mathcal{Z}_{(i)})$ denotes the matrix rank for $i = 1, \ldots, k$.

A tensor $\mathcal{Z} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$ of the form $\mathcal{Z} = \mathbf{u}^{(1)} \circ \cdots \circ \mathbf{u}^{(k)}$, where $\mathbf{u}^{(i)} \in \mathbb{R}^{m_i}$ and $\circ$ denotes the outer product, is said to be a rank-1 tensor, which is also called a simple tensor [19] or decomposable tensor [18]. The CP rank of a tensor $\mathcal{Z}$ is defined as the minimum number of rank-1 tensors which sum to $\mathcal{Z}$ [21, 29]:

$$\mathrm{rank}_{\mathrm{CP}}(\mathcal{Z}) = \min\{R \in \mathbb{Z}_+ : \exists \{\mathbf{u}_r^{(1)}, \ldots, \mathbf{u}_r^{(k)}\}_{r=1,\ldots,R} \text{ s.t. } \mathcal{Z} = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \cdots \circ \mathbf{u}_r^{(k)}\}.$$

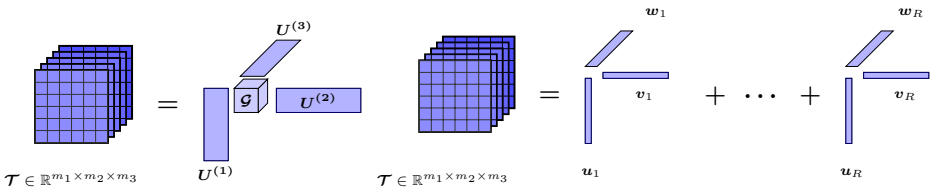


FIG. 1. *Left: Tucker decomposition. Right: CP decomposition.*

DEFINITION 2.2 (CPD). *Let $\mathcal{Z} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$ be a tensor such that* $\operatorname{rank}_{\mathrm{CP}}(\mathcal{Z}) = R$. *Then there exist k matrices $U^{(i)} \in \mathbb{R}^{m_i \times R}$ with full column-rank, for $i = 1, \ldots, k$, such that*

$$\mathcal{Z} = [\![U^{(1)}, \ldots, U^{(k)}]\!] = \sum_{r=1}^{R} U^{(1)}_{:,r} \circ \cdots \circ U^{(k)}_{:,r},$$

which is referred to as the CPD [21, 29, 26] *of $\mathcal{Z}$.*

The CPD can be considered as the "diagonalized" version of the Tucker decomposition (Definition 2.1) in the sense that a tensor that admits a rank-R CPD also admits a Tucker decomposition with an $R \times \cdots \times R$ hypercube as core tensor, whose nonzero values are only on the diagonal entries. Figure 1 illustrates the Tucker decomposition and CPD of a third-order tensor. The matricization of a tensor in the CPD form $\mathcal{Z} = [\![U^{(1)}, \ldots, U^{(k)}]\!]$ can be written using the Khatri–Rao product:

$$\mathcal{Z}_{(i)} = U^{(i)}[(U^{(j)})^{\odot_{j \neq i}}]^T := U^{(i)}(U^{(k)} \odot \cdots \odot U^{(i+1)} \odot U^{(i-1)} \odot \cdots \odot U^{(1)})^T. \tag{2.1}$$

A point in the search space $\mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R}$ of (1.1) is denoted by $U = (U^{(1)}, \ldots, U^{(k)})$, where $U^{(i)} \in \mathbb{R}^{m_i \times R}$ is the ith *factor matrix* of U. In fact, $\mathcal{M}$ is a smooth manifold, and the tangent space to $\mathcal{M}$ at any point $U \in \mathcal{M}$ is $T_U\mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R}$. Therefore, a tangent vector in $T_U\mathcal{M}$ is also a tuple of k factor matrices, denoted as $\xi = (\xi^{(1)}, \ldots, \xi^{(k)})$, where $\xi^{(i)} \in \mathbb{R}^{m_i \times R}$ for $i = 1, \ldots, k$. The Euclidean metric at U is defined and denoted as, for all $\xi, \eta \in T_U\mathcal{M}$, $\langle \xi, \eta \rangle = \sum_{i=1}^{k} \operatorname{tr}({\xi^{(i)}}^T \eta^{(i)})$, where $\operatorname{tr}(\cdot)$ is the trace of a matrix.

Let $\mathcal{M}$ be endowed with a Riemannian metric g; then the Riemannian gradient of a real-valued smooth function f at $U \in \mathcal{M}$, denoted as $\operatorname{grad} f(U)$, is the unique element in $T_U\mathcal{M}$ that satisfies, for all $\xi \in T_U\mathcal{M}$, $g_U(\operatorname{grad} f(U), \xi) = \mathrm{D}f(U)[\xi]$, where $\mathrm{D}f(U)$ denotes the (Euclidean) first-order differential of f at U. In particular, the Euclidean gradient of f at U is denoted as $\nabla f(U)$.

Problem statement. By setting the regularizer as $\psi(U) = \frac{\lambda}{2} \sum_{i=1}^{k} \|U^{(i)}\|_{\mathrm{F}}^2$, in a similar way as in maximum-margin matrix factorization [46], we specify the polyadic decomposition–based model (1.1) as follows:

$$\underset{U \in \mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R}}{\text{minimize}} \; f(U) := f_\Omega(U) + \frac{\lambda}{2} \sum_{i=1}^{k} \|U^{(i)}\|_{\mathrm{F}}^2, \tag{2.2}$$

where the first term $f_\Omega(U) := \frac{1}{2p} \|\mathcal{P}_\Omega([\![U^{(1)}, \ldots, U^{(k)}]\!] - \mathcal{T}^\star)\|_{\mathrm{F}}^2$ is referred to as the data fitting function of the problem, and $p = |\Omega|/(m_1 \cdots m_k)$ is a constant called the *sampling rate*. Note that when the regularization parameter $\lambda > 0$, the regularizer ψ has an effect of keeping the variable U in a compact subset of $\mathcal{M}$.

3. Algorithms. In this section, we propose a new metric on the manifold $\mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R}$. Under the proposed metric, we develop RGD and RCG algorithms on $\mathcal{M}$.

3.1. A preconditioned metric. The Riemannian preconditioned algorithms in [33, 25] on the product space of full column-rank matrices were proposed to improve the Euclidean gradient descent method. In these algorithms, a Riemannian metric was defined on the search space according to the differential properties of the cost function. Specifically, it is designed based on an operator that approximates the "diagonal blocks" of the second-order differential of the cost function. The chosen metric plays a role of preconditioning in the optimization algorithms such as RGD. We refer to [34] for a more general view on this topic of Riemannian preconditioning.

Starting from the idea of Riemannian preconditioning, we design a metric for the polyadic decomposition–based problem (1.1), in which the rank constraints are more relaxed than the fixed Tucker-rank constraint in [25]. First, we construct an operator $\mathcal{H}(U) : T_U\mathcal{M} \mapsto T_U\mathcal{M}$ using the "diagonal blocks" of the second-order differential of the data fitting function f_Ω in (2.2). Roughly speaking, such an operator satisfies

$$\langle \mathcal{H}(U)[\xi], \eta \rangle \approx \nabla^2 f_\Omega(U)[\xi, \eta] \tag{3.1}$$

for all $\xi, \eta \in T_U\mathcal{M}$, where $\nabla^2 f_\Omega(U)$ denotes the (Euclidean) second-order differential of f_Ω and $\langle \cdot, \cdot \rangle$ is the Euclidean metric. Then, we design a metric that behaves locally like $(\xi, \eta) \mapsto \langle \mathcal{H}(U)[\xi], \eta \rangle$. Assume that the operator $\mathcal{H}(U)$ is invertible and that $\tilde{g} : (\xi, \eta) \mapsto \langle \mathcal{H}(U)[\xi], \eta \rangle$ forms a Riemannian metric on $\mathcal{M}$; then the Riemannian gradient $\mathrm{grad} f_\Omega(U)$ satisfies, by definition, $\tilde{g}_U(\mathrm{grad} f_\Omega(U), \xi) = \mathrm{D} f_\Omega(U)[\xi] = \langle \nabla f_\Omega(U), \xi \rangle$ for all $\xi \in T_U\mathcal{M}$, where $\nabla f_\Omega(U)$ denotes the Euclidean gradient of f_Ω. Hence, $\mathrm{grad} f_\Omega(U) = \mathcal{H}(U)^{-1}[\nabla f_\Omega(U)]$. In view of $\mathcal{H}(U)$ as in (3.1), $\mathrm{grad} f_\Omega(U)$ is an approximation of the Newton direction of f_Ω, which results in a much improved convergence behavior than the Euclidean gradient descent.

For the tensor completion problem (1.1), one difficulty in designing such a metric using $\mathcal{H}(U)$ as in (3.1) is that we need to find an appropriate approximation to the second-order differential $\nabla^2 f_\Omega(U)$. This requires finding the explicit forms of the first- and second-order derivatives of f_Ω as defined in (2.2) and, hence, those of the CPD map $\Psi : \mathcal{M} \mapsto \mathbb{R}^{m_1 \times \ldots \times m_k} : U \mapsto [\![U^{(1)}, \ldots, U^{(k)}]\!]$. Concretely, we proceed as follows.

First, we deduce the second-order partial derivatives of f_Ω. Let U be a point in $\mathcal{M}$. The Euclidean gradient of f_Ω at $U \in \mathcal{M}$ is

$$\nabla f_\Omega(U) = (\partial_{U^{(1)}} f_\Omega(U), \ldots, \partial_{U^{(k)}} f_\Omega(U)), \tag{3.2}$$

where the partial derivatives have the following elementwise expression for $i = 1, \ldots, k$:

$$[\partial_{U^{(i)}} f_\Omega(U)]_{\ell,r} := \frac{\mathrm{d}}{\mathrm{d}t} f_\Omega(U^{(1)}, \ldots, U^{(i)} + tE_{\ell,r}, \ldots, U^{(k)})\Big|_{t=0}, \tag{3.3}$$

where $E_{\ell,r} = \mathbf{e}_\ell \mathbf{e}_r^T$. The right-hand side of (3.1) can be written in terms of the following second-order partial derivatives:

$$\nabla^2 f_\Omega(U)[\xi, \eta] := \sum_{i=1}^{k} \langle \partial_{ii}^2 f_\Omega(U)[\xi], \eta \rangle + \sum_{i=1, j' \neq i}^{k} \langle \partial_{ij'}^2 f_\Omega(U)[\xi], \eta \rangle,$$

where

$$\partial^2_{ij} f_\Omega(U)[\xi] := \frac{\mathrm{d}}{\mathrm{d}t}\left(\partial_{U^{(i)}} f_\Omega(U^{(1)},\dots,U^{(j)}+t\xi^{(j)},\dots,U^{(k)})\right)\Big|_{t=0} \tag{3.4}$$

for $i,j=1,\dots,k$. Subsequently, we construct $\mathcal{H}(U)$ as an operator on $T_U\mathcal{M}$ based on the action of the aforementioned "diagonal blocks" of $\nabla^2 f_\Omega(U)$, i.e., $\partial^2_{ii} f_\Omega(U)[\xi]$ for $i=1,\dots,k$. More accurately, to design a metric that works for all tensor completion problems with a certain subsampling pattern Ω, we use the expectation of these terms over the subsampling operator. Therefore, we define $\mathcal{H}(U): T_U\mathcal{M} \mapsto T_U\mathcal{M}$ as follows:

$$\mathcal{H}(U)[\xi] = \left(\mathbb{E}_\Omega\left[\partial^2_{11} f_\Omega(U)[\xi]\right],\dots,\mathbb{E}_\Omega\left[\partial^2_{kk} f_\Omega(U)[\xi]\right]\right). \tag{3.5}$$

Given the polyadic decomposition–based data fitting function f_Ω in (2.2), f_Ω can be rewritten in terms of $U^{(i)}$ and $(U^{(j)})^{\odot_{j\neq i}}$ through the mode-i tensor matricizations (2.1): $f_\Omega(U) = \frac{1}{2p}\|\mathcal{P}_{\Omega_{(i)}}(U^{(i)}((U^{(j)})^{\odot_{j\neq i}})^T - \mathcal{T}^\star_{(i)})\|_F{}^2$, where $\Omega_{(i)}$ is the mode-i matricization of Ω. Therefore, from (3.3), the first-order derivatives have the following expression:

$$\partial_{U^{(i)}} f_\Omega(U) = \frac{1}{p}\mathcal{S}_{(i)}(U^{(j)})^{\odot_{j\neq i}}, \tag{3.6}$$

where $\mathcal{S}_{(i)}$ is the mode-i matricization of the residual $\mathcal{S} := \mathcal{P}_\Omega\left(\llbracket U^{(1)},\dots,U^{(k)}\rrbracket - \mathcal{T}^\star\right)$. Combining (3.4) and (3.6), it follows that

$$\partial^2_{ii} f_\Omega(U)[\xi] = \frac{1}{p}\mathcal{P}_{\Omega_{(i)}}\left(\xi^{(i)}((U^{(j)})^{\odot_{j\neq i}})^T\right)(U^{(j)})^{\odot_{j\neq i}}. \tag{3.7}$$

Let Ω be a random index set of entries that are independent and identically distributed samples of the Bernoulli distribution: $(i_1,\dots,i_k)\in\Omega$, with probability p, for $(i_1,\dots,i_k)\in[\![m_1]\!]\times\cdots\times[\![m_k]\!]$. Then by taking the expectation over the index set Ω, (3.7) has the following approximation:

$$\mathbb{E}_\Omega\left[\partial^2_{ii} f_\Omega(U)[\xi]\right] = \xi^{(i)}((U^{(j)})^{\odot_{j\neq i}})^T(U^{(j)})^{\odot_{j\neq i}}. \tag{3.8}$$

Therefore, the operator $\mathcal{H}(U)$ as defined in (3.5) reads

$$\mathcal{H}(U)[\xi] = \left(\xi^{(1)}(U^{(j)})^{\odot_{j\neq 1}})^T(U^{(j)})^{\odot_{j\neq 1}},\dots,\xi^{(k)}(U^{(j)})^{\odot_{j\neq k}})^T(U^{(j)})^{\odot_{j\neq k}}\right). \tag{3.9}$$

Note that, by using the approximation (3.8), we finally obtain an operator $\mathcal{H}(U)$ that is independent of the subsampling index set Ω. Interestingly, the idea of Riemannian preconditioning using (3.1)–(3.9), for the problem (2.2), is similar to the Riemannian Gauss–Newton (RGN) method for minimizing $\frac{1}{2p}\|\Psi(U)-\mathcal{T}^\star\|^2_{\mathrm{F}}$ in [10], where the normal equation of their RGN method involves the Jacobian J of the CPD map $\Psi: U\mapsto\llbracket U^{(1)},\dots,U^{(k)}\rrbracket$. The operator (3.9) is thus similar to the term J^TJ in the RGN method.

A second difficulty is that the operator $\mathcal{H}(U)$ in (3.9) may not be invertible, since the R-by-R symmetric matrices in the form of $((U^{(j)})^{\odot_{j\neq i}})^T(U^{(j)})^{\odot_{j\neq i}}$ are not necessarily positive definite. To this end, we propose to regularize $\mathcal{H}(U)$ with the identity operator on $T_U\mathcal{M}$. This is done by *shifting* the aforementioned matrices by adding a constant diagonal matrix δI_R, where I_R denotes the R-by-R identity matrix and $\delta>0$ is a relatively small parameter. Consequently, we define the following inner product.

DEFINITION 3.1. *Given $U = (U^{(1)}, \ldots, U^{(k)}) \in \mathcal{M}$, let g_U be an inner product in $T_U\mathcal{M}$ as follows:*

$$g_U(\xi, \eta) = \sum_{i=1}^{k} \operatorname{tr}\left(\xi^{(i)} H_{i,U} (\eta^{(i)})^T\right) \quad \text{for } \xi, \eta \in T_U\mathcal{M}, \tag{3.10}$$

where $H_{i,U}$ is an R-by-R matrix defined as

$$H_{i,U} = ((U^{(j)})^{\odot_{j\neq i}})^T (U^{(j)})^{\odot_{j\neq i}} + \delta I_R \tag{3.11}$$

and $\delta > 0$ is a constant parameter.

Note that $H_{i,U}$ is positive definite even when there is a rank-deficient factor matrix among $\{U^{(i)}\}$. Moreover, g is smooth on $\mathcal{M}$ and is therefore a Riemannian metric.

Now, we consider $\mathcal{M}$ as a manifold endowed with the Riemannian metric (3.10). The associated norm of a tangent vector $\xi \in T_U\mathcal{M}$ is defined and denoted by $\|\xi\|_U = \sqrt{g_U(\xi, \xi)}$. Based on the Euclidean gradient of f in (3.2), the Riemannian gradient of f is, by definition,

$$\operatorname{grad} f(U) = \left(\partial_{U^{(1)}} f(U) H_{1,U}^{-1}, \ldots, \partial_{U^{(k)}} f(U) H_{k,U}^{-1}\right). \tag{3.12}$$

As mentioned in the beginning of this subsection, the Riemannian gradient (3.12) can be seen as the result of preconditioning of the Euclidean gradient $\nabla f(U)$ with the operator $\mathcal{H}(U)$ (up to a "rescaling" with δI_R). We refer to the new metric (3.10) as the *preconditioned metric* on $\mathcal{M}$.

3.2. The Riemannian preconditioned algorithms. With the gradient defined in (3.12) using Riemannian preconditioning, we adapt RGD and RCG algorithms (e.g., [1]) to solve the problem (2.2).

Algorithm 3.1. RGD.

Input: $f : \mathcal{M} \mapsto \mathbb{R}$, $x_0 \in \mathcal{M}$, tolerance $\epsilon > 0$; $t = 0$.
Output: x_t.

1: **while** $\|\operatorname{grad} f(x_t)\| > \epsilon$ **do**
2: Set $\eta_t = -\operatorname{grad} f(x_t)$. # See (3.12)
3: Set stepsize s_t through one of the rules (3.13), (3.14), or (3.15).
4: Update: $x_{t+1} = x_t + s_t\eta_t$; $t \leftarrow t + 1$.
5: **end while**

The Riemannian gradient algorithm is given in Algorithm 3.1, which consists mainly of setting the descent direction as the negative Riemannian gradient and selecting the stepsizes. Note that since the search space is $\mathcal{M} = \mathbb{R}^{m_1 \times R} \times \cdots \times \mathbb{R}^{m_k \times R}$, the retraction map in this algorithm (line 4) is chosen as the identity map. In line 3, given an iterate $x_t \in \mathcal{M}$ and $\eta_t \in T_{x_t}\mathcal{M}$, the stepsize s_t is chosen by one of the following three methods.

Stepsize by line minimization. The line minimization consists in computing a stepsize as follows:

$$s_t = \operatorname*{arg\,min}_{s>0} h(s) := f(x_t + s\eta_t). \tag{3.13}$$

With third-order tensors ($k = 3$), the solution can be obtained numerically by selecting from the roots of the derivative $h'(s)$, which is a polynomial of degree 5.

Backtracking line search with the Armijo condition. We first set up a trial stepsize s_t^0 using the classical strategy [36, section 3.4]: (i) when $t \leq 1$, $s_t^0 = 1$, (ii) when $t \geq 2$, $s_t^0 = 2(f(x_{t-1}) - f(x_{t-2}))/g_{x_{t-1}}(\eta_{t-1}, \mathrm{grad} f(x_{t-1}))$; then the stepsize s_t is returned by a backtracking procedure, i.e., finding the smallest integer $\ell \geq 0$ such that

$$f(x_t) - f\left(x_t + s_t\eta_t\right) \geq \sigma s_t g_{x_t}(-\mathrm{grad} f(x_t), \eta_t) \tag{3.14}$$

for $s_t := \max(s_t^0\beta^\ell, s_{\min})$ with a constant parameter $s_{\min} > 0$. The backtracking parameters are fixed with $\sigma, \beta \in (0, 1)$.

The Barzilai–Borwein stepsize. Recently, the Riemannian Barzilai–Borwein (RBB) stepsize [22] has proven to be an efficient stepsize rule for Riemannian gradient methods. Hence, we choose the stepsize as

$$s_t^{\mathrm{RBB1}} := \frac{\|z_{t-1}\|_{x_t}^2}{|g_{x_t}(z_{t-1}, y_{t-1})|} \quad \text{or} \quad s_t^{\mathrm{RBB2}} := \frac{|g_{x_t}(z_{t-1}, y_{t-1})|}{\|y_{t-1}\|_{x_t}^2}, \tag{3.15}$$

where $z_{t-1} = x_t - x_{t-1}$ and $y_{t-1} = \mathrm{grad} f(x_t) - \mathrm{grad} f(x_{t-1})$.

The RCG algorithm is similar to RGD (Algorithm 3.1) in terms of the stepsize selection (line 3) and the update step (line 4) but differs with RGD in the choice of the search direction (line 2). More specifically, the search direction of RCG is defined as

$$\eta_t = -\mathrm{grad} f(x_t) + \beta_t \eta_{t-1},$$

where β_t is the Conjugate gradient (CG) parameter. In the numerical experiments, we choose the Riemannian version [9] of the modified Hestenes–Stiefel rule [20] as follows:

$$\beta_t = \max\left(0, \frac{g_{x_t}(\xi_t - \xi_{t-1}, \xi_t)}{g_{x^t}(\xi_t - \xi_{t-1}, \eta_{t-1})}\right).$$

The vector transport operation involved in the computation of β_t is chosen to be the identity map.

In both algorithms, the cost for calculating the Riemannian gradient (3.12) is a dominant term of the total cost. Therefore, we propose an efficient method for evaluating the Riemannian gradient in the next section.

3.3. Computation of the gradient. We focus on the computation of the Riemannian gradient of f in (2.2). From the definition (3.12), the computation of $\mathrm{grad} f(U) = (\eta^{(1)}, \ldots, \eta^{(k)})$ consists of two parts: (i) computing the partial derivatives $D_i := \partial_{U^{(i)}} f(U) \in \mathbb{R}^{m_i \times R}$ and (ii) computing the matrix multiplications $\eta^{(i)} = D_i H_{i,U}^{-1}$. From the expressions (3.6) and (3.11), we have

$$D_i = \frac{1}{p} \underbrace{\mathcal{S}_{(i)}(U^{(j)})^{\odot_{j\neq i}}}_{\tilde{D}_i} + \lambda U^{(i)}, \tag{3.16}$$

$$\eta^{(i)} = D_i \underbrace{(((U^{(j)})^{\odot_{j\neq i}})^T (U^{(j)})^{\odot_{j\neq i}} + \delta I_R)}_{H_{i,U}}{}^{-1}. \tag{3.17}$$

In a straightforward manner, these two parts require mainly the following operations:

1. Computing the sparse tensor $\mathcal{S}$ as in (3.6), which requires $2|\Omega|R$ flops.
2. Computing $(U^{(j)})^{\odot_{j\neq i}}$ for $i = 1, \ldots, k$, which requires $2\sum_{i=1}^k m_{-i} R$ flops, where $m_{-i} := \prod_{j\neq i} m_j$.

3. Forming the sparse matricizations $\mathcal{S}_{(i)}$ for $i = 1, \dots, k$, which takes some extra time for input/output with the sparse tensor $\mathcal{S}$ and the matricizations of the index set Ω.
4. Computing the sparse-dense matrix products $\tilde{D}_i := \mathcal{S}_{(i)}(U^{(j)})^{\odot_{j\neq i}}$ for $i = 1, \dots, k$, which require $2k|\Omega|R$ flops. Then, one has access to $\{D_i\}_{i=1,\dots,k}$ after the matrix additions with $\lambda U^{(i)}$ (whose cost is not listed out since it is fixed regardless of the computational method).
5. Computing the R-by-R matrix $H_{i,U}$ (3.11) based on the matrix $(U^{(j)})^{\odot_{j\neq i}}$ (obtained in step 2), which consists of a dense-dense matrix multiplication of sizes $R \times m_{-i}$ and $m_{-i} \times R$ for $i = 1, \dots, k$, which mainly requires $2\sum_{i=1}^{k} m_{-i}R^2$.
6. Computing $D_i H_{i,U}^{-1}$ given D_i (obtained in step 4) and $H_{i,U}$ (obtained in step 5), through Cholesky decomposition of $H_{i,U}$ for $i = 1, \dots, k$, which requires $\sum_{i=1}^{k} 2m_i R^2 + C_{\text{chol}} R^3$.

The sum of the flops counted in the above list of operations is

$$2(k+1)|\Omega|R + \Big(\sum_{i=1}^{k} 2m_{-i}(R^2+R)\Big) + \Big(\sum_{i=1}^{k} 2m_i R^2 + C_{\text{chol}} R^3\Big). \tag{3.18}$$

An efficient computational method. We propose a computational method that avoids the matricizations of the residual tensor $\mathcal{S}$ and the computations of the Khatri–Rao products $(U^{(j)})^{\odot_{j\neq i}}$.

Given the residual tensor $\mathcal{S}$ after step 1 above, we propose to compute $\tilde{D}_i$ in (3.16) without passing through steps 2 and 3. In fact, the computation of $\tilde{D}_i = \mathcal{S}_{(i)}(U^{(j)})^{\odot_{j\neq i}}$ corresponds to the matricized tensor times Khatri–Rao product (MTTKRP), which is a common routine in the tensor computations. Through basic tensor computations, the entrywise expression of this MTTKRP does not require forming the matricizations of $\mathcal{S}$ explicitly. For brevity, we demonstrate these relations concretely in the case of third-order tensors ($k = 3$), knowing that their extension to higher-order tensors is straightforward. The matrix $\tilde{D}_1 = \mathcal{S}_{(1)}(U^{(j)})^{\odot_{j\neq 1}} \in \mathbb{R}^{m_1 \times R}$ in (3.16) has the entrywise expression below:

$$\left[\tilde{D}_1\right]_{i_1 \ell} = \sum_{i_2=1}^{m_2} \sum_{i_3=1}^{m_3} \mathcal{S}_{i_1 i_2 i_3} U^{(3)}_{i_3 \ell} U^{(2)}_{i_2 \ell} \tag{3.19}$$

for $(i_1, i_2, i_3) \in [\![m_1]\!] \times \cdots \times [\![m_3]\!]$ and $\ell = 1, \dots, R$. Based on (3.19), Algorithm 3.2 presents an efficient way to compute the MTTKRPs of (3.16), with a sparse residual tensor $\mathcal{S}$ as input. Note that the computations of $\tilde{D}_2$ and $\tilde{D}_3$ correspond to the same equation (3.19) but with the indices (i_1, i_2, i_3) swapped via the rotations $(1, 2, 3; 2, 3, 1)$ and $(1, 2, 3; 3, 1, 2)$, respectively.

Subsequently, we propose to compute $H_{i,U}$ (3.11) without large matrix multiplications. In fact, the large matrix multiplications with $(U^{(j)})^{\odot_{j\neq i}}$ in (3.11) can be decomposed into smaller ones. Note that these matrix multiplications satisfy the following identity:

$$\left((U^{(j)})^{\odot_{j\neq i}}\right)^T \left((U^{(j)})^{\odot_{j\neq i}}\right) = G_k \star \cdots G_{i+1} \star G_{i-1} \star \cdots \star G_1, \tag{3.20}$$

where $G_j := {U^{(j)}}^T U^{(j)}$ for $j \neq i$ and the product by $\star$ denotes the Hadamard product. Using this property, the computation of $H_{i,U}$ reduces to computing $G_j = {U^{(j)}}^T U^{(j)}$

Algorithm 3.2. Sparse MTTKRP.

Input: The index sets by axis $I_\Omega := \{i : (i,j,k) \in \Omega\}$, J_Ω, and K_Ω. The sparse tensor $\mathcal{S}$ in the form of a $|\Omega|$-by-1 array of observed entries $\{S_p = \mathcal{S}_{i_p,j_p,k_p} : (i_p,j_p,k_p) \in \Omega\}$. The factor matrices $(U^{(i)})_{i=1,2,3}$.

Output: $\tilde{D} := \mathcal{S}_{(1)}U^{(3)} \odot U^{(2)}$.

```
D̃ = 0.
for p = 1, ..., |Ω| do
    for ℓ = 1, ..., R do
        D̃_{i_p ℓ} = D̃_{i_p ℓ} + S_p U^(3)_{k_p ℓ} U^(2)_{j_p ℓ}.
    end for
end for
```

and then the entrywise multiplications between the (small) R-by-R matrices $\{G_j\}$, which require only $\sum_{i=1}^{k}(2m_i+k-1)R^2$ flops, since the computation of the matrices G_j cost $2\sum_{i=1}^{k} m_i R^2$ and the entrywise multiplications between G_j cost $(k-1)R^2$.

In summary, the operations reduce to the following steps.

a. Computing the sparse tensor $\mathcal{S}$ as in (3.6). This is identical to step 1 above, which requires $2|\Omega|R$ flops;
b. Computing $\tilde{D}_i := \mathcal{S}_{(i)}(U^{(j)})^{\odot_{j\neq i}}$ for $i = 1,\ldots,k$, using $\mathcal{S}$ (obtained in step a) and U; see Algorithm 3.2. The computational cost of this step is $2k|\Omega|R$. Then, one has access to $\{D_i\}_{i=1,\ldots,k}$ after the matrix additions with $\lambda U^{(i)}$.
c. Computing the R-by-R matrix $H_{i,U}$ (3.11) using U (input data); see (3.20). The computational cost of this step is $\sum_{i=1}^{k}(2m_i+k-1)R^2$.
d. Computing $D_i H_{i,U}^{-1}$ given D_i (obtained in step b) and $H_{i,U}$ (obtained in step c), through Cholesky decomposition of $H_{i,U}$ for $i = 1,\ldots,k$. This is identical to step 6 above, which requires $\sum_{i=1}^{k} 2m_i R^2 + C_{\text{chol}}R^3$ flops.

Therefore, the total cost of the above steps is

$$2(k+1)|\Omega|R + \sum_{i=1}^{k}(4m_i+k-1)R^2 + C_{\text{chol}}R^3,$$

which is significantly reduced compared to the cost (3.18) of the naive method. In particular, for third-order (or a bit higher-order) tensors ($k \ll m_i$) with a low rank parameter R, the dominant term in (3.18) is $2(k+1)|\Omega|R+\sum_{i=1}^{k}(2m_{-i}+2m_i)R^2$, while the dominant term in the cost of the proposed method is $2(k+1)|\Omega|R+\sum_{i=1}^{k} 4m_i R^2$. The reduction in the cost can be seen from the fact that $m_i \ll m_{-i} = \prod_{j\neq i} m_j$ and $m_i \ll |\Omega| = pm_1 \ldots m_k$. Note that on top of the above reduction in flops, the time efficiency is further improved as the matricizations of the residual tensor $\mathcal{S}$ are not needed. In particular, speedups related to step b (instead of steps 2–4 in the naive method) are demonstrated in Table 3.

4. Convergence analysis. In this section, we analyze the convergence behavior of Algorithm 3.1. Let $\{x_t\}_{t\geq 0}$ denote the sequence generated by this algorithm. First, we demonstrate in Proposition 4.3 that every accumulation point of $\{x_t\}_{t\geq 0}$ is a stationary point. Second, we analyze the iterate convergence property of the algorithm in Theorem 4.5.

The following lemma generalizes the class of functions with Lipschitz-continuous gradient to functions defined on Riemannian manifolds and will be used in Lemma 4.2.

LEMMA 4.1 (see [8, Lemma 2.7]). *Let $\mathcal{M}' \subset \mathcal{M}$ be a compact Riemannian submanifold. Let $\mathcal{R}_x : T_x\mathcal{M}' \mapsto \mathcal{M}'$. If $f : \mathcal{M}' \mapsto \mathbb{R}$ has Lipschitz continuous gradient in the convex hull of $\mathcal{L}$, then there exists $L > 0$ such that, for all $x \in \mathcal{M}'$ and $\xi \in T_x\mathcal{M}'$,*

$$|f(\mathcal{R}_x(\xi)) - (f(x) + g_x(\xi, \mathrm{grad} f(x))| \leq \frac{L}{2}\|\xi\|_x^2. \tag{4.1}$$

Starting from the above lemma, we show that the proposed algorithm ensures a sufficient decrease property at each iteration.

LEMMA 4.2. *Let $\{x_t\}_{t\geq 0}$ be the sequence generated by Algorithm* 3.1 *(RGD), in which the stepsizes are chosen by either line minimization* (3.13) *or Armijo line search* (3.14)*. For all $t \geq 0$, there exists $\bar{C} > 0$ such that*

$$f(x_t) - f(x_{t+1}) \geq \bar{C}\|\mathrm{grad} f(x_t)\|_{x_t}^2. \tag{4.2}$$

In particular, with the stepsizes chosen by line minimization (3.13)*, there exists a Lipschitz-like constant $L_0 > 0$ such that* (4.2) *holds for $\bar{C} = \frac{1}{2L_0}$.*

Proof. Due to the fact that the objective function is coercive (because of the Frobenius norm–based terms), the sublevel set $\mathcal{L} = \{x \in \mathcal{M} : f(x) \leq f(x_0)\}$ is a closed and bounded subset of $\mathcal{M}$. Due to the boundedness of $\mathcal{L}$, the convex hull of $\mathcal{L}$, denoted as $\bar{\mathcal{L}}$, is bounded. Therefore, f has Lipschitz continuous gradient in $\bar{\mathcal{L}}$ since $f \in C^2(\mathcal{M})$. From Lemma 4.1, there exists a Lipschitz-like constant $L_0 > 0$ such that (4.1) holds. The inequality (4.1) ensures an upper bound of $f(\mathcal{R}_x(s\xi)) = f(x+s\xi)$ as follows: $f(x+s\xi) \leq f(x)+g_x(s\xi, \mathrm{grad} f(x))+\frac{L_0}{2}\|s\xi\|_x^2$, for all $s \geq 0$. Consequently, when the stepsize $s_t = s^*$ is selected by line minimization (3.13), we have

$$\begin{aligned} f(x_{t+1}) &= f(x_t - s^*\mathrm{grad} f(x_t)) \\ &\leq \min_{s\geq 0}\left(f(x) - s\left(1 - \frac{L_0 s}{2}\right)\|\mathrm{grad} f(x_t)\|_{x_t}^2\right) = f(x_t) - \bar{C}\|\mathrm{grad} f(x_t)\|_{x_t}^2 \end{aligned}$$

with $\bar{C} = \frac{1}{2L_0}$. When the stepsize s_t is selected using Armijo line search, the new iterate $x_{t+1} = x_t - s_t\mathrm{grad} f(x_t)$ is an Armijo point, where $s_t \geq s_{\min} > 0$, by construction of the line search procedure (with the parameter of lower bound of stepsizes $s_{\min}$). Hence, through (3.14), we have

$$f(x_t) - f(x_{t+1}) \geq \sigma s_t\|\mathrm{grad} f(x_t)\|_{x_t}^2 \geq \bar{C}\|\mathrm{grad} f(x_t)\|_{x_t}^2,$$

where $\bar{C} = \sigma s_{\min} > 0$ with the line search parameter $\sigma \in (0, 1)$.

In conclusion, the sufficient decrease property is satisfied with the two stepsize selection methods in the statement. ∎

PROPOSITION 4.3. *The sequence $\{x_t\}_{t\geq 0}$ generated by Algorithm* 3.1*, with stepsizes chosen by either line minimization* (3.13) *or Armijo line search* (3.14)*, satisfies the following convergence properties:* (i) *every accumulation point is a stationary point;* (ii) *the algorithm needs at most $\lceil \frac{(f^*-f(x_0))}{\bar{C}}\frac{1}{\epsilon^2}\rceil$ iterations to reach an ϵ-stationary solution for a constant $\bar{C} > 0$.*

Proof. (i) Let $x_* \in \mathcal{M}$ be an accumulation point; then there exists a subsequence $(x_{k(t)})_{t\geq 0}$, where $\{k(t) : t \geq 0\} \subset \mathbb{N}$, such that $\lim_{t\to\infty}(f(x_{k(t)}) - f(x_*)) = 0$. This

entails that $\sum_{t=0}^{\infty} f(x_{k(t)}) - f(x_{k(t+1)}) = f(x_{k(0)}) - f(x_*) < \infty$. Applying the sufficient decrease property (4.2) of Lemma 4.2 to this previous inequality, we have

$$\sum_{t=0}^{\infty} \bar{C}\|\mathrm{grad} f(x_{k(t)})\|^2_{x_{k(t)}} \leq \sum_{t=0}^{\infty} f(x_{k(t)}) - f(x_{k(t+1)}) < \infty \tag{4.3}$$

for a constant $\bar{C} > 0$. Therefore, $\lim_{t\to\infty} \|\mathrm{grad} f(x_{k(t)})\|_{x_{k(t)}} = 0$. (ii) Suppose that the algorithm does not attain an ϵ-stationary point (a point on which the gradient norm is bounded by ϵ) at iteration $T-1$; then $\|\mathrm{grad} f(x_t)\| > \epsilon$ for all $0 \leq t \leq T-1$. Using (4.2), we have $f(x_0) - f(x_T) \geq \bar{C} \sum_{t=0}^{T-1} \|\mathrm{grad} f(x_t)\|^2_{x_t} \geq \bar{C}\epsilon^2 T$. Therefore $T \leq \frac{f(x_0)-f(x_*)}{\bar{C}} \frac{1}{\epsilon^2}$. ∎

Next, we prove the iterate convergence of the RGD algorithm in Theorem 4.5 using the Łojasiewicz property. We first give the definition of the Łojasiewicz inequality for functions defined on a Riemannian manifold [42].

Definition 4.4 (Łojasiewicz inequality [42, Definition 2.1]). *Let $\mathcal{M} \subset \mathbb{R}^n$ be a Riemannian submanifold of $\mathbb{R}^n$. The function $f : \mathcal{M} \mapsto \mathbb{R}$ satisfies a Łojasiewicz gradient inequality at a point $x \in \mathcal{M}$ if there exist $\delta > 0$, $\sigma > 0$, and $\theta \in (0, 1/2]$ such that for all $y \in \mathcal{M}$ with $\|y - x\| \leq \delta$, it holds that*

$$|f(x) - f(y)|^{1-\theta} \leq \sigma \|\mathrm{grad} f(y)\|, \tag{4.4}$$

where θ is called the Łojasiewicz exponent.

Proposition 2.2 of [42] guarantees that (4.4) is satisfied for real analytic functions defined on an analytic manifold. Since the objective function of (2.2) is indeed real analytic and the search space $\mathcal{M}$ is an analytic manifold, the Łojacisiewicz inequality (4.4) holds. Consequently, we have the following iterate convergence result.

Theorem 4.5. *Let $\{x_t\}_{t\geq 0}$ be the sequence generated by Algorithm 3.1 with stepsizes chosen by either line minimization (3.13) or Armijo line search (3.14). Then $\{x_t\}_{t\geq 0}$ converges to a stationary point $x_* \in \mathcal{M}$. Moreover, the local convergence rate of $\{x_t\}_{t\geq 0}$ follows:*

$$\|x_t - x_*\| \leq C \begin{cases} e^{-ct} & \text{if } \theta = 1/2, \\ t^{-\theta/(1-2\theta)} & \text{otherwise}, \end{cases}$$

with the Łojacisiewicz exponent $\theta \in (0, 1/2]$ and constants $c > 0$ and $C > 0$.

Proof. The inequality (4.2) ensures that the sequence $\{x_t\}_{t\geq 0}$ is monotonically decreasing. Hence, the RGD algorithm satisfies the conditions in [42, Theorem 2.3]. More precisely, it follows from (4.2) of Lemma 4.2 that

$$\begin{aligned} |f(x_{t+1}) - f(x_t)| &\geq \bar{C}\|\mathrm{grad} f(x_t)\|^2_{x_t} = (\bar{C}/s_t)\|x_{t+1} - x_t\|_{x_t} \|\mathrm{grad} f(x_t)\|_{x_t} \\ &\geq \kappa_0 \|x_{t+1} - x_t\|_{x_t} \|\mathrm{grad} f(x_t)\|_{x_t}, \end{aligned} \tag{4.5}$$

where $\kappa_0 > 0$, since with line minimization (3.13), $s_t = s^* > 0$ for all $t \geq 0$ is chosen from a finite number of numerical solutions, and with Armijo line search (3.14), $0 < s_{\min} \leq s_t \leq s_t^0$, where $s_t^0 > 0$ is the initial stepsize before backtracking. In addition, the RGD update rule ensures that

$$\|x_{t+1} - x_t\|_{x_t} = s_t \|\mathrm{grad} f(x_t)\|_{x_t} \geq \kappa \|\mathrm{grad} f(x_t)\|_{x_t} \tag{4.6}$$

for $\kappa > 0$. The result of the theorem is obtained by combining (4.5), (4.6), and the Łojasiewicz inequality (4.4) and using [42, Theorem 2.3]. ∎

As far as we know, the global convergence of the RGD algorithm using RBB stepsizes without line search is not known. However, one can apply the RBB stepsize as an initial trial stepsize to the backtracking line search procedure, and consequently, all the convergence results in this section can be proved. Interested readers are referred to [22] for details.

5. Experiments. In this section, we carry out numerical experiments for tensor completion using the proposed algorithms and several existing algorithms in the related work. Details of these algorithms are as follows.

The proposed Algorithm 3.1 is labeled as Precon RGD, and the proposed RCG algorithm is labeled as Precon RCG. Depending on the stepsize selection method, these algorithms are labeled with a descriptor (i) line minimization (linemin) for the stepsize rule (3.13) and (ii) RBB for (3.15). We choose to restrict ourselves to linemin and RBB in our experiments since they appear to show better performances in practice and are easier to use than the Armijo line search rule; see Appendix A for a detailed discussion.

Euclidean gradient descent (Euclidean GD) and nonlinear CG (Euclidean CG) algorithms refer to the algorithms using the Euclidean gradient (3.2) in the definition of the search directions on $\mathcal{M}$. The stepsize selection rules are the same as the proposed algorithms; these algorithms are implemented along with the proposed algorithms in the source code. INDAFAC is a damped Gauss–Newton method for CPD-based tensor completion proposed by Tomasi and Bro [47]. CP-WOPT [2] is a nonlinear CG algorithm for CPD-based tensor completion. AltMin [17] is an alternating minimization algorithm for CPD-based tensor completion, which uses the linear CG for each of the least squares subproblems.

KM16 refers to a Riemannian optimization algorithm proposed by Kasai and Mishra [25] for tensor completion with a fixed Tucker rank. The Riemannian gradient in this algorithm is defined under a metric selected through Riemannian preconditioning on the manifold corresponding to a (fixed-rank) Tucker decomposition. In this algorithm, the tensor candidate is represented by a tuple of factor matrices and a core tensor via the Tucker decomposition. In our experiments on third-order tensors, this algorithm is labeled KM16 (r_1, r_2, r_3) according to the Tucker rank (r_1, r_2, r_3) with which it is tested. Note that the dimension of the search space of KM16 is $\sum_{i=1}^{k} (m_i r_i - r_i^2) + \prod_{i=1}^{k} r_i$, which is different than the dimension of $\mathcal{M}$ (search space of the CPD/Polyadic decomposition–based algorithms); in particular, the difference in these dimensions is marginal when $r_i \approx R$ for $i = 1, 2, 3$, with $R \ll \min(m_1, \ldots, m_k)$.

All the CPD/Polyadic decomposition–based algorithms are initialized with the same randomly generated point on $\mathcal{M}$, and the Tucker decomposition–based algorithm (KM16) is initialized with a point such that its tensor representation is close enough to that of the initial point of the other algorithms; see Appendix B for details.

All numerical experiments were performed on a workstation with 8-core Intel Core i7-4790 CPUs and 32GB of memory running Ubuntu 16.04 and MATLAB R2019. The source code is available at https://gitlab.com/shuyudong.x11/tcprecon/. Implementations of the existing algorithms are also publicly available.

5.1. Synthetic data.

Tensor model. We consider a low-rank tensor model that is composed of a low Tucker-rank tensor and independent additive noises: with a given Tucker-rank parameter $r^\star = (r_1^\star, r_2^\star, r_3^\star)$, we generate such a tensor $\mathcal{T}^\star$ using the following procedure:

$$\mathcal{T}^\star = \mathbb{T}_{r^\star}(\mathcal{T}) + \mathcal{E}, \tag{5.1}$$

where $T \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ is a third-order tensor composed of independent and identically distributed Gaussian entries; that is, $T_{ijk} \sim \mathcal{N}(0,1)$, the operator in the form of $\mathbb{T}_r(\cdot)$ is a Tucker-rank (rank_{tc}) truncation operator defined as the best Tucker-rank-r approximation of $\mathcal{T}$. The truncation $\mathbb{T}_r(\cdot)$ can be obtained using existing implementations that are available in state-of-the-art tensor toolboxes (e.g., Tensor Toolbox [5] and Tensorlab [49]). Here we use the function `tucker_als.m` in the MATLAB Tensor Toolbox. In the scenario of noiseless observations, $\mathcal{E} = 0$; otherwise $\mathcal{E} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ contains independent noises such that $\mathcal{E}_{\ell_1 \ell_2 \ell_3} \sim \mathcal{N}(0, \sigma)$, where σ is set according to a given signal-to-noise ratio; see Appendix B.

Low-rank tensor recovery from partial, noiseless observations. A synthetic tensor $\mathcal{T}^\star$ is generated with the model (5.1) without noise. The tensor $\mathcal{T}^\star$ is only observed on an index set Ω, which is composed of indices drawn from the Bernoulli distribution: $(i,j,k) \in \Omega$ with probability $p \in (0,1)$ for all $(i,j,k) \in [\![m_1]\!] \times [\![m_2]\!] \times [\![m_3]\!]$.

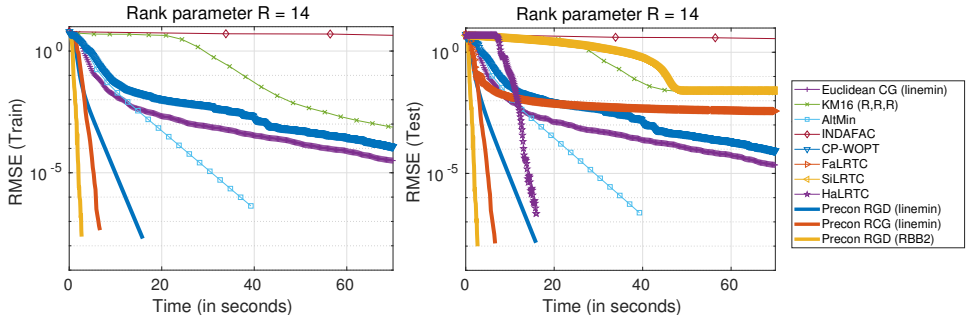
For the problem model (2.2), we set the regularization parameter λ to zero, which allows for recovering the low-rank tensor $\mathcal{T}^\star$ without any bias, provided that the sampling rate p is sufficient. Then we test the aforementioned algorithms with a given rank parameter R, assuming that the rank (CP or Tucker rank) of the hidden tensor $\mathcal{T}^\star$ is unknown to all the algorithms. For the CPD- and polyadic decomposition–based algorithms, we set the rank parameter R to an arbitrary value such that $R \geq \max(r_1^\star, r_2^\star, r_3^\star)$. Since the optimal CP rank of the tensor candidate is unknown, a larger-than-expected rank parameter is interesting because it allows for searching solutions in a fairly large tensor space, so there is a better chance that optimal solutions are in the search space $\mathcal{M}$. For parameter δ of (3.11) involved in our Riemannian gradients, we choose to use very small values since we are mostly interested in the performance of the Riemannian preconditioning technique. In all the experiments of section 5, we set $\delta = 10^{-7}$.

The termination of the proposed algorithms (Precon RGD and Precon RCG) is controlled by a tolerance parameter ($\epsilon = 10^{-7}$ in this experiment) against the norm of the gradient; KM16 uses a Riemannian CG algorithm with Armijo line search and default stopping criteria. On top of their respective stopping criteria, all the algorithms are tested within heuristic iteration budget maxiter $= 1000$ and $T_{\max} = 100$ seconds. Note that for all tested algorithms except AltMin, one iteration corresponds to one pass over the whole training data $P_\Omega(\mathcal{T}^\star)$; for AltMin, one iteration corresponds to multiple passes over the training data, since each of its iteration has a number of inner iterations for solving the underlying alternating subproblem. Table 1 shows the performances of the tested algorithms in terms of recovery errors and time, under the sampling rate $p = 0.3$ and rank parameters $R \in \{12, 14, 16\}$. The iteration histories of these algorithms (including KM16) with $R = 14$ are shown in Figure 2. Specifically, for the fixed Tucker-rank algorithm, KM16, we also test several other rank parameters than $r = (R, R, R)$; its tensor recovery performances along with those of the proposed algorithms are presented in Table 2. In Table 2, "#variables" indicates the dimension of the search space of each of algorithms, depending on the rank parameters.

From the results shown in Figure 2 and Tables 1–2, we have the following observations: (i) For all three values of the rank parameter R that are larger than $\max(r_1^\star, r_2^\star, r_3^\star)$, the proposed algorithms and HaLRTC [31] succeeded in recovering exactly the true hidden tensor $\mathcal{T}^\star$ (with a test root-mean-square error (RMSE) lower than 10^{-6}). AltMin, CP-WOPT, and Euclidean CG successfully made exact recoveries only with one or two of the rank parameter choices, and their convergences are slower than the proposed algorithms by orders of magnitude. The test error of KM16 stagnated at a certain level as the core tensor dimensions chosen are not exactly the

TABLE 1
Tensor completion with noiseless observations. The size of $\mathcal{T}^\star$ is $(100, 100, 200)$ with a Tucker rank $r^\star = (3, 5, 7)$. The sampling rate is 0.3. The rank parameters R tested are $\{12, 14, 16\}$.

Algorithm	R	Iter	Time (s)	RMSE (test)	RMSE (train)
Euclidean CG (linemin)	12	592	100.03	1.57e-12	2.03e-12
AltMin	12	67	100.97	8.05e-06	8.09e-06
INDAFAC	12	7	129.91	3.97e-01	4.57e-01
CP-WOPT	12	405	29.96	5.74e-07	6.70e-07
HaLRTC	12	142	15.81	2.19e-07	–
Precon RGD (linemin)	12	96	16.25	3.84e-08	4.79e-08
Precon RCG (linemin)	12	48	8.23	1.96e-08	3.58e-08
Precon RGD (RBB2)	12	65	**3.91**	9.52e-09	4.74e-07
Euclidean CG (linemin)	14	518	100.11	2.20e-06	3.10e-06
AltMin	14	19	39.27	2.39e-07	4.35e-07
INDAFAC	14	5	125.77	1.35e+00	2.60e+00
CP-WOPT	14	1561	94.29	2.40e-06	3.35e-06
HaLRTC	14	142	15.97	2.19e-07	–
Precon RGD (linemin)	14	82	15.90	1.38e-08	1.99e-08
Precon RCG (linemin)	14	34	6.65	1.29e-08	4.39e-08
Precon RGD (RBB2)	14	39	**2.69**	9.84e-09	2.38e-08
Euclidean CG (linemin)	16	456	100.01	1.37e-07	1.53e-07
AltMin	16	8	31.79	2.67e-08	3.32e-08
INDAFAC	16	5	154.34	9.78e-01	1.19e+00
CP-WOPT	16	1766	99.02	5.23e-06	7.00e-06
HaLRTC	16	142	16.34	2.19e-07	–
Precon RGD (linemin)	16	40	8.82	1.56e-09	2.51e-09
Precon RCG (linemin)	16	50	11.09	2.59e-09	5.08e-09
Precon RGD (RBB2)	16	39	**3.03**	1.53e-10	3.14e-10



(a) Training RMSEs (b) Test RMSEs

FIG. 2. *Tensor completion from noiseless observations. The size of $\mathcal{T}^\star$ is $(100, 100, 100)$ with a Tucker rank $r^\star = (3, 5, 7)$. The sampling rate is 0.3. The rank parameter is set as $R = 14$.*

same as the Tucker rank of $\mathcal{T}^\star$. (ii) Among the algorithms that successfully recovered the true hidden tensor, the proposed algorithms (Precon RGD and Precon RCG) outperform AltMin in time with a speedup of around 10 times, and they achieve speedups between 2 and 6 times compared to HaLRTC. Especially, Precon RGD (RBB2) has the fastest convergence behavior due to the fact that the RBB stepsize implicitly involves second-order information through a rough approximation of the Hessian. (iii) For KM16 specifically, the time efficiency and the recovery performance of KM16 (r, r, r) improves significantly when the core tensor dimensions (r, r, r) decrease (and get closer to $r^\star$). In particular, when r is only 1/2 of the rank parameter

TABLE 2

*Tensor completion with the noiseless observations. Proposed algorithms vs KM*16 (r, r, r) *with different choices of* r. *The size of* $\mathcal{T}^\star$ *is* (100, 100, 200) *with a Tucker rank* $r^\star = (3, 5, 7)$.

Algorithm	R	#variables	Iter	Time (sec.)	RMSE (test)
KM16 (R, R, R)	–	7756	29	102.20	2.70e-02
KM16 (12, 12, 12)	–	6096	30	88.32	9.00e-04
KM16 (9, 9, 9)	–	4086	21	29.63	2.70e-05
KM16 (7, 7, 7)	–	2996	22	14.39	2.99e-05
Precon RGD (linemin)	14	5600	82	15.90	1.38e-08
Precon RCG (linemin)	14	5600	34	6.65	1.29e-08
Precon RGD (RBB2)	14	5600	39	**2.69**	9.84e-09

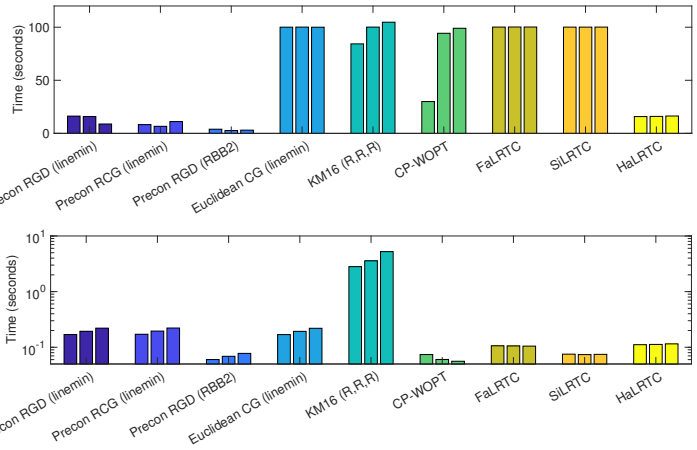


FIG. 3. *Computation time for tensor completion tasks. Top: Total time when reaching the target accuracy* $\epsilon = 10^{-7}$ *or the time budget* $T_{\max} = 100$ *seconds. Bottom: Average time per iteration. The three bars of each algorithm correspond to the cases with* $R = 12$, 14, *and* 16, *respectively. The size of* $\mathcal{T}^\star$ *is* $(100, 100, 200)$ *with a Tucker rank* $r^\star = (3, 5, 7)$. *The sampling rate is* 0.3.

$R = 14$, the time efficiency of KM16 (r, r, r) gets close to those of the proposed algorithms. Note that when $r \approx R/2$, the dimensions of the search space of KM16 (r, r, r) is much smaller than that of the proposed algorithms; see Table 2. These comparisons can be explained by the fact that the per-iteration cost of KM16 is much larger than the proposed algorithms, even when the dimensions of its search space is close or smaller than that of the proposed algorithms; see Figure 3 for detailed comparisons of their average per-iteration time. The high computational cost of KM16 lies in its computation of the Riemannian gradient, which scales poorly with the Tucker rank as it involves computing the Gram matrix of the matricizations of the core tensor—with a cost of $O(r_1 r_2 r_3 (r_1 + r_2 + r_3))$—and solving Lyapunov equations (for $k = 3$ times) of the $r_i \times r_i$ matrices.

Furthermore, we make a more thorough test to evaluate the tensor completion performances of the proposed algorithms, alongside Euclidean CG for comparison, in the same tensor completion task on a 6×10 grid of $(p, R) \in \{0.12, 0.152, \ldots, 0.28\} \times \{4, 6, \ldots, 16, 17, 19, 21\}$, for 5 random runs at each (p, R) (each run is based on a randomly generated index set Ω); see results in Figure 8 in Appendix C. These results provide a broader view on the difficulties—in terms of chances of exactly recovering $\mathcal{T}^\star$—of the tensor completion task with different sampling rates and rank parameter values.

Finally, in addition to tensor completion, we extend the application of the proposed algorithms to tensor approximation, where the tensor $\mathcal{T}^\star$ is a fully observed tensor and the goal is to approximate $\mathcal{T}^\star$ with a tensor with an upper-bounded CP rank. We give performance comparisons between the proposed algorithms and the RGN of [10]; see details in Appendix C.

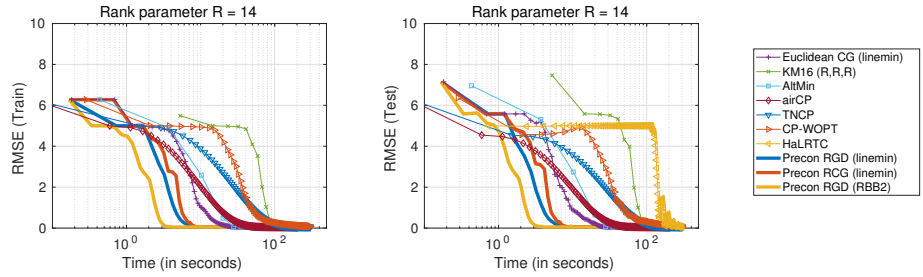
Low-rank tensor recovery from partial, noisy observations. In the following experiments, we conduct tensor completion tests under the same tensor model as in the previous experiment, except that the revealed tensor entries are observed with additive noise, and the noise level σ in the model (5.1) is set according to a given signal-to-noise ratio of 40 dB. To make this experiment similar to experiments on real data (MovieLens 1M), in which case the sampling rate is usually at the order of 1% or even lower, we set the sampling rate $p = 5\%$.

In this experiment, the regularization parameter λ of the problem (2.2) is selected from $\{0, 1/p, 10^{1/2}/p, 10/p\}$ (where the scalar p is the sampling rate) for a rank parameter $R = 14$ and the selected value of λ is $10^{1/2}/p$. All the tested algorithms are terminated if the relative change of the training error attains a given tolerance value:

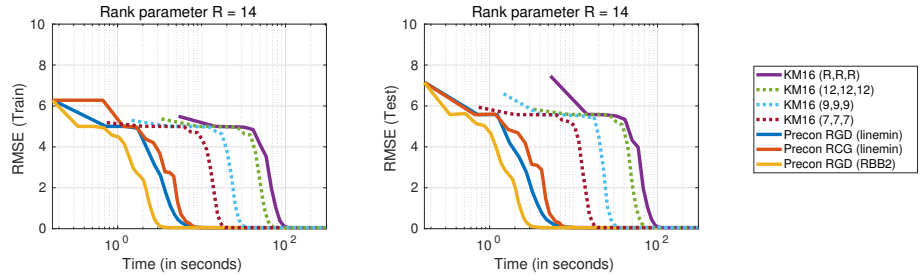
$$\texttt{relchg} = \frac{|E(U^{t+1}) - E(U^t)|}{|E(U^t)|} \leq \texttt{tol}, \tag{5.2}$$

where E denotes the training RMSE. Also, the algorithms terminate if a heuristic time budget $T_{\max}$ is attained. We set the tolerance parameter as $\texttt{tol} = 10^{-6}$ and the time budget as $T_{\max} = 300$s (seconds).

The performances of the tested algorithms are shown in Figure 4. From these results, we have similar observations as in the previous experiments: (i) For a polyadic decomposition rank R that is larger than $\max(r_1^\star, r_2^\star, r_3^\star)$, all algorithms achieve a



(a) Training RMSEs of all algorithms (b) Test RMSEs of all algorithms



(d) Comparison with KM16 (e) Comparison with KM16

FIG. 4. *Tensor completion from noisy observations. The size of* $\mathcal{T}^\star$ *is* $(300, 500, 200)$ *with rank* $r^\star = (3, 7, 5)$. *The sampling rate is* 5%. *The rank parameter* $R = 14$.

recovery performance of the same level (with a test RMSE around 0.050); (ii) the proposed algorithms (Precon RGD and Precon RCG) outperform the rest of the algorithms in time with speedups between 4 and 15 times at the several accuracy levels near their respective termination point; (iii) the time efficiency of KM16 (r, r, r) improves significantly when the core tensor dimensions (r, r, r) decrease, but it remains inferior to those of the proposed algorithms; see Figure 4(d)–4(e).

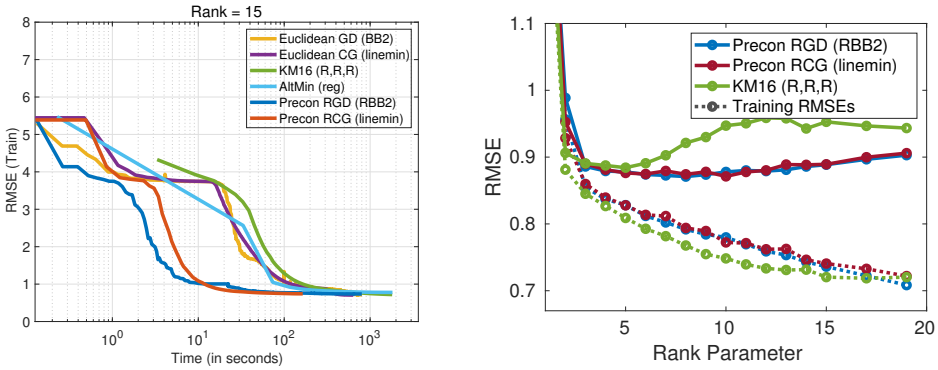
5.2. Real data. In this subsection, we conduct experiments on a real-world dataset. We focus on evaluating the time efficiency of the proposed algorithms under various choices of the rank parameter R. To ensure a good generalization performance of the tensor completion model, we activate the regularization terms of the problem (2.2) with a regularization parameter $\lambda > 0$.

Dataset and algorithms. The tensor completion tests are conducted on the MovieLens 1M dataset,[1] which consists of 1 million movie ratings from 6040 users on 3952 movies in a seven-month period from September 19, 1997, through April 22, 1998. Each movie rating in this dataset has a time stamp, which is the number of week during which a movie rating was given. Therefore, this dataset has a tensor form $\mathcal{T}^\star$ of size $6040 \times 3952 \times 150$, where the first two indices are the user and movie identities and the third-order index is the time stamp. The dataset contains over 10^6 ratings, which correspond to the known entries of the data tensor $\mathcal{T}^\star$. For the tensor completion tasks, we randomly select 80% of the known ratings as the training set. Note that in this case, the absolute sampling rate $p = |\Omega|/(m_1 m_2 m_3)$ is 2.23%.

Due to the large dimensions of the data tensor of MovieLens 1M, several aforementioned algorithms in the related work were not tested on this dataset due to excessive memory requirements. In the implementation of these algorithms, in tensor index filtering operations such as accessing $\mathcal{T}^\star_{|\Omega}$ or $\mathcal{T}_{|\Omega}$ in iterations, a dense tensor format is used, which—for the same size as $\mathcal{T}^\star$—requires the storage of over 3.5×10^9 entries in total. Such a data format poses a memory requirement bottleneck that blocks the test. On the other hand, the proposed algorithms and Euclidean GD/CG, KM16, and AltMin can be run without the memory issue since they use the coordinates (COO) format for the training set Ω, which corresponds to only 10^6 entries on the same dataset. Note that for all tensor decomposition–based algorithms, the memory requirement for the decomposition variables (or factor matrices) is $O((m_1 + m_2 + m_3)R)$—which is bounded by 10^5 for any rank parameter $R < 100$ in the experiments on MovieLens 1M—is also memory-efficient. Therefore, the algorithms that are tested are the proposed algorithms (Precon RGD/RCG), Euclidean GD/CG, KM16, and AltMin.

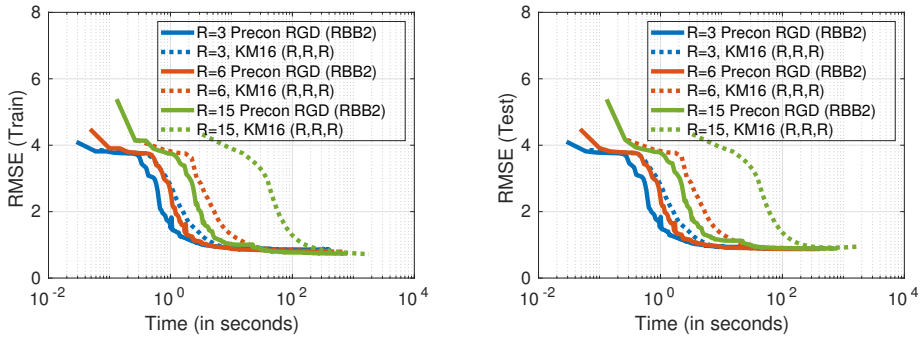
Experiments and results. Given the data tensor $\mathcal{T}^\star$ and the index set Ω as the training set, we conduct performance evaluations using various choices for the rank parameter R, after selection of the regularization parameter λ. The parameter λ for the CPD-based algorithms is selected among $\{0, 1/p, 10^{1/3}/p, 10^{2/3}/p, 10/p, 10^{4/3}/p\}$ via 3-fold cross validation using the Euclidean GD algorithm (instead of the proposed ones), where the rank parameter R is set to be 5, 10, and 15, respectively, and the values selected by these cross validation procedures are $10^{2/3}/p$ (when $R = 5$ and 10) and $10^{4/3}/p$ (when $R = 15$). For the Tucker decomposition–based algorithm—KM16—with the Tucker rank $r = (R, R, R)$, for $R \in \{5, 10, 15\}$, the values of λ selected after the same cross validation procedure are 0. Subsequently, we test all algorithms using the selected parameters. Similar to previous tests, the stopping criteria for all the tested algorithms use the relative change of training errors, i.e.,

[1] https://grouplens.org/datasets/movielens/1m/.



(a) Training RMSEs by time (b) Training and test RMSEs

FIG. 5. *Tensor completion on the MovieLens* 1*M dataset. Recovery performances with various rank parameters and comparisons of the algorithms (with* $R = 15$*) in time.*



(a) Training RMSE (b) Test RMSE

FIG. 6. *Proposed algorithms and KM*16 [25]*. Dataset: MovieLens* 1*M. The rank parameters* R *are* $\{3, 6, 15\}$.

`relchg` in (5.2), and a large enough time budget $T_{\max}$. We set the tolerance parameter for `relchg` (5.2) as `tol` $= 10^{-5}$ and the maximal time budget as $T_{\max} = 1800$s.

We present the iteration histories of all algorithms under the rank parameter $R = 15$ in Figure 5(a). We also observe the recovery performances of the algorithms under a series of different rank parameters—for $R \in \{1, \ldots, 15, 17, 19\}$ and $\lambda = 10^{2/3}/p$; see Figure 5(b).

Moreover, Figure 6 shows the iteration histories of the proposed algorithm Precon RGD (using the Barzilai–Borwein stepsize) in comparison with KM16, with the rank parameters $R \in \{3, 6, 15\}$.

From Figure 5(a), we observe that (i) the two proposed algorithms have faster convergence behaviors than all the rest of the tested algorithms and (ii) in particular, the proposed algorithms achieve the same recovery error with speedups of around 10 times compared to KM16 and 6 times compared to Euclidean CG (linemin). The results in Figure 5(b) give an overview of the performance of our algorithms and KM16 in terms of their trade-offs between the model complexity (for the minimization of the fitting error) and the generalization of the model for the prediction of missing entries. From these results, we can see that (under the several randomly generated

regularization parameters so far explored), the rank choice of $R = 8$ provides the best recovery error on (unknown) test entries. Moreover, for rank choices that are larger than $R = 8$, the decrease in the recovery performances of the proposed algorithms (compared to the case with $R = 8$) is much smaller than that of KM16. This can be explained by the fact that the search space of our algorithms under larger rank parameters contain those with smaller ranks, while the search space of KM16 corresponds to matrix spaces of a fixed column-rank.

6. Conclusion. We proposed a new class of Riemannian preconditioned first-order algorithms for tensor completion through low-rank polyadic decomposition. We have analyzed the convergence properties of RGD using the proposed Riemannian preconditioning. The main feature of the proposed algorithms stems from a new Riemannian metric defined on the product space of the factor matrices of polyadic decomposition. This metric induces a local preconditioning on the Euclidean GD direction of the polyadic decomposition–based objective function; the underlying preconditioner has the form of an approximated inverse of the diagonal blocks of the Hessian of the objective function.

These Riemannian preconditioned algorithms share some characteristics with the related work [25], which deals with tensor completion with a fixed Tucker rank. They differ, however, in the following sense: the polyadic decomposition model allows for finding a low-rank tensor candidate within a range of CP ranks, while the algorithm of [25] searches a tensor solution with a fixed (Tucker) rank.

Because of the more flexible decomposition modeling, our algorithms perform well with various arbitrary choices of the rank parameter in the tensor completion tasks on both synthetic and MovieLens 1M datasets. Moreover, we have observed that the proposed algorithms provide significant speedup over several state-of-the-art algorithms for CPD-based tensor completion while providing comparable or better tensor recovery quality.

Appendix A. Algorithmic details.

Speedup by using Sparse MTTKRP (Algorithm 3.2*).* Algorithm 3.2 is implemented in a mexfunction and has shown significant speedup over the so-far-implemented computations (explicit sparse matricizations times the explicitly computed Khatri–Rao products). Table 3 shows comparative results on the MovieLens 1M dataset. "Naive" corresponds to the implementation where the gradient computations involve (i) forming sparse matricizations of the residual tensor, (ii) computing the Khatri–Rao products, and (iii) sparse-dense matrix multiplication. "Proposed" corresponds to the results of the implementation using sparse MTTKRP (Algorithm 3.2).

Performance of the three stepsize methods. We compare the performances of the three stepsize methods (in section 3.2) in the same experimental settings as in section 5.1. The three stepsize methods for the proposed algorithms (Precon RGD and

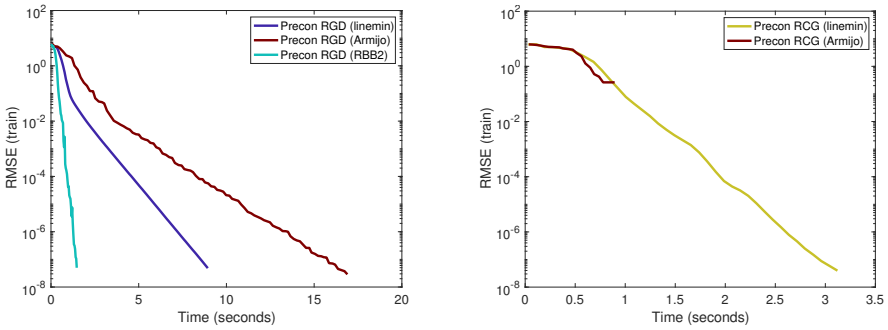
TABLE 3
Speedups of the efficient computational method. The tensor dimensions are $6040 \times 3952 \times 150$.

Iter	Time (s)		Average	RMSE
	Naive	Proposed	speedup	Naive/Proposed
1	0.568	0.067	–	4.778 / 4.778
101	71.848	16.199	4.435×	0.795 / 0.795
201	142.697	32.333	4.413×	0.765 / 0.765
301	213.897	48.508	4.410×	0.759 / 0.759
401	285.117	64.609	4.413×	0.759 / 0.759
501	356.405	80.712	4.416×	0.759 / 0.759

TABLE 4

Comparison of the three stepsize methods under the same setting as in section 5.1. *The time (s) and RMSE scores at termination of each algorithm are the average values over the results of* 20 *random tests. The size of* $\mathcal{T}^\star$ *is* $(100, 100, 200)$ *with a Tucker rank* $r^\star = (3, 5, 7)$. *The sampling rate is* 0.3. *The Armijo rule uses default settings in* [9].

	Stepsize rule	R	#Succ/20	Time (s)	RMSE (test)	RMSE (train)
Precon RGD	(Armijo)	14	20/20	12.176	1.8932e-08	1.8438e-08
	(linemin)	14	20/20	6.169	2.7023e-08	3.2530e-08
	(RBB2)	14	20/20	1.468	2.5175e-08	7.9817e-08
Precon RCG	(Armijo)	14	18/20	2.630	1.3037e-02	1.3139e-02
	(linemin)	14	20/20	3.919	1.6501e-08	2.9628e-08



(a) With Precon RGD (b) With Precon RCG

FIG. 7. *Comparisons between the stepsize selection methods. Tensor size* $(100, 100, 200)$, *Tucker rank* $r^\star = (3, 5, 7)$. *The rank parameter* $R = 14$. *The sampling rate is set to* 0.3.

RCG) are tested in tensor completion tasks with sufficiently large sampling rate. In Table 4, the time (s) and RMSE scores of each method correspond to the average of their respective values (at termination) after 20 random tests. Table 4 shows that (i) for Precon RGD, all three stepsize methods yield successful recovery results in terms of the average test RMSE, but linemin and RBB2 are faster than Armijo, and (ii) for Precon RCG, the percentage of successful recoveries (#Succ/20) with Armijo is inferior to those with linemin and RBB2. In particular, we show the iteration history of one failed attempt of Precon RCG (Armijo) in Figure 7(b), in which the algorithm stagnates prematurely. Therefore, for better practical performance in the experiment of section 5.1, we favor the choice of linemin and RBB2 for the computation of the stepsizes.

Appendix B. Experimental details.

Synthetic model. The noise level in the synthetic tensor model (5.1) is determined according to the signal-to-noise ratio (SNR): $\text{SNR} = \mathbb{E}[T^2]/\mathbb{E}[E^2]$, where T and E are the random variables that represent the tensor entries of the low-rank tensor $\mathcal{T}$ and the noise tensor $\mathcal{E}$ in (5.1). In the experiments with $T \sim \mathcal{N}(0, 1)$ and $E \sim \mathcal{N}(0, \sigma_N)$, the parameter σ_N is computed for a given SNR. The SNR expressed with the logarithmic decibel scale (dB) is defined as $\text{SNR (dB)} = 10 \log_{10}(\text{SNR})$.

Initialization. The initial point of all CPD- and polyadic decomposition–based algorithms is a tuple $U_0 = (U_0^{(1)}, \ldots, U_0^{(k)})$, where the $m_i \times R$ factor matrices are random Gaussian matrices: $[U_0^{(i)}]_{\ell r} \sim \mathcal{N}(0, 1)$. For the Tucker decomposition–based algorithm (KM16), we choose to construct an initial point that is close enough to $U_0 \in \mathcal{M}$ for fair comparisons. For a Tucker rank $(r_1, \ldots, r_k)$, we initialize KM16

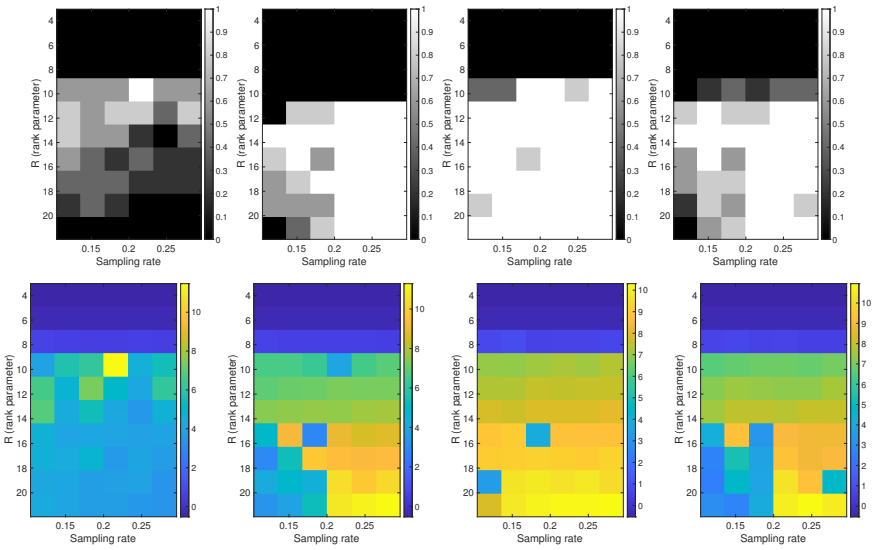
$(r_1, \ldots, r_k)$ with a point in Tucker decomposition form $(G; \tilde{U}^{(1)}, \ldots, \tilde{U}^{(k)})$ such that its tensor representation is close enough to $[\![U_0^{(1)}, \ldots, U_0^{(k)}]\!]$. For this purpose, we set $\tilde{U}_0^{(i)}$ as random Gaussian matrices of size $m_i \times r_i$ with $\tilde{U}_0^{(i)} \sim \mathcal{N}(0,1)$ and set the core tensor G of size $r_1 \times \cdots \times r_k$ as a random Gaussian matrix with $G_{i_1,\ldots,i_k} \sim \mathcal{N}(0, \sigma)$, where $\sigma = \sqrt{R/r_1 \ldots r_k}$. The choice of this variance parameter is based on the observation that the CPD form can be seen as a special Tucker with diagonal core tensor $D = \mathrm{diag}(1, \ldots, 1) \in \mathbb{R}^{R \times \ldots \times R}$. Restricting G to have the same Frobenius norm as D requires that the variance parameter $\sigma = \sqrt{R/r_1 \ldots r_k}$. In particular, in the case whre $r_i = R$, we set $\tilde{U}_0^{(i)} = U_0^{(i)}$ for $i = 1, \ldots, k$.

Performance evaluation. In the experiments, we evaluate the quality of tensor completion with the RMSE for a tensor candidate $\mathcal{T}$ and a given index set Ω', $\mathrm{RMSE}(\Omega') = \|\mathcal{P}_{\Omega'}(\mathcal{T} - \mathcal{T}^\star)\|_{\mathrm{F}} / \sqrt{|\Omega'|}$. The training and test RMSE refer to $\mathrm{RMSE}(\Omega)$ and $\mathrm{RMSE}(\tilde{\Omega}^c)$, respectively, where Ω is the (training) index set of the observed entries used in the definition of the data fitting function f_Ω in (2.2) and the test set $\tilde{\Omega}^c$ is the complementary of Ω in the set of all available entries. In some of the experiments, $\tilde{\Omega}^c$ is a subset of the complementary set (with uniformly distributed indices) such that $|\tilde{\Omega}^c| = (1/4)|\Omega|$ in order to reduce the time for evaluating the test RMSE if the whole complementary set is overwhelmingly large (2.10^6).

Appendix C. Supplementary experiments.

Tensor recovery performances. In addition to the tensor completion results shown in section 5.1, we conducted tensor completion tasks on a 6×10 grid of $(p, R) \in \{0.12, 0.152, \ldots, 0.28\} \times \{4, 6, \ldots, 16, 17, 19, 21\}$ for 5 random runs at each (p, R) (each run is based on a randomly generated index set Ω).

Figure 8 shows the average final RMSEs of the solutions given by the algorithms and the corresponding rate of successful recoveries among the 5 random runs. Each



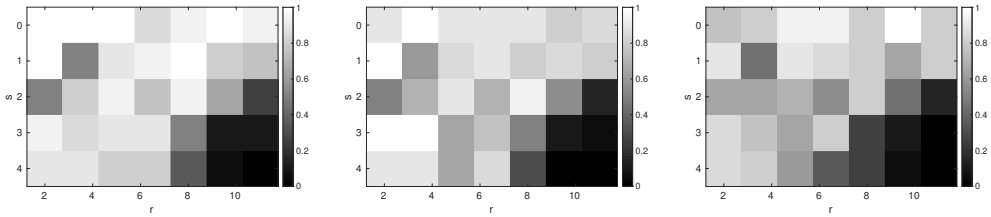
(a) Euclidean CG (b) PreconRGD (lmin) (c) PreconRCG (lmin) (d) PreconRGD (rbb2)

FIG. 8. *Tensor completion performances based on different sampling rates and rank parameters. Top: Rate of successful recoveries. Bottom: Average errors in* $-\log_{10}$*(RMSE). The size of* $\mathcal{T}^\star$ *is* $(100, 100, 200)$ *with a Tucker rank* $r^\star = (3, 5, 7)$. *(lmin) is short for (linemin) in some of the algorithm descriptors above.*

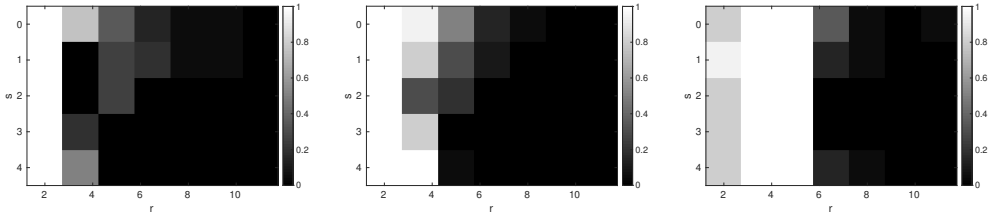
solution is counted as a successful recovery if its RMSE is lower than 10^{-7}. In Figure 8, we observe, for each algorithm tested, a clear enough phase transition of recovery rates from the unsuccessful regime to the successful regime, when the rank parameter R increases. We observe that the minimal value of R (for all these algorithms) to ensure a good chance of exact recovery is around $R = 10$, which corresponds to a search space slightly larger but close enough in dimensionality to the tensor space with a Tucker rank $\text{rank}_{\text{tc}}(\mathcal{T}^\star) = (3, 5, 7)$.

Tensor approximation. In addition to tensor completion tasks, we investigate the performance of the proposed algorithms for low-rank CPD and compare them with a low-rank CPD algorithm using the RGN method [10].

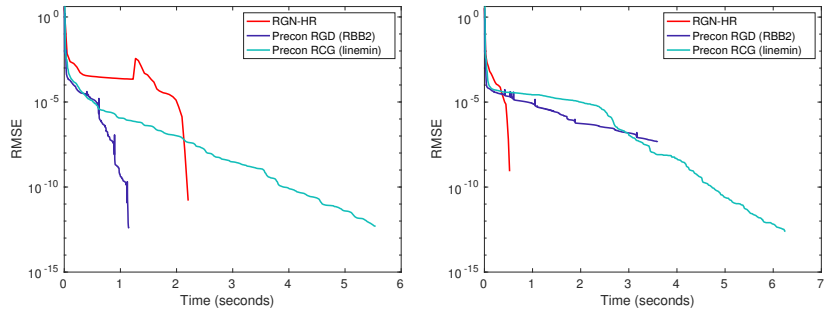
The tensor approximation task here refers to the problem of finding the best approximation to a given, fully observed tensor by a tensor of bounded CP rank, which is a special case of (1.1) with the sampling operator reduced to identity (where Ω is the full index set). The performance of these algorithms in the tensor approximation tasks is evaluated as follows: (i) Randomly sample kth-order low-rank decomposition from



(a) RGN-HR



(b) Precon RGD (RBB2)



(c) $(s = 0,\ r = 9)$

(d) $(s = 1,\ r = 9)$

FIG. 9. *Tensor approximation performances on tensors $\mathcal{T}^\star$ generated from the class $\mathcal{G}_r(s)$.* (a): *Rate of successful attempts by RGN-HR using $R = \lfloor 1.5r \rfloor$, $R = 2r$, and $R = 3r$ (from left to right), respectively.* (b): *Rate of successful attempts by Precon RGD (RBB2) using the same choices of R.* (c-d): *Iteration histories randomly picked from successful attempts.*

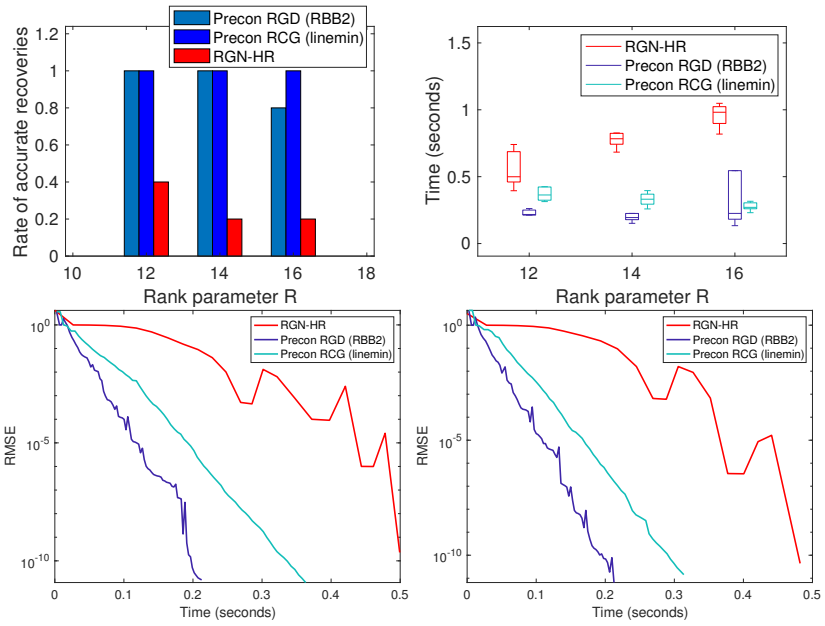


FIG. 10. *The tensor $\mathcal{T}^\star$ is of size $10 \times 30 \times 30$ and has a Tucker rank $r^\star = (3, 5, 7)$, which is generated in the same manner as in the experiment of Figure 2.*

the model (5.1) with a low Tucker rank and Model 2 in [10, section 7.4], which consists of generating $A = (A^{(1)}, \ldots, A^{(k)}) \in \mathcal{G}_r(s)$ for a low CP rank r and $s \in \{0, 1, 2, 3, 4\}$ and letting $\mathcal{A} := [\![A^{(1)}, \ldots, A^{(k)}]\!]$; (ii) create a perturbed tensor $\mathcal{T}^\star := \frac{\mathcal{A}}{\|\mathcal{A}\|} + 10^{-e}\frac{E}{\|E\|}$, where all entries of $E \sim \mathcal{N}(0, 1)$ and $e > 0$; (iii) solve the tenso-approximation of $\mathcal{T}^\star$ with a rank parameter $R \geq r$, from a random starting point q using each algorithm, where $q = (Q^{(1)}, \ldots, Q^{(k)}) \in \mathcal{M} = R^{m_1 \times R} \times \cdots \times R^{m_k \times R}$ with all entries of $Q^{(i)} \sim \mathcal{N}(0, 1)$.

We compare the performances of Precon RGD (RBB2) and Precon RCG (linemin) with Riemannian Gauss–Newton using Hot Restart (RGN-HR) [10] under the two models described above. For the proposed algorithms, we set $\delta = 10^{-15}$. In the tests with Model 2 of [10, section 7.4], we explore (r, s) on a 7×5 grid of $\{2, 3, 4, 5, 7, 9, 11\} \times \{0, 1, 2, 3, 4\}$ and test with the CP rank parameter $R \in \{r, \lfloor 1.5r \rfloor, 2r, 3r\}$. The chances of successful recovery are estimated after 20 random runs for each (s, r) setting and each value of R. The comparative results on these two models are given in Figures 9–10, respectively.

From Figure 9, we observe that RGN-HR performs better than the proposed algorithms in a large part of the 7×5 grid of (r, s), where the tensors $\mathcal{T}^\star$ under Model 2 of [10, section 7.4] are challenging because of their high condition number. From Figure 10, we observe that the proposed algorithms outperform RGN-HR in both successful recovery rate and convergence time for tensors of the model (5.1), where the core tensor of $\mathcal{T}^\star$ is not cubic.

Acknowledgments. We thank the referees for their insightful comments. We gratefully acknowledge the helpful discussions with P.-A. Absil and his valuable comments during the preparation of this paper.

REFERENCES

[1] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, Princeton, NJ, 2008.

[2] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, *Scalable tensor factorizations for incomplete data*, Chemometr. Intell. Lab. Syst., 106 (2011), pp. 41–56.

[3] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, *Tensor decompositions for learning latent variable models*, J. Mach. Learn. Res., 15 (2014), pp. 2773–2832.

[4] C. A. Andersson and R. Bro, *Improving the speed of multi-way algorithms: Part* I. *Tucker*3, Chemometr. Intell. Lab. Syst., 42 (1998), pp. 93–103.

[5] B. W. Bader and T. G. Kolda, *MATLAB Tensor Toolbox Version* 3.1, available online, June 2019.

[6] D. Banco, S. Aeron, and W. S. Hoge, *Sampling and recovery of MRI data using low rank tensor models*, in Proceedings of the 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), IEEE, 2016, pp. 448–452.

[7] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, *Image inpainting*, in Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 417–424.

[8] N. Boumal, P. A. Absil, and C. Cartis, *Global rates of convergence for nonconvex optimization on manifolds*, IMA J. Numer. Anal., 39 (2019), pp. 1–33.

[9] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, *Manopt, a Matlab toolbox for optimization on manifolds*, J. Mach. Learn. Res., 15 (2014), pp. 1455–1459.

[10] P. Breiding and N. Vannieuwenhoven, *A Riemannian trust region method for the canonical tensor rank approximation problem*, SIAM J. Optim., 28 (2018), pp. 2435–2465.

[11] W. Chu and Z. Ghahramani, *Probabilistic models for incomplete multi-dimensional arrays*, in Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, 2009, pp. 89–96.

[12] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, *Tensor decompositions for signal processing applications: From two-way to multiway component analysis*, IEEE Signal Process. Mag., 32 (2015), pp. 145–163.

[13] C. Da Silva and F. Herrmann, *Hierarchical Tucker tensor optimization—applications to tensor completion*, in Proceedings of the 10th International Conference on Sampling Theory and Application, Jacobs University Bremen, 2013.

[14] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.

[15] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-*1 *and rank-*$(R_1, R_2, \ldots, R_N)$ *approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[16] L. Grasedyck, M. Kluge, and S. Krämer, *Variants of alternating least squares tensor completion in the tensor train format*, SIAM J. Sci. Comput., 2015 (37), pp. A2424–A2450.

[17] Y. Guan, S. Dong, P.-A. Absil, and F. Glineur, *Alternating Minimization Algorithms for Graph Regularized Tensor Completion*, arXiv preprint, arXiv:2008.12876 [math. NA], 2020, pp. 1–30.

[18] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer Ser. Comput. Math. 42, Springer, Cham, 2012.

[19] D. Hernandez, *Simple tensor products*, Invent. Math., 181 (2010), pp. 649–675.

[20] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 409–436.

[21] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189.

[22] B. Iannazzo and M. Porcelli, *The Riemannian Barzilai–Borwein method with nonmonotone line search and the matrix geometric mean computation*, IMA J. Numer. Anal., 38 (2018), pp. 495–517.

[23] P. Jain and S. Oh, *Provable tensor factorization with missing data*, Adv. Neural. Inf. Process. Syst., 2014, pp. 1431–1439.

[24] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, *Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering*, in Proceedings of the Fourth ACM Conference on Recommender Systems, ACM, 2010, pp. 79–86.

[25] H. Kasai and B. Mishra, *Low-rank tensor completion: A Riemannian manifold preconditioning approach*, in Proceedings of the International Conference on Machine Learning, 2016, pp. 1012–1021.

[26] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[27] T. Korah and C. Rasmussen, *Spatiotemporal inpainting for recovering texture maps of occluded building facades*, IEEE Trans. Image. Process., 16 (2007), pp. 2262–2271.
[28] D. Kressner, M. Steinlechner, and B. Vandereycken, *Low-rank tensor completion by riemannian optimization*, BIT, 54 (2014), pp. 447–468.
[29] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95–138.
[30] J. B. Kruskal, *Rank, decomposition, and uniqueness for* 3*-way and n-way arrays*, in Multiway Data Analysis, Elsevier, Amsterdam, 1989, pp. 7–18.
[31] J. Liu, P. Musialski, P. Wonka, and J. Ye, *Tensor completion for estimating missing values in visual data*, IEEE Trans. Pattern. Anal. Mach. Intell., 35 (2012), pp. 208–220.
[32] Y. Liu, F. Shang, H. Cheng, J. Cheng, and H. Tong, *Factor matrix trace norm minimization for low-rank tensor completion*, in Proceedings of the SIAM International Conference on Data Mining, SIAM, 2014, pp. 866–874.
[33] B. Mishra, K. Adithya, and A. R. Sepulchre, *A Riemannian Geometry for Low-Rank Matrix Completion*, arXiv preprint, arXiv:1211.1550 [CS. LG], 2012.
[34] B. Mishra and R. Sepulchre, *Riemannian preconditioning*, SIAM J. Optim., 26 (2016), pp. 635–660.
[35] M. Mørup, L. K. Hansen, C. S. Herrmann, J. Parnas, and S. M. Arnfred, *Parallel factor analysis as an exploratory tool for wavelet transformed event-related EEG*, NeuroImage, 29 (2006), pp. 938–947.
[36] J. Nocedal and S. Wright, *Numerical Optimization*, Springer Science & Business Media, New York, 2006.
[37] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.
[38] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, *Tensors for data mining and data fusion: Models, applications, and scalable algorithms*, ACM Trans. Intell. Syst. Technol., 8 (2016), pp. 1–44.
[39] H. N. Phien, H. D. Tuan, J. A. Bengua, and M. N. Do, *Efficient Tensor Completion: Low-Rank Tensor Train*, arXiv preprint, arXiv:1601.01083 [CS. NA], 2016.
[40] H. Rauhut, R. Schneider, and Ž. Stojanac, *Tensor completion in hierarchical tensor representations*, in Compressed Sensing and Its Applications, Springer, Cham, 2015, pp. 419–450.
[41] H. Rauhut, R. Schneider, and Ž. Stojanac, *Low rank tensor recovery via iterative hard thresholding*, Linear Algebra Appl., 523 (2017), pp. 220–262.
[42] R. Schneider and A. Uschmajew, *Convergence results for projected line-search methods on varieties of low-rank matrices via Łojasiewicz inequality*, SIAM J. Optim., 25 (2015), pp. 622–646.
[43] L. Sorber, M. Van Barel, and L. De Lathauwer, *Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-(l_r,l_r,*1*) terms, and a new generalization*, SIAM J. Optim., 23 (2013), pp. 695–720.
[44] L. Sorber, M. Van Barel, and L. De Lathauwer, *Structured data fusion*, IEEE J. Sel. Top. Signal Process., 9 (2015), pp. 586–600.
[45] M. Sørensen and L. De Lathauwer, *Fiber sampling approach to canonical polyadic decomposition and application to tensor completion*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 888–917.
[46] N. Srebro, J. D. Rennie, and T. S. Jaakkola, *Maximum-margin matrix factorization*, Adv. Neural. Inf. Process. Syst., 17 (2005), pp. 1329–1336.
[47] G. Tomasi and R. Bro, *PARAFAC and missing values*, Chemometr. Intell. Lab. Syst., 75 (2005), pp. 163–180.
[48] L. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.
[49] N. Vervliet, O. Debals, L. Sorber, M. V. Barel, and L. D. Lathauwer, *Tensorlab* 3.0, available online, Mar. 2016.
[50] T. Yokota, Q. Zhao, and A. Cichocki, *Smooth PARAFAC decomposition for tensor completion*, IEEE Trans. Signal Process., 64 (2016), pp. 5423–5436.